9.4

Développement d'applications pour IBM MQ

Remarque

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section «Remarques», à la page 1331.

Cette édition s'applique à la version 9 édition 4 d' IBM® MQ et à toutes les éditions et modifications ultérieures, sauf indication contraire dans les nouvelles éditions.

Table des matières

Développement d'applications pour IBM MQ

Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

Vous découvrez comment développer des applications pour IBM MQ?

Pour en savoir plus sur le développement d'applications pour IBM MQ, visitez le site IBM Developer:

- IBM MQ Developer Essentials *(apprendre les bases, exécuter une démonstration, coder une application, suivre des tutoriels plus avancés)*
- IBM MQ Downloads for Developers *(y compris les éditions gratuites pour développeurs et les versions d'essai)*

Vous pouvez également trouver plus facile de développer vos applications si vous connaissez les concepts décrits dans les sections suivantes:

- «Concepts de développement d'applications», à la page 7
- «Remarques sur la conception des applications IBM MQ», à la page 52

Prise en charge des langages et des infrastructures orientés objet

IBM MQ fournit une prise en charge de base pour les applications développées dans les langages et infrastructures suivants:

- JMS
- Java
- C++
- .NET

Voir aussi «Applications orientées objet», à la page 16.

.NET prend en charge les applications développées dans de nombreuses langues. Pour illustrer l'utilisation des classes IBM MQ pour .NET afin d'accéder aux files d'attente IBM MQ , la documentation du produit MQ contient des informations pour les langues suivantes:

- C# exemple de code et exemples d'applications
- Exemples d'applications C++
- Exemples d'applicationVisual Basic

Voir «Ecriture et déploiement de programmes IBM MQ .NET», à la page 592.

IBM MQ prend en charge .NET Core pour les applications dans les environnements Windows à partir de IBM MQ 9.1.1 et pour les applications dans les environnements Linux® à partir de IBM MQ 9.1.2. Pour plus d'informations, voir «Installation de IBM MQ classes for .NET», à la page 569.

IBM MQ prend également en charge les clients AMQP qui implémentent le protocole OASIS AMQP 1.0 .

MQ Light, Apache Les clients Qpid tels que les API Apache Qpid Proton et Apache Qpid JMS sont basés sur ce protocole.

Les API MQ Light sont disponibles à l'adresse IBM MQ Light.

Les clients Apache Qpid sont disponibles à l'adresse QPid Proton.

Les liaisons de langage suivantes sont fournies telles qu'elles sont:

- une liaison Go

- une implémentation d'API JavaScript qui fonctionne avec des applications Node.js

Prise en charge des API REST par programmation

IBM MQ prend en charge les API REST de programmation suivantes pour l'envoi et la réception de messages:

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

Voir «Développement d'applications REST avec IBM MQ», à la page 719, ainsi que le tutoriel Initiation à l'API REST de messagerie IBM MQ dans la zone IBM MQ d' IBM Developer. Ce tutoriel inclut des exemples dans les langages suivants, fournis en l'état, à utiliser avec IBM MQ messaging REST API:

- Exemple d'utilisation de l'API REST de messagerie MQ
- Exemple de fichier Node.js utilisant le module HTTPS
- Exemple Node.js avec le module Promise

Prise en charge des langages de programmation procéduraux

IBM MQ prend en charge les applications développées dans les langages de programmation procédurale suivants:

- C
- Windows Visual Basic (systèmesWindows uniquement)
- COBOL
- z/OS Assembleur (IBM MQ for z/OS uniquement)
- z/OS PL/I (IBM MQ for z/OS uniquement)
- IBM i RPG (IBM MQ for IBM i uniquement)

Ces langues utilisent l'interface de file d'attente de messages (MQI) pour accéder aux services de mise en file d'attente de messages. Voir «Développement d'applications MQI avec IBM MQ», à la page 734. Notez que le modèle d'objet IBM MQ , utilisé par les langages orientés objet et les infrastructures, fournit des fonctions supplémentaires qui ne sont pas disponibles pour les langages procéduraux utilisant l'interface MQI.

Spécification du nom de l'application

ALW

Avant IBM MQ 9.1.2, vous pouviez spécifier un nom d'application sur les applications client Java ou JMS . Depuis la IBM MQ 9.1.2, vous pouvez également spécifier le nom de l'application sur des langages de programmation supplémentaires. Pour plus d'informations, voir «Spécification du nom d'application dans les langages de programmation pris en charge», à la page 55.

Tâches associées

«Développement d'applications pour MQ Telemetry», à la page 1274

«Développement d'applications Microsoft Windows Communication Foundation avec IBM MQ», à la page 1299

Le canal personnalisé Microsoft Windows Communication Foundation (WCF) pour IBM MQ envoie et reçoit des messages entre les clients et les services WCF.

Référence associée

«Développement d'applications pour Managed File Transfer», à la page 1252

Spécifiez les programmes à exécuter avec Managed File Transfer, utilisez Apache Ant avec Managed File Transfer, personnalisez Managed File Transfer avec des exits utilisateur et contrôlez Managed File Transfer en plaçant les messages dans la file d'attente de commandes de l'agent.

Concepts de développement d'applications

Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

Pour plus d'informations sur les types d'application que vous pouvez écrire pour IBM MQ, voir «Développement d'applications pour IBM MQ», à la page 5 et «Actions que vos applications peuvent effectuer», à la page 7.

Concepts associés

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

Actions que vos applications peuvent effectuer

Vous pouvez développer des applications pour envoyer et recevoir des messages dont vous avez besoin pour prendre en charge vos processus métier. Vous pouvez également développer des applications pour gérer vos gestionnaires de files d'attente et les ressources associées.

Actions que vos applications peuvent effectuer sur IBM MQ for Multiplatforms

Multi

Sous Multiplateformes, vous pouvez écrire des applications qui effectuent les actions suivantes:

- Envoyez des messages à d'autres applications s'exécutant sous les mêmes systèmes d'exploitation. Les applications peuvent se trouver sur le même système ou sur un autre système.
- Envoyez des messages aux applications qui s'exécutent sur d'autres plateformes IBM MQ .
- Utilisez la mise en file d'attente de messages depuis CICS pour les systèmes suivants:
  - AIX TXSeries pour AIX
  - IBM i IBM i
  - Windows Windows
- Utilisez la mise en file d'attente de messages depuis Encina pour les systèmes suivants:
  - AIX AIX
  - Windows Windows
- Utilisez la mise en file d'attente de messages depuis Tuxedo pour les systèmes suivants:
  - AIX AIX
  - AT&T
  - Windows Windows
- Utilisez IBM MQ comme gestionnaire de transactions, en coordonnant les mises à jour effectuées par les gestionnaires de ressources externes au sein des unités d'oeuvre IBM MQ . Les gestionnaires de ressources externes suivants sont pris en charge et sont conformes à l'interface X/OPEN XA
  - Db2
  - Informix

  - Oracle
  - Sybase
- Traitez plusieurs messages ensemble en tant qu'unité de travail unique pouvant être validée ou annulée.
- Exécuter à partir d'un environnement IBM MQ complet ou à partir d'un environnement client IBM MQ .

Actions que vos applications peuvent effectuer sur IBM MQ for z/OS

Sous z/OS, vous pouvez écrire des applications qui effectuent les actions suivantes:

- Utilisez la mise en file d'attente de messages dans CICS ou IMS.
- Envoyez des messages entre les applications par lots, CICSet IMS en sélectionnant l'environnement le plus approprié pour chaque fonction.
- Envoyez des messages aux applications qui s'exécutent sur d'autres plateformes IBM MQ .
- Traitez plusieurs messages ensemble en tant qu'unité de travail unique pouvant être validée ou annulée.
- Envoyez des messages aux applications IMS et interagissez avec elles à l'aide de la passerelle IMS .
- Participer aux unités de travail coordonnées par RRS.

Chaque environnement dans z/OS possède ses propres caractéristiques, avantages et inconvénients. L'avantage de IBM MQ for z/OS est que les applications ne sont liées à aucun environnement, mais qu'elles peuvent être distribuées pour tirer parti des avantages de chaque environnement. Par exemple, vous pouvez développer des interfaces utilisateur final à l'aide de TSO ou de CICS, exécuter des modules à traitement intensif dans z/OS Batch et exécuter des applications de base de données dans IMS ou CICS. Dans tous les cas, les différentes parties de l'application peuvent communiquer à l'aide de messages et de files d'attente.

Les concepteurs des applications IBM MQ doivent être conscients des différences et des limitations imposées par ces environnements. Exemple :

- IBM MQ fournit des fonctions qui permettent l'intercommunication entre les gestionnaires de files d'attente (c'est ce qu'on appelle la *mise en file d'attente répartie*).
- Les méthodes de validation et d'annulation des modifications diffèrent entre les environnements de traitement par lots et CICS .
- IBM MQ for z/OS fournit une prise en charge dans l'environnement IMS pour les programmes de traitement de messages en ligne (MPP), les programmes Fast Path interactifs (IFP) et les programmes de traitement de messages par lots (BMP). Si vous écrivez des programmes par lots DL/I, suivez les instructions fournies dans les rubriques telles que «Building z/OS batch applications», à la page 1051 et «z/OS batch considerations», à la page 749 pour les programmes par lots z/OS .
- Bien que plusieurs instances de IBM MQ for z/OS puissent exister sur un seul système z/OS , une région CICS ne peut se connecter qu'à un seul gestionnaire de files d'attente à la fois. Toutefois, plusieurs régions CICS peuvent être connectées au même gestionnaire de files d'attente. Dans les environnements de traitement par lots IMS et z/OS , les programmes peuvent se connecter à plusieurs gestionnaires de files d'attente.
- IBM MQ for z/OS permet aux files d'attente locales d'être partagées par un groupe de gestionnaires de files d'attente, ce qui améliore le débit et la disponibilité. Ces files d'attente sont appelées *files d'attente partagées*et les gestionnaires de files d'attente forment un *groupe de partage de files d'attente*, qui peut traiter des messages dans les mêmes files d'attente partagées. Les applications par lots peuvent se connecter à l'un des plusieurs gestionnaires de files d'attente au sein d'un groupe de partage de files d'attente en spécifiant le nom du groupe de partage de files d'attente au lieu d'un nom de gestionnaire de files d'attente particulier. C'est ce que l'on appelle la *liaison par lots de groupe*, ou plus simplement la *liaison de groupe*. Voir Files d'attente partagées et groupes de partage de files d'attente.

z/OS Les différences entre les environnements pris en charge et leurs limitations sont expliquées plus en détail dans «Using and writing applications on IBM MQ for z/OS», à la page 914.

Concepts associés

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

«Ecriture d'une application de procédure pour la mise en file d'attente», à la page 738
La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

«Ecriture d'applications de procédure client», à la page 937
Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85
IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

«Utilisation IBM MQ classes for Java», à la page 356
Utilisez IBM MQ dans un environnement Java . IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

«Développement d'applications C++», à la page 539
IBM MQ fournit des classes C++ équivalentes à des objets IBM MQ et des classes supplémentaires équivalentes aux types de données de tableau. Il fournit un certain nombre de fonctions qui ne sont pas disponibles via l'interface MQI.

«Création d'une application procédurale», à la page 1027
Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

Tâches associées

«Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085
Ces exemples de programmes sont écrits dans des langages procéduraux et présentent des utilisations typiques de l'interface MQI (Message Queue Interface). Programmes IBM MQ sur différentes plateformes.

«Développement d'applications .NET», à la page 567
Les applications IBM MQ classes for .NET permettent aux applications .NET de se connecter à IBM MQ en tant que IBM MQ MQI client ou de se connecter directement à un serveur IBM MQ .

«Développement d'applications Microsoft Windows Communication Foundation avec IBM MQ», à la page 1299
Le canal personnalisé Microsoft Windows Communication Foundation (WCF) pour IBM MQ envoie et reçoit des messages entre les clients et les services WCF.

Sécurisation

Multi Applications, noms d'application et instances d'application

Avant de commencer à concevoir et à écrire vos applications, familiarisez-vous avec les concepts de base des applications, des noms d'application et des instances d'application.

Applications

Multi

Les connexions à un gestionnaire de files d'attente sont considérées comme provenant de la même *application* si elles fournissent le même *nom d'application*. Le nom de l'application s'affiche sous la forme de l'attribut APPLTAG de la commande DISPLAY CONN (*) TYPE CONN.

Remarques :

1. Pour les applications utilisant une version de IBM MQ client antérieure à IBM MQ 9.1.2, le nom de l'application est automatiquement défini par IBM MQ client. Sa valeur dépend du langage de programmation d'application et de la plateforme sur laquelle l'application s'exécute. Pour plus d'informations, voir PutApplName .
2. Pour les applications IBM MQ client utilisant un IBM MQ client à la IBM MQ 9.1.2 ou une version ultérieure, il est possible de définir le nom de l'application sur une valeur spécifique. Dans la plupart des cas, il n'est pas nécessaire de modifier le code de l'application ni de recompiler l'application. Pour plus d'informations, voir «Utilisation du nom d'application dans les langages de programmation pris en charge», à la page 56.

Instances d'application

Multi

Les connexions sont subdivisées en *instances d'application*. Une instance d'une application est un ensemble de connexions étroitement liées qui fournissent une "unité d'exécution" pour cette application. En règle générale, il s'agit d'un processus de système d'exploitation unique, qui peut comporter un certain nombre d'unités d'exécution et des connexions IBM MQ associées.

Sous IBM MQ for Multiplatforms , une instance d'application est associée à une balise de connexionspécifique. Le gestionnaire de files d'attente associe automatiquement de nouvelles connexions à une instance d'application existante, lorsqu'il peut voir qu'elles sont liées.

Remarques :

- Si vous utilisez des connexions client, ces processus peuvent se connecter au gestionnaire de files d'attente via un ou plusieurs canaux en cours d'exécution.
- Dans les applications JMS , une instance d'application est mappée à une connexion JMS spécifique et à toutes les sessions JMS associées.

Les instances d'application sont particulièrement importantes sous IBM MQ for Multiplatforms lors de l'utilisation de l' équilibrage automatique des applicationsde cluster uniforme. Sur les plateformes IBM MQ for Multiplatforms , vous pouvez afficher les instances d'application actuellement connectées à l'aide de la commande DISPLAY APSTATUS .

Dans certains cas, le gestionnaire de files d'attente ne peut pas établir correctement une connexion à une association d'instance d'application, en particulier:

- Si plusieurs connexions sont établies sur une conversation partagée à partir du même processus, en utilisant des noms d'application différents.
- Si des bibliothèques client de niveau plus ancien sont utilisées. Par exemple, les installations du client IBM MQ JMS dans IBM MQ 9.1.2 et les versions antérieures.

Dans ces situations, si les applications ne se définissent pas comme reconnectables, cela sera autorisé, mais certains regroupements d'instances d'application peuvent être incorrects. Si l'une des connexions est déclarée comme MQCNO_RECONNECT, cela a un impact négatif significatif sur l'équilibrage des applications ; l'appel MQCONN sera donc rejeté avec MQCNO_RECONNECT_INCOMPATIBLE.

Concepts associés

«Spécification du nom d'application dans les langages de programmation pris en charge», à la page 55
Avant IBM MQ 9.2.0, vous pouviez déjà spécifier un nom d'application sur les applications client Java ou JMS . Depuis IBM MQ 9.2.0 , cette fonction est étendue à d'autres langages de programmation sous IBM MQ for Multiplatforms.

Programmes d'application utilisant l'interface MQI

Les programmes d'application IBM MQ ont besoin de certains objets pour pouvoir s'exécuter correctement.

La Figure 1, à la page 11 illustre une application qui supprime des messages d'une file d'attente, les traite, puis envoie des résultats à une autre file d'attente du même gestionnaire de files d'attente.

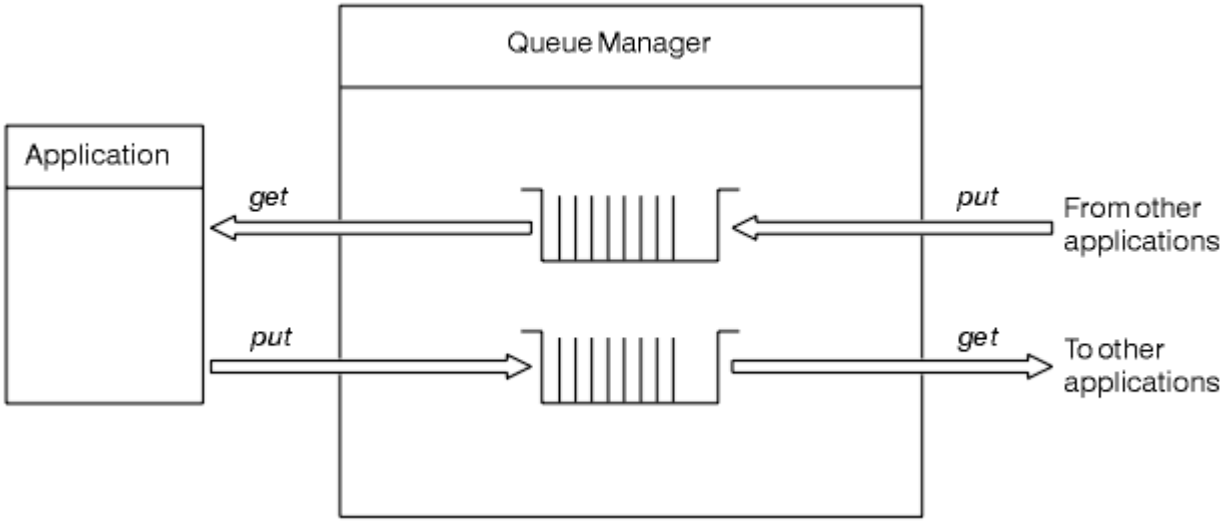


Figure 1. Files d'attente, messages et applications

Alors que les applications peuvent placer des messages dans des files d'attente locales ou distantes (à l'aide de MQPUT), elles ne peuvent extraire des messages que directement des files d'attente locales (à l'aide de MQGET).

Pour que cette application puisse s'exécuter, les conditions suivantes doivent être remplies:

- Le gestionnaire de files d'attente doit exister et être en cours d'exécution.
- La première file d'attente d'application, dont les messages doivent être supprimés, doit être définie.
- La deuxième file d'attente, dans laquelle l'application insère les messages, doit également être définie.
- L'application doit pouvoir se connecter au gestionnaire de files d'attente. Pour ce faire, il doit être lié à IBM MQ. Voir «Création d'une application procédurale», à la page 1027.
- Les applications qui placent les messages dans la première file d'attente doivent également se connecter à un gestionnaire de files d'attente. S'ils sont distants, ils doivent également être configurés avec des files d'attente de transmission et des canaux. Cette partie du système n'est pas illustrée dans la Figure 1, à la page 11.

Utilisation des connexions client pour la connexion à plusieurs gestionnaires de files d'attente IBM MQ

Il est possible de configurer des applications connectées au client pour qu'elles se connectent à plusieurs gestionnaires de files d'attente (pour des raisons d'équilibrage de charge ou de disponibilité des services).

Les principaux mécanismes permettant d'atteindre cet objectif dans le client IBM MQ sont l'utilisation des tables de définition de canal du client, voir Configuration des tables de définition de canal du clientou des listes de connexions.

Il est également possible d'obtenir un comportement similaire à l'aide de produits d'équilibrage de charge externes ou en encapsulant le code de connexion IBM MQ dans un "stub" qui peut rediriger les noms d'hôte ou les adresses IP.

Chacune de ces techniques est fournie avec certaines restrictions et peut être plus ou moins adaptée à des exigences d'application particulières. Les sections suivantes, bien qu'elles ne soient pas exhaustives,

décrivent les aspects particuliers que vous devez prendre en considération, ainsi que l'effet de ces différentes approches sur ces aspects.

IBM MQ Les clusters uniformes, voir A propos des clusters uniformes, fournissent un mécanisme puissant permettant d'effectuer une mise à l'échelle horizontale des applications sur plusieurs gestionnaires de files d'attente en s'appuyant sur le mécanisme de base de la table de définition de canal du client pour fournir plusieurs destinations. Les clusters uniformes peuvent fournir des fonctionnalités au-delà de ce qui est possible à l'aide d'un équilibreur de charge externe ignorant les protocoles IBM MQ sous-jacents et éviter certains des problèmes décrits ci-dessous. Par conséquent, envisagez d'utiliser un cluster uniforme plutôt que d'utiliser d'autres techniques, le cas échéant.

Avertissement : Vous devez utiliser avec précaution les applications utilisant IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, y compris celles utilisant l'un des adaptateurs de ressources IBM MQ , qui se connectent aux gestionnaires de files d'attente à l'aide des technologies d'équilibrage de charge. Si vous rencontrez des problèmes, recréez ces problèmes sans essayer d'utiliser l'équilibrage de charge.

Il y a de multiples questions en jeu, ce qui signifie que ces connexions sont au mieux problématiques et, au pire, tout à fait peu fiables:

- Une attention particulière est requise lors de la connexion d'une application qui établit plusieurs connexions au gestionnaire de files d'attente à l'aide de n'importe quelle forme d'équilibrage de charge. Cela inclut toutes les applications utilisant les classes IBM MQ pour JMS/Jakarta Messaging car elles créent plusieurs connexions IBM MQ d'utilisation générale. Si vous utilisez un équilibreur de charge externe ou un module de remplacement de code personnalisé, vous devez à tout moment acheminer les connexions de la même instance d'application vers le même gestionnaire de files d'attente.
- L'utilisation de la gestion des transactions XA ou de JTA (Java Transaction API) repose sur la possibilité de se connecter de manière cohérente au même gestionnaire de files d'attente. En pratique, il est peu probable que cela soit pratique avec une quelconque forme d'équilibrage de charge.
- -La gestion uniforme des clusters repose sur la possibilité de demander aux clients de se reconnecter à des gestionnaires de files d'attente spécifiques sans interférence. Il n'est pas conseillé de tenter de combiner l'équilibrage de charge externe avec l'utilisation de clusters uniformes

Vous devez utiliser la fonctionnalité de cluster uniforme IBM MQ pour obtenir une mise à l'échelle horizontale des applications sur plusieurs gestionnaires de files d'attente, plutôt que des technologies d'équilibrage de charge externes. Voir Configuration d'un cluster uniforme, ainsi que les rubriques suivantes, pour plus d'informations sur les clusters uniformes, y compris sur la manière de créer et d'utiliser des clusters uniformes.

Termes utilisés dans ces informations

CCDT-MultiQMGR

Fichier CCDT contenant plusieurs canaux de connexion client (CLNTCONN) avec le même groupe, c'est-à-dire l'attribut de connexion client du nom du gestionnaire de files d'attente (QMNAME CLNTCONN), où différentes entrées CLNTCONN sont résolues dans des gestionnaires de files d'attente différents.

Il est distinct d'un fichier CCDT qui contient plusieurs entrées CLNTCONN qui sont simplement des adresses IP ou des noms d'hôte différents pour le même gestionnaire de files d'attente multi-instance, ce qui est une approche que vous pouvez choisir de combiner avec un module de remplacement de code.

Si vous choisissez une approche à plusieurs gestionnaires de files d'attente CCDT, vous devez choisir de définir les priorités des entrées ou de disposer d'une gestion de charge de travail (WLM) aléatoire:

Menaces

Utilisez plusieurs entrées classées par ordre alphabétique avec les attributs CLNTWGHT (1) et AFFINITY (PREFERRED) pour mémoriser la dernière connexion correcte.

Aléatoire
Utilisez les attributs CLNTWGHT (1) et AFFINITY (NONE). Vous pouvez ajuster la pondération WLM sur des serveurs IBM MQ mis à l'échelle différemment en ajustant CLNTWGHT

Remarque : Vous devez éviter les différences importantes dans CLNTWGHT entre les canaux.

Équilibreur de charge
Dispositif réseau avec une adresse IP virtuelle (VIP) configurée avec la surveillance de port des programmes d'écoute TCP/IP de plusieurs gestionnaires de files d'attente IBM MQ . La façon dont l'adresse IP virtuelle est configurée dans le dispositif réseau dépend du dispositif réseau que vous utilisez.

Les options suivantes concernent uniquement les applications qui envoient des messages ou qui initient des messages de demande et de réponse synchrones. Les considérations relatives aux applications gérant ces messages et ces demandes, par exemple les programmes d'écoute, sont complètement distinctes et sont abordées en détail dans la rubrique "Connexion d'un programme d'écoute de message à une file d'attente".

Echelle de changement de code requise pour les applications existantes qui se connectent à un seul gestionnaire de files d'attente

Liste CONNAME, CCDT multi-QMGR et équilibreur de charge
MQCONN ("NOMQM") à MQCONN ("*QMNAME")

Le nom du gestionnaire de files d'attente peut se trouver dans la configuration JNDI (Java Naming and Directory Interface) pour les applications Java Platform, Enterprise Edition (Java EE). Dans le cas contraire, un changement de code à un caractère est nécessaire.

Module de remplacement de code
Remplacez la logique de connexion JMS ou MQI existante par un module de remplacement de code.

Prise en charge de différentes stratégies WLM

Liste CONNAME
Priorisé uniquement.

Cela aura probablement un effet négatif sur le code.

CCDT à plusieurs gestionnaires de files d'attente
Priorisé ou aléatoire.

Il est peu probable que cela ait un effet sur le code.

Équilibreur de charge
Tout, y compris chaque connexion pour tous les messages.

Cela aura probablement un effet positif sur le code.

Module de remplacement de code
Tout, y compris chaque message pour tous les messages.

Cela aura probablement un effet positif sur le code.

Temps système de performances lors de l'indisponibilité du gestionnaire de files d'attente principal

Liste CONNAME
Toujours essayer en premier dans la liste.

Cela aura probablement un effet négatif sur le code.

CCDT à plusieurs gestionnaires de files d'attente
Se souvient de la dernière bonne connexion.

Cela aura probablement un effet positif sur le code.

Équilibreur de charge
La surveillance des ports évite les mauvais gestionnaires de files d'attente.

Cela aura probablement un effet positif sur le code.

Module de remplacement de code
Peut se souvenir de la dernière bonne connexion, et réessayer intelligemment.

Cela aura probablement un effet positif sur le code.

Support des transactions XA

Liste CONNAME, CCDT multi-QMGR et équilibreur de charge
Le gestionnaire de transactions doit stocker les informations de reprise qui se reconnectent à la même ressource de gestionnaire de files d'attente.

Un appel MQCONN qui se résout en différents gestionnaires de files d'attente invalide généralement cette situation. Par exemple, dans Java EE, une seule fabrique de connexions doit être résolue en un seul gestionnaire de files d'attente lors de l'utilisation de XA.

Cela aura probablement un effet négatif sur le code.

Module de remplacement de code
Le module de remplacement de code peut répondre aux exigences XA d'un gestionnaire de transactions, par exemple, plusieurs fabriques de connexions.

Cela aura probablement un effet positif sur le code.

Flexibilité de l'administrateur pour masquer les changements d'infrastructure dans les applications

Liste CONNAME
DNS uniquement.

Cela aura probablement un effet négatif sur le code.

CCDT à plusieurs gestionnaires de files d'attente
DNS et système de fichiers partagé, système de fichiers partagé ou commande push de fichier CCDT.

Il est peu probable que cela ait un effet sur le code.

Équilibreur de charge
Adresse IP virtuelle dynamique (VIP).

Cela aura probablement un effet positif sur le code.

Module de remplacement de code
Entrées CCDT DNS ou de gestionnaire de files d'attente unique.

Il est peu probable que cela ait un effet sur le code.

Eviter les interruptions dues à la maintenance planifiée

Il existe une autre situation que vous devez prendre en compte et planifier, qui consiste à éviter les interruptions des applications, par exemple les erreurs et les délais d'attente visibles par les utilisateurs finaux, lors de la maintenance planifiée d'un gestionnaire de files d'attente. La meilleure approche pour éviter les interruptions consiste à supprimer tout le travail d'un gestionnaire de files d'attente avant qu'il ne soit arrêté.

Envisagez un scénario de demande et de réponse. Vous voulez que toutes les demandes en cours soient terminées et que les réponses soient traitées par l'application, mais vous ne voulez pas qu'un travail supplémentaire soit soumis dans le système. La mise au repos du gestionnaire de files d'attente ne répond pas à ce besoin, car les applications codées reçoivent un code retour RC2161 MQRC_Q_MGR_QUIESCING, avant de recevoir leurs messages de réponse pour les demandes en cours.

Vous pouvez définir PUT (DISABLED) sur les files d'attente de demandes utilisées pour soumettre des travaux, tout en laissant les files d'attente de réponses à la fois PUT (ENABLED) et GET (ENABLED). Ainsi, vous pouvez surveiller la profondeur des files d'attente de demande, de transmission et de réponse. Une

fois qu'ils sont tous stabilisés, c'est-à-dire que les demandes en cours sont terminées ou qu'elles arrivent à terme, vous pouvez arrêter le gestionnaire de files d'attente.

Toutefois, un bon codage dans les applications demandeuses est nécessaire pour gérer une file d'attente de demandes PUT (DISABLED), ce qui génère le code retour RC2051 MQRC_PUT_INHIBÉ lors de la tentative d'envoi d'un message.

Notez que l'exception ne se produit pas lors de la création de la connexion à IBM MQou de l'ouverture de la file d'attente des demandes. L'exception se produit uniquement lorsqu'une tentative d'envoi effectif d'un message est effectuée à l'aide de l'appel MQPUT.

La génération d'un module de remplacement de code qui inclut cette logique de traitement des erreurs pour les scénarios de demande et de réponse, et la demande à vos équipes d'application d'utiliser un tel module de remplacement de code à l'avenir, peuvent vous aider à développer des applications avec un comportement cohérent.

Développement d'applications client flexibles et évolutives

Pour la tolérance aux pannes et l'évolutivité, le déploiement d'applications client prenant en charge les options de connexion dans des clusters uniformes permet de rééquilibrer les instances de l'application entre les gestionnaires de files d'attente.

Pour une présentation des clusters uniformes, voir A propos des clusters uniformes .

Dans l'idéal, ce rééquilibrage est invisible pour l'application, mais seuls certains types d'application sont adaptés à ce type de déploiement et il peut être nécessaire de prendre en compte la conception de l'application.

Ces considérations se répartissent en deux grandes catégories:

- Rares sont les *chemins d'erreur* qui peuvent déjà exister pour les applications reconnectables, mais qui deviennent plus probables lorsqu'ils sont déployés dans un cluster uniforme. Par exemple, après une reconnexion, toute unité d'oeuvre en cours est annulée et les curseurs sont réinitialisés. Il peut s'agir d'un événement rare pour votre application reconnectable dans son environnement en cours et, par conséquent, elle n'est pas traitée aussi proprement que possible par le code de l'application. La révision de la logique de l'application, afin de s'assurer que le traitement approprié est en place pour de telles situations, permet d'éviter des problèmes inattendus.
- *Affinités* avec un gestionnaire de files d'attente particulier. Si vous savez qu'une application doit toujours se reconnecter au même gestionnaire de files d'attente ou à un gestionnaire de files d'attente spécifique, l'application doit être configurée pour se reconnecter à ce gestionnaire de files d'attente ou sa connexion à ce gestionnaire de files d'attente ne doit pas être activée. Toutefois, ces affinités peuvent être temporaires, comme l'attente d'un message de réponse. L'influence de l'algorithme d'équilibrage sur la prise en compte de ces affinités à partir du code d'application est abordée dans la section suivante. Pour plus de détails sur ces options et pour savoir comment obtenir une approche similaire via la configuration plutôt que le code d'application, voir Influencer le rééquilibrage des applications dans des clusters uniformes.

Influence des options de reconnexion dans l'interface MQI

Pour plus d'informations sur MQCNO_RECONNECT, voir Options de reconnexion .

Si vous savez qu'une application doit toujours se reconnecter au même gestionnaire de files d'attente ou à un gestionnaire de files d'attente spécifique, elle doit être configurée en tant que MQCNO_RECONNECT_Q_MGR ou MQCNO_RECONNECT_DISABLED.

Influence de l'algorithme d'équilibrage dans l'interface MQI

Cependant, vous pouvez vouloir contrôler ou influencer ce comportement de rééquilibrage pour répondre aux besoins de types d'application spécifiques ; par exemple, en réduisant les interruptions dans les transactions en cours ou en vous assurant que les applications du demandeur reçoivent leurs réponses avant d'être déplacées.

Certains comportements souhaitables par défaut sont pris en compte et traités dans la rubrique Influencer le rééquilibrage des applications dans des clusters uniformes. Vous pouvez également influencer le comportement d'applications spécifiques lors de la configuration ou du déploiement via le fichier client.ini , comme indiqué dans cette rubrique.

Dans d'autres cas, il peut être plus logique de faire en sorte que le comportement d'équilibrage et les exigences fassent partie de la logique d'application. Dans ces cas, les mêmes caractéristiques pertinentes de l'application peuvent être fournies à IBM MQ lors de la connexion au gestionnaire de files d'attente sur l'appel MQCONNX, dans une structure appelée MQBNO (options d'équilibrage).

Si vous fournissez une structure MQBNO, elle doit fournir toutes les informations requises par IBM MQ pour décider quand et comment l'application doit être invitée à se reconnecter à un autre gestionnaire de files d'attente.

Vous devez fournir:

- Le **Type** de l'application
- **Timeout** au niveau duquel l'instance est rééquilibrée quel que soit l'état
- Tout **BalanceOptions** spécial

La seule exception à cette règle est que vous pouvez conserver le délai d'attente MQBNO_TIMEOUT_DEFAULT si nécessaire. Dans ce cas, le délai d'attente est résolu en n'importe quelle valeur du fichier client.ini , de l'application ou des sections globales, si elles sont fournies et, à défaut, en indiquant la valeur par défaut de base de 10 secondes.

Voir MQBNO pour plus de détails sur le format de cette structure.

Pour les applications .NET, voir Influencer le rééquilibrage des applications dans .NET pour plus d'informations.

Applications orientées objet

IBM MQ prend en charge JMS, Java, C + + et .NET. Ces langages et infrastructures utilisent le modèle d'objet IBM MQ , qui fournit des classes fournissant les mêmes fonctionnalités que les appels et les structures IBM MQ .

Certains des langages et des infrastructures qui utilisent le modèle d'objet IBM MQ fournissent des fonctions supplémentaires qui ne sont pas disponibles lorsque vous utilisez des langages de procédure avec l'interface MQI (Message Queue Interface).

Pour plus de détails sur les classes, les méthodes et les propriétés fournies par ce modèle, voir «Modèle d'objet IBM MQ», à la page 17.

JMS

IBM MQ fournit des classes qui implémentent les spécifications Jakarta Messaging 3.0 et Java Message Service 2.0 . Pour plus de détails sur IBM MQ classes for JMS, voir Utilisation de IBM MQ classes for JMS. Pour plus d'informations sur les différences entre IBM MQ classes for Java et IBM MQ classes for JMS, voir «Développement d'applications JMS/Jakarta Messaging et Java», à la page 84.

IBM MQ Message Service Client (XMS) for C/C++ et IBM MQ Message Service Client (XMS) for .NET fournissent une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que Java Message Service (JMS) API. Pour plus d'informations, voir «Développement d'applications XMS .NET», à la page 629.

Java

Voir Utilisation de IBM MQ classes for Java pour plus d'informations sur le codage des programmes à l'aide du modèle d'objet IBM MQ dans Java.

Stabilized IBM n'apportera pas d'autres améliorations à IBM MQ classes for Java ; cette fonctionnalité est stabilisée au niveau livré dans IBM MQ 8.0. Pour plus d'informations sur les différences entre le IBM MQ classes for Java et le IBM MQ classes for JMS afin de vous aider à décider lequel utiliser, voir «Développement d'applications JMS/Jakarta Messaging et Java», à la page 84.

C++
IBM MQ fournit des classes C++ équivalentes à des objets IBM MQ et des classes supplémentaires équivalentes aux types de données de tableau. Il fournit un certain nombre de fonctions qui ne sont pas disponibles via l'interface MQI. Voir Utilisation de C++ pour plus d'informations sur le codage des programmes à l'aide du modèle d'objet IBM MQ dans C + +. Les clients de service de message pour C/C++ et .NET fournissent une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que le Java Message Service (JMS) API.

.NET
Pour plus d'informations sur le codage des programmes .NET à l'aide des classes IBM MQ .NET , voir Développement d'applications .NET . Les clients de service de messagerie pour C/C++ et .NET fournissent une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que le Java Message Service (JMS) API.

Concepts associés

«Développement d'applications MQI avec IBM MQ», à la page 734
IBM MQ prend en charge C, Visual Basic, COBOL, Assembler, RPG, pTALet PL/I. Ces langages procéduraux utilisent l'interface de file d'attente de messages (MQI) pour accéder aux services de mise en file d'attente de messages.

Présentation technique

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

Référence associée

Références relatives au développement d'applications

Modèle d'objet IBM MQ

Le modèle d'objet IBM MQ se compose de classes, de méthodes et de propriétés.

Le modèle d'objet IBM MQ se compose des éléments suivants:

- *Classes* représentant des concepts IBM MQ familiers tels que les gestionnaires de files d'attente, les files d'attente et les messages.
- *Méthodes* sur chaque classe correspondant aux appels MQI.
- *Propriétés* sur chaque classe correspondant aux attributs des objets IBM MQ .

Lorsque vous créez une application IBM MQ à l'aide du modèle d'objet IBM MQ , vous créez des instances de ces classes dans l'application. Une instance d'une classe en programmation orientée objet est appelée *objet*. Lorsqu'un objet a été créé, vous interagissez avec l'objet en examinant ou en définissant les valeurs des propriétés de l'objet (l'équivalent de l'émission d'un appel MQINQ ou MQSET) et en effectuant des appels de méthode sur l'objet (l'équivalent de l'émission des autres appels MQI).

Classes

Le modèle d'objet IBM MQ fournit l'ensemble de classes de base suivant.

L'implémentation réelle du modèle varie légèrement entre les différents environnements orientés objet pris en charge.

MQQueueManager
Un objet de la classe MQQueueManager représente une connexion à un gestionnaire de files d'attente. Il comporte des méthodes Connect (), Disconnect (), Commit () et Backout () (l'équivalent de MQCONN ou MQCONNX, MQDISC, MQCMIT et MQBACK). Il possède des propriétés correspondant aux attributs d'un gestionnaire de files d'attente. L'accès à une propriété d'attribut de gestionnaire de files d'attente permet de se connecter implicitement au gestionnaire de files d'attente s'il n'est pas déjà connecté. La destruction d'un objet MQQueueManager se déconnecte implicitement du gestionnaire de files d'attente.

MQQUEUE
Un objet de la classe MQQueue représente une file d'attente. Il comporte des méthodes pour placer () et extraire () des messages vers et depuis la file d'attente (l'équivalent de MQPUT et MQGET). Il possède des propriétés correspondant aux attributs d'une file d'attente. L'accès à une propriété d'attribut de file d'attente ou l'émission d'un appel de méthode Put () ou Get () ouvre implicitement la file d'attente (l'équivalent de MQOPEN). La destruction d'un objet MQQueue ferme implicitement la file d'attente (l'équivalent de MQCLOSE).

Sujet MQ
Un objet de la classe MQTopic représente une rubrique. Il comporte des méthodes pour placer () (publier) et extraire () (recevoir ou s'abonner) des messages vers et depuis la rubrique (l'équivalent de MQPUT et de MQGET). Il possède des propriétés correspondant aux attributs d'une rubrique. Un objet MQTopic n'est accessible que pour la publication ou l'abonnement, pas simultanément. Lorsqu'il est utilisé pour la réception de messages, l'objet MQTopic peut être créé avec un abonnement non géré ou géré et en tant qu'abonné durable ou non durable-plusieurs constructeurs surchargés sont fournis pour ces différents scénarios.

Message MQ
Un objet de la classe MQMessage représente un message à insérer dans une file d'attente ou à obtenir d'une file d'attente. Il contient une mémoire tampon et encapsule à la fois les données d'application et MQMD. Il possède des propriétés correspondant à des zones MQMD et des méthodes qui vous permettent d'écrire et de lire des données utilisateur de différents types (par exemple, des chaînes, des entiers longs, des entiers courts, des octets simples) vers et depuis la mémoire tampon.

MQPutMessage-Options
Un objet de la classe MQPutMessageOptions représente la structure MQPMO. Il possède des propriétés correspondant aux zones MQPMO.

Options de MQGetMessage
Un objet de la classe MQGetMessageOptions représente la structure MQGMO. Il possède des propriétés correspondant à des zones MQGMO.

Processus MQ
Un objet de la classe MQProcess représente une définition de processus (utilisée avec déclenchement). Il possède des propriétés qui représentent les attributs d'une définition de processus.

MQDistributionList
Un objet de la classe MQDistributionList représente une liste de distribution (utilisée pour envoyer plusieurs messages avec un seul MQPUT). Il contient une liste d'objets d'élément MQDistributionList.

Elément MQDistributionList
Un objet de la classe d'élément MQDistributionListreprésente une destination de liste de distribution unique. Il encapsule les structures MQOR, MQRR et MQPMR et possède des propriétés correspondant aux zones de ces structures.

Références d'objet

Dans un programme IBM MQ qui utilise l'interface MQI, IBM MQ renvoie des descripteurs de connexion et des descripteurs d'objet au programme.

Ces descripteurs doivent être transmis en tant que paramètres lors des appels IBM MQ ultérieurs. Avec le modèle d'objet IBM MQ , ces descripteurs sont masqués dans le programme d'application. A la place, la création d'un objet à partir d'une classe entraîne le renvoi d'une référence d'objet au programme d'application. C'est cette référence d'objet qui est utilisée lors des appels de méthode et des accès de propriété à l'objet.

Codes retour

L'émission d'un appel de méthode ou la définition d'une valeur de propriété entraîne la définition de codes retour.

Ces codes retour sont un code achèvement et un code raison, et sont eux-mêmes des propriétés de l'objet. Les valeurs du code achèvement et du code anomalie sont les mêmes que celles définies pour l'interface MQI, avec des valeurs supplémentaires spécifiques à l'environnement orienté objet.

Messages IBM MQ

Un message IBM MQ se compose de propriétés de message et de données d'application. Le descripteur de message de mise en file d'attente de messages (MQMD) contient les informations de contrôle qui accompagnent les données d'application lorsqu'un message circule entre les applications d'envoi et de réception.

Parties d'un message

Les messages IBM MQ se composent de deux parties:

- Propriétés des messages
- Données d'application

Figure 2, à la page 19 représente un message et montre comment il est logiquement divisé en propriétés de message et en données d'application.

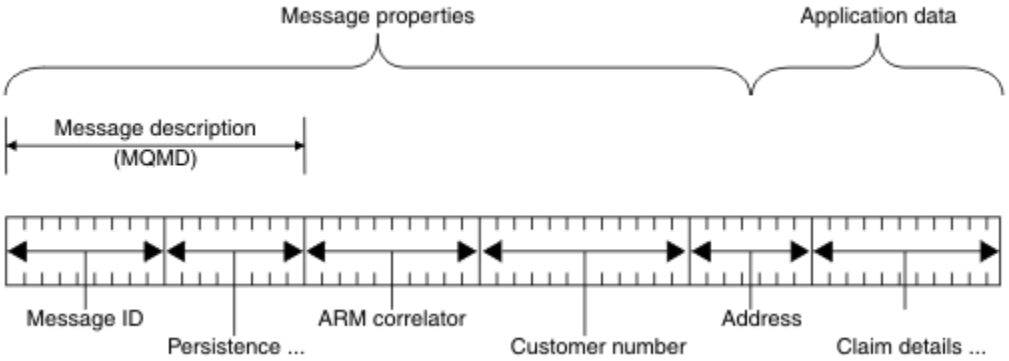


Figure 2. Représentation d'un message

Les données d'application contenues dans un message IBM MQ ne sont pas modifiées par un gestionnaire de files d'attente sauf si une conversion de données est effectuée sur ce dernier. De plus, IBM MQ n'impose aucune restriction sur le contenu de ces données. La longueur des données de chaque message ne peut pas dépasser la valeur de l'attribut **MaxMsgLength** de la file d'attente et du gestionnaire de files d'attente.

ALW Sous AIX, Linux, and Windows, l'attribut *MaxMsgLength* du gestionnaire de files d'attente et de la file d'attente prend par défaut la valeur 4 Mo (4 194 304 octets), que vous pouvez modifier jusqu'à un maximum de 100 Mo (104 857 600 octets) si nécessaire.

IBM i Sous IBM i, l'attribut *MaxMsgLength* du gestionnaire de files d'attente et de la file d'attente prend par défaut la valeur 4 Mo (4 194 304 octets), que vous pouvez modifier jusqu'à un maximum de 100 Mo (104 857 600 octets) si nécessaire. Si vous avez l'intention d'utiliser des messages IBM MQ supérieurs à 15 Mo sous IBM i, voir «Génération de votre application procédurale sur IBM i», à la page 1034.

z/OS Sous z/OS, l'attribut **MaxMsgLength** du gestionnaire de files d'attente est fixé à 100 Mo et l'attribut **MaxMsgLength** de la file d'attente est défini par défaut sur 4 Mo (4 194 304 octets), que vous pouvez modifier jusqu'à un maximum de 100 Mo si nécessaire.

Faites en sorte que vos messages soient légèrement plus courts que la valeur de l'attribut **MaxMsgLength** dans certains cas. Pour plus d'informations, voir «Les données de votre message», à la page 777.

Vous créez un message lorsque vous utilisez les appels MQI MQPUT ou MQPUT1 . En entrée de ces appels, vous fournissez les informations de contrôle (telles que la priorité du message et le nom d'une file d'attente de réponses) et vos données, puis l'appel place le message dans une file d'attente. Pour plus d'informations sur ces appels, voir MQPUT et MQPUT1 .

Descripteur de message

Vous pouvez accéder aux informations de contrôle des messages à l'aide de la structure MQMD, qui définit le *descripteur de message*.

Pour une description complète de la structure MQMD, voir MQMD-Descripteur de message.

Pour savoir comment utiliser les zones du MQMD qui contiennent des informations sur l'origine du message, voir «Contexte de message», à la page 49.

Il existe différentes versions du descripteur de message. Des informations supplémentaires sur le regroupement et la segmentation des messages (voir «Groupes de messages», à la page 46) sont fournies dans la version 2 du descripteur de message (ou MQMDE). Il s'agit du même descripteur de message que celui de la version 1, mais il comporte des zones supplémentaires. Ces zones sont décrites dans MQMDE-Extension de descripteur de message.

Types de message

Il existe quatre types de message définis par IBM MQ.

Ces quatre messages sont les suivants:

- Datagramme
- Messages de demande
- Messages de réponse
- Messages de rapport
  - Types de message de rapport
  - Options de message de rapport

Les applications peuvent utiliser les trois premiers types de message pour transmettre des informations entre elles. Le quatrième type, le rapport, est destiné aux applications et aux gestionnaires de files d'attente à utiliser pour signaler des informations sur des événements tels que l'occurrence d'une erreur.

Chaque type de message est identifié par une valeur MQMT_ *. Vous pouvez également définir vos propres types de message. Pour la plage de valeurs que vous pouvez utiliser, voir MsgType.

Datagrammes

Utilisez un *datagramme* lorsque vous n'avez pas besoin d'une réponse de l'application qui reçoit le message (c'est-à-dire que vous obtenez le message de la file d'attente).

Un exemple d'application qui peut utiliser des datagrammes est celui qui affiche les informations de vol dans un salon d'aéroport. Un message peut contenir les données d'un écran complet d'informations de vol. Il est peu probable qu'une telle application demande un accusé de réception pour un message car cela n'a probablement pas d'importance si un message n'est pas distribué. L'application envoie un message de mise à jour après un court laps de temps.

Messages de demande

Utilisez un *message de demande* lorsque vous souhaitez une réponse de l'application qui reçoit le message.

Un exemple d'application qui peut utiliser des messages de demande est celui qui affiche le solde d'un compte courant. Le message de demande peut contenir le numéro du compte et le message de réponse peut contenir le solde du compte.

Si vous souhaitez lier votre message de réponse à votre message de demande, vous disposez de deux options:

- Faites en sorte que l'application qui gère le message de demande soit chargée de s'assurer qu'elle insère des informations dans le message de réponse associé au message de demande.
- Utilisez la zone de rapport dans le descripteur de message de votre message de demande pour spécifier le contenu des zones *MsgId* et *CorrelId* du message de réponse:
  - Vous pouvez demander que le *MsgId* ou le *CorrelId* du message d'origine soit copié dans la zone *CorrelId* du message de réponse (l'action par défaut consiste à copier *MsgId*).
  - Vous pouvez demander qu'un nouveau *MsgId* soit généré pour le message de réponse ou que le *MsgId* du message d'origine soit copié dans la zone *MsgId* du message de réponse (l'action par défaut consiste à générer un nouvel identificateur de message).

Messages de réponse

Utilisez un *message de réponse* lorsque vous répondez à un autre message.

Lorsque vous créez un message de réponse, respectez les options qui ont été définies dans le descripteur de message du message auquel vous répondez. Les options de rapport spécifient le contenu des zones d'identificateur de message (*MsgId*) et d'identificateur de corrélation (*CorrelId*). Ces zones permettent à l'application qui reçoit la réponse de corréler la réponse avec sa demande d'origine.

Messages de rapport

Les *messages de rapport* informent les applications des événements tels que l'occurrence d'une erreur lors du traitement d'un message.

Ils peuvent être générés par:

- un gestionnaire de files d'attente,
- Un agent MCA (par exemple, s'ils ne peuvent pas distribuer le message), ou
- Une application (par exemple, si elle ne peut pas utiliser les données du message).

Les messages de rapport peuvent être générés à tout moment et peuvent arriver dans une file d'attente lorsque votre application ne les attend pas.

Types de message de rapport

Lorsque vous placez un message dans une file d'attente, vous pouvez choisir de recevoir:

- Un *message de rapport d'exception*. Ce message est envoyé en réponse à un message avec l'indicateur d'exceptions défini. Il est généré par l'agent MCA ou l'application.
- Un *message de rapport d'expiration*. Indique qu'une application a tenté d'extraire un message qui avait atteint son seuil d'expiration ; le message est marqué comme devant être supprimé. Ce type de rapport est généré par le gestionnaire de files d'attente.
- Un *message de rapport de confirmation d'arrivée (COA)*. Indique que le message a atteint sa file d'attente cible. Il est généré par le gestionnaire de files d'attente.
- Un *message de rapport de confirmation de distribution (COD)*. Indique que le message a été extrait par une application de réception. Il est généré par le gestionnaire de files d'attente.
- Un *message de rapport de notification d'action positive (PAN)*. Indique qu'une demande a été traitée avec succès (c'est-à-dire que l'action demandée dans le message a été effectuée avec succès). Ce type de rapport est généré par l'application.
- Un *message de rapport de notification d'action négative (NAN)*. Cela indique qu'une demande n'a pas été traitée correctement (c'est-à-dire que l'action demandée dans le message n'a pas été effectuée correctement). Ce type de rapport est généré par l'application.

Remarque : Chaque type de message de rapport contient l'un des éléments suivants:

- L'intégralité du message d'origine

- Les 100 premiers octets de données dans le message d'origine
- Aucune donnée du message d'origine

Vous pouvez demander plusieurs types de message de rapport lorsque vous placez un message dans une file d'attente. Si vous sélectionnez le message de confirmation de distribution et les options de message de rapport d'exception, si le message ne parvient pas à être distribué, vous recevez un message de rapport d'exception. Toutefois, si vous sélectionnez uniquement l'option de message de confirmation de distribution et que le message ne parvient pas à être distribué, vous n'obtenez pas de message de rapport d'exception.

Les messages de rapport que vous demandez, lorsque les critères de génération d'un message particulier sont remplis, sont les seuls que vous recevez.

Options de message de rapport

Vous pouvez *supprimer* un message après l'apparition d'une exception. Si vous sélectionnez l'option de suppression et que vous avez demandé un message de rapport d'exception, le message de rapport est envoyé à *ReplyToQ* et *ReplyToQMgr*et le message d'origine est supprimé.

Remarque : L'avantage est que vous pouvez réduire le nombre de messages placés dans la file d'attente des messages non livrés. Cependant, cela signifie que votre application, à moins qu'elle n'envoie que des messages de datagramme, doit traiter les messages renvoyés. Lorsqu'un message de rapport d'exception est généré, il hérite de la persistance du message d'origine.

Si un message de rapport ne peut pas être distribué (si la file d'attente est saturée, par exemple), le message de rapport est placé dans la file d'attente de rebut.

Si vous souhaitez recevoir un message de rapport, indiquez le nom de votre file d'attente de réponse dans la zone *ReplyToQ* ; sinon, la commande MQPUT ou MQPUT1 de votre message d'origine échoue avec MQRC_MISSING_REPLY_TO_Q.

Vous pouvez utiliser d'autres options de rapport dans le descripteur de message (MQMD) d'un message pour spécifier le contenu des zones *MsgId* et *CorrelId* des messages de rapport créés pour le message:

- Vous pouvez demander que le *MsgId* ou le *CorrelId* du message d'origine soit copié dans la zone *CorrelId* du message de rapport. L'action par défaut consiste à copier l'identificateur de message. Utilisez MQRO_COPY_MSG_ID_TO_CORRELID car il permet à l'expéditeur d'un message de corréler le message de réponse ou de rapport avec le message d'origine. L'identificateur de corrélation du message de réponse ou de rapport est identique à l'identificateur du message d'origine.
- Vous pouvez demander qu'un nouveau *MsgId* soit généré pour le message de rapport ou que le *MsgId* du message d'origine soit copié dans la zone *MsgId* du message de rapport. L'action par défaut consiste à générer un nouvel identificateur de message. Utilisez MQRO_NEW_MSG_ID car il garantit que chaque message du système possède un identificateur de message différent et qu'il peut être distingué sans ambiguïté de tous les autres messages du système.
- Les applications spécialisées peuvent avoir besoin d'utiliser MQRO_PASS_MSG_ID ou MQRO_PASS_CORREL_ID. Toutefois, vous devez concevoir l'application qui lit les messages à partir de la file d'attente pour vous assurer qu'elle fonctionne correctement lorsque, par exemple, la file d'attente contient plusieurs messages avec le même identificateur de message.

  Les applications serveur doivent vérifier les paramètres de ces indicateurs dans le message de demande et définir les zones *MsgId* et *CorrelId* dans le message de réponse ou de rapport de manière appropriée.

  Les applications qui agissent en tant qu'intermédiaires entre une application de demandeur et une application de serveur n'ont pas besoin de vérifier les paramètres de ces indicateurs. En effet, ces applications doivent généralement transmettre le message à l'application serveur avec les zones *MsgId*, *CorrelId*et *Report* inchangées. Cela permet à l'application serveur de copier le *MsgId* à partir du message d'origine dans la zone *CorrelId* du message de réponse.

Lors de la génération d'un rapport sur un message, les applications serveur doivent tester si l'une de ces options a été définie.

Pour plus d'informations sur l'utilisation des messages de rapport, voir Rapport.

Pour indiquer la nature du rapport, les gestionnaires de files d'attente utilisent une série de codes retour. Ils placent ces codes dans la zone *Feedback* du descripteur de message d'un message de rapport. Les gestionnaires de files d'attente peuvent également renvoyer des codes anomalie MQI dans la zone *Feedback* . IBM MQ définit une plage de codes retour à utiliser par les applications.

Pour plus d'informations sur les commentaires en retour et les codes raison, voir Commentaires en retour.

Un exemple de programme qui peut utiliser un code retour est celui qui surveille les charges de travail d'autres programmes servant une file d'attente. S'il existe plusieurs instances d'un programme servant une file d'attente et que le nombre de messages arrivant dans la file d'attente ne le justifie plus, ce programme peut envoyer un message de rapport (avec le code retour MQFB_QUIT) à l'un des programmes servant pour indiquer que le programme doit mettre fin à son activité. (Un programme de surveillance peut utiliser l'appel MQINQ pour déterminer combien de programmes servent une file d'attente.)

Multi *Rapports et messages segmentés*

Si un message est segmenté et que vous demandez la génération de rapports, vous risquez de recevoir plus de rapports que si le message n'avait pas été segmenté. Les rapports sur les messages segmentés ne sont disponibles que sur Multiplatforms.

Pour obtenir une description des messages segmentés, voir «Segmentation des messages», à la page 812.

Pour les rapports générés par IBM MQ

Si vous segmentez vos messages ou autorisez le gestionnaire de files d'attente à le faire, il n'y a qu'un seul cas dans lequel vous pouvez vous attendre à recevoir un seul rapport pour l'ensemble du message. C'est lorsque vous avez demandé uniquement des rapports COD et que vous avez spécifié MQGMO_COMPLETE_MSG dans l'application d'obtention.

Dans d'autres cas, votre demande doit être préparée pour traiter plusieurs rapports, généralement un pour chaque segment.

Remarque : Si vous segmentez vos messages et que vous n'avez besoin que des 100 premiers octets des données de message d'origine à renvoyer, modifiez le paramètre des options de rapport pour demander des rapports sans données pour les segments dont le décalage est égal ou supérieur à 100. Si vous n'effectuez pas cette opération et que vous laissez le paramètre de sorte que chaque segment demande 100 octets de données et que vous extrayez les messages de rapport avec une instruction MQGET unique spécifiant MQGMO_COMPLETE_MSG, les rapports s'assemblent en un message volumineux contenant 100 octets de données lues à chaque décalage approprié. Si cela se produit, vous avez besoin d'une mémoire tampon de grande taille ou vous devez spécifier MQGMO_ACCEPT_TRUNCATED_MSG.

Pour les rapports générés par les applications

Si votre application génère des rapports, copiez toujours les en-têtes IBM MQ présents au début des données de message d'origine dans les données de message de rapport.

Ajoutez ensuite aucune, 100 octets ou toutes les données de message d'origine (ou toute autre quantité que vous incluez généralement) aux données de message de rapport.

Vous pouvez reconnaître les en-têtes IBM MQ qui doivent être copiés en examinant les noms de format successifs, en commençant par le MQMD et en passant par les en-têtes présents. Les noms `Format` suivants indiquent ces en-têtes IBM MQ :

- MQMDE
- MQDLH
- MQXQH
- MQIIH

- MQH *

MQH* signifie tout nom commençant par les caractères MQH.

Le nom `Format` apparaît à des positions spécifiques pour MQDLH et MQXQH, mais pour les autres en-têtes IBM MQ , il apparaît à la même position. La longueur de l'en-tête est contenue dans une zone qui se trouve également à la même position pour MQMDE, MQIMSet tous les en-têtes MQH*.

Si vous utilisez un MQMD version 1 et que vous signalez un segment, un message dans un groupe ou un message pour lequel la segmentation est autorisée, les données de rapport doivent commencer par un MQMDE. Définissez la zone *`OriginalLength`* sur la longueur des données du message d'origine, à l'exclusion des longueurs des en-têtes IBM MQ trouvés.

Extraction de rapports

Si vous demandez des rapports COA ou COD, vous pouvez demander qu'ils soient réassemblés pour vous avec MQGMO_COMPLETE_MSG.

Une requête MQGET avec MQGMO_COMPLETE_MSG est satisfaite lorsqu'un nombre suffisant de messages de rapport (d'un type unique, par exemple COA, et avec le même *`GroupId`*) sont présents dans la file d'attente pour représenter un message d'origine complet. Cela est vrai même si les messages de rapport eux-mêmes ne contiennent pas les données d'origine complètes ; la zone *`OriginalLength`* de chaque message de rapport indique la longueur des données d'origine représentées par ce message de rapport, même si les données elles-mêmes ne sont pas présentes.

Vous pouvez utiliser cette technique même si plusieurs types de rapport différents sont présents dans la file d'attente (par exemple, COA et COD), car MQGET avec MQGMO_COMPLETE_MSG réassemble les messages de rapport uniquement s'ils ont le même code *`Feedback`* . Toutefois, vous ne pouvez généralement pas utiliser cette technique pour les rapports d'exception car, en général, ils ont des codes *`Feedback`* différents.

Vous pouvez utiliser cette technique pour obtenir une indication positive que le message entier est arrivé. Toutefois, dans la plupart des cas, vous devez tenir compte de la possibilité que certains segments arrivent alors que d'autres peuvent générer une exception (ou expiration, si vous l'avez autorisé). Vous ne pouvez pas utiliser MQGMO_COMPLETE_MSG dans ce cas, car, en général, vous pouvez obtenir des codes *`Feedback`* différents pour différents segments et vous pouvez obtenir plusieurs rapports pour un segment. Vous pouvez toutefois utiliser MQGMO_ALL_SEGMENTS_AVAILABLE.

Pour ce faire, vous devrez peut-être extraire des rapports à mesure qu'ils arrivent et créer une image dans votre application de ce qui est arrivé au message d'origine. Vous pouvez utiliser la zone *`GroupId`* dans le message de rapport pour corréler les rapports avec le *`GroupId`* du message d'origine et la zone *`Feedback`* pour identifier le type de chaque message de rapport. La manière dont vous effectuez cette opération dépend des exigences de votre application.

Une approche est la suivante:

- Demandez des rapports COD et des rapports d'exception.
- Après une heure spécifique, vérifiez si un ensemble complet de rapports COD a été reçu à l'aide de MQGMO_COMPLETE_MSG. Si tel est le cas, votre application sait que la totalité du message a été traitée.
- Si ce n'est pas le cas, et si des rapports d'exception relatifs à ce message sont présents, gérez le problème comme pour les messages non segmentés, mais veillez à nettoyer les segments orphelins à un moment donné.
- S'il existe des segments pour lesquels il n'existe aucun rapport d'aucune sorte, les segments d'origine (ou les rapports) peuvent être en attente de reconnexion d'un canal ou le réseau peut être surchargé à un moment donné. Si aucun rapport d'exception n'a été reçu (ou si vous pensez que ceux que vous avez ne sont que temporaires), vous pouvez décider de laisser votre application attendre un peu plus longtemps.

  Comme précédemment, cela est similaire aux remarques que vous avez sur le traitement des messages non segmentés, sauf que vous devez également prendre en compte la possibilité de nettoyer les segments orphelins.

Si le message d'origine n'est pas critique (par exemple, s'il s'agit d'une requête ou d'un message qui peut être répété ultérieurement), définissez une heure d'expiration pour vous assurer que les segments orphelins sont supprimés.

Gestionnaires de files d'attente de niveau antérieur

Lorsqu'un rapport est généré par un gestionnaire de files d'attente qui prend en charge la segmentation, mais qu'il est reçu sur un gestionnaire de files d'attente qui ne prend pas en charge la segmentation, la structure MQMDE (qui identifie les *Offset* et *OriginalLength* représentés par le rapport) est toujours incluse dans les données du rapport, en plus de zéro, 100 octets ou toutes les données d'origine du message.

Toutefois, si un segment d'un message passe par un gestionnaire de files d'attente qui ne prend pas en charge la segmentation, si un rapport y est généré, la structure MQMDE du message d'origine est uniquement traitée comme des données. Il n'est donc pas inclus dans les données du rapport si aucun octet des données d'origine n'a été demandé. Sans MQMDE, le message de rapport peut ne pas être utile.

Demandez au moins 100 octets de données dans les rapports s'il est possible que le message transite par un gestionnaire de files d'attente de niveau antérieur.

Format des informations de contrôle de message et des données de message

Le gestionnaire de files d'attente ne s'intéresse qu'au format des informations de contrôle dans un message, tandis que les applications qui traitent le message s'intéressent au format des informations de contrôle et des données.

Format des informations de contrôle de message

Les informations de contrôle des zones de chaîne de caractères du descripteur de message doivent figurer dans le jeu de caractères utilisé par le gestionnaire de files d'attente.

L'attribut **CodedCharSetId** de l'objet gestionnaire de files d'attente définit ce jeu de caractères. Les informations de contrôle doivent figurer dans ce jeu de caractères car, lorsque des applications transmettent des messages d'un gestionnaire de files d'attente à un autre, les agents MCA qui transmettent les messages utilisent la valeur de cet attribut pour déterminer la conversion de données à effectuer.

Format des données de message

Vous pouvez spécifier l'un des éléments suivants:

- Format des données d'application
- Jeu de caractères des données de type caractères
- Format des données numériques

Pour ce faire, utilisez les zones suivantes:

Format
Indique au destinataire d'un message le format des données d'application du message.

Lorsque le gestionnaire de files d'attente crée un message, il utilise dans certains cas la zone *Format* pour identifier le format de ce message. Par exemple, lorsqu'un gestionnaire de files d'attente ne parvient pas à distribuer un message, il le place dans une file d'attente de rebut (message non distribué). Il ajoute un en-tête (contenant plus d'informations de contrôle) au message et modifie la zone *Format* pour l'afficher.

Le gestionnaire de files d'attente possède un certain nombre de *formats intégrés* dont les noms commencent par MQ, par exemple MQFMT_STRING. Si elles ne répondent pas à vos besoins, vous pouvez définir vos propres formats (*formats définis par l'utilisateur*), mais vous ne devez pas utiliser de noms commençant par MQ pour ces formats.

Lorsque vous créez et utilisez vos propres formats, vous devez écrire un exit de conversion de données pour prendre en charge un programme qui obtient le message à l'aide de MQGMO_CONVERT.

CodedCharSetId
Définit le jeu de caractères des données de type caractère dans le message. Si vous souhaitez définir ce jeu de caractères sur celui du gestionnaire de files d'attente, vous pouvez définir cette zone sur la constante MQCCSI_Q_MGR ou MQCCSI_INHERIT.

Lorsque vous obtenez un message d'une file d'attente, comparez la valeur de la zone *CodedCharSetId* avec la valeur attendue par votre application. Si les deux valeurs diffèrent, vous devrez peut-être convertir des données de type caractères dans le message ou utiliser un exit de message de conversion de données s'il en existe un.

Encoding
Cette section décrit le format des données de message numérique qui contiennent des entiers binaires, des entiers décimaux condensés et des nombres à virgule flottante. Il est généralement codé en fonction de la machine particulière sur laquelle le gestionnaire de files d'attente s'exécute.

Lorsque vous placez un message dans une file d'attente, vous spécifiez généralement la constante MQENC_NATIVE dans la zone *Encoding* . Cela signifie que le codage de vos données de message est identique à celui de la machine sur laquelle votre application s'exécute.

Lorsque vous obtenez un message d'une file d'attente, comparez la valeur de la zone *Encoding* dans le descripteur de message avec la valeur de la constante MQENC_NATIVE sur votre machine. Si les deux valeurs sont différentes, vous devrez peut-être convertir des données numériques dans le message ou utiliser un exit de message de conversion de données s'il en existe un.

Conversion des données d'application
Il se peut que les données d'application doivent être converties en jeu de caractères et en codage requis par une autre application lorsque des plateformes différentes sont concernées.

Il peut être converti au niveau du gestionnaire de files d'attente émetteur ou du gestionnaire de files d'attente récepteur. Si la bibliothèque de formats intégrés ne répond pas à vos besoins, vous pouvez définir la vôtre. Le type de conversion dépend du format de message spécifié dans la zone de format du descripteur de message, MQMD.

Remarque : Les messages avec MQFMT_NONE spécifié ne sont pas convertis.

Conversion au niveau du gestionnaire de files d'attente émetteur

Définissez l'attribut de canal CONVERT sur YES si vous avez besoin de l'agent MCA (Message Channel Agent) pour convertir les données d'application.

La conversion est effectuée au niveau du gestionnaire de files d'attente d'envoi pour certains formats intégrés et pour les formats définis par l'utilisateur si un exit utilisateur approprié est fourni.

Formats intégrés

Ces gestionnaires sont les suivants :

- Messages contenant tous les caractères (à l'aide du nom de format MQFMT_STRING)
- Messages définis par IBM MQ , par exemple des formats de commande programmables

  IBM MQ utilise des messages au format de commande programmable pour les messages et les événements d'administration (le nom de format utilisé est MQFMT_ADMIN dans ce cas). Vous pouvez utiliser le même format (à l'aide du nom de format MQFMT_PCF) pour vos propres messages et tirer parti de la conversion de données intégrée.

Les formats intégrés du gestionnaire de files d'attente ont tous des noms commençant par MQFMT. Ils sont répertoriés et décrits dans Format.

Formats définis par l'application

Pour les formats définis par l'utilisateur, la conversion des données d'application doit être effectuée par un programme d'exit de conversion de données (pour plus d'informations, voir «Ecriture des exits

de conversion de données», à la page 1010). Dans un environnement client-serveur, l'exit est chargé sur le serveur et la conversion y est effectuée.

Conversion au niveau du gestionnaire de files d'attente de réception

Les données de message d'application peuvent être converties par le gestionnaire de files d'attente de réception pour les formats intégrés et définis par l'utilisateur.

La conversion est effectuée lors du traitement d'un appel MQGET si vous spécifiez l'option MQGMO_CONVERT. Pour plus de détails, voir Options

Jeux de caractères codés

Les produits IBM MQ prennent en charge les jeux de caractères codés fournis par le système d'exploitation sous-jacent.

Lorsque vous créez un gestionnaire de files d'attente, l'ID de jeu de caractères codés (CCSID) du gestionnaire de files d'attente utilisé est basé sur celui de l'environnement sous-jacent. S'il s'agit d'une page de codes mixte, IBM MQ utilise la partie SBCS de la page de codes mixte comme CCSID du gestionnaire de files d'attente.

Pour la conversion générale des données, si le système d'exploitation sous-jacent prend en charge les pages de codes DBCS, IBM MQ peut l'utiliser.

Consultez la documentation de votre système d'exploitation pour plus de détails sur les jeux de caractères codés qu'il prend en charge.

Vous devez tenir compte de la conversion des données d'application, des noms de format et des exits utilisateur lorsque vous écrivez des applications qui s'étendent sur plusieurs plateformes. Pour plus d'informations sur l'appel et l'écriture des exits de conversion de données, voir «Ecriture des exits de conversion de données», à la page 1010 .

Priorités des messages

Vous pouvez soit définir la priorité du message sur une valeur numérique, soit laisser le message prendre la priorité par défaut de la file d'attente.

Vous définissez la priorité d'un message (dans la zone *Priority* de la structure MQMD) lorsque vous placez le message dans une file d'attente. Vous pouvez définir une valeur numérique pour la priorité ou laisser le message prendre la priorité par défaut de la file d'attente.

L'attribut **MsgDeliverySequence** de la file d'attente détermine si les messages de la file d'attente sont stockés dans la séquence FIFO (premier entré, premier sorti) ou dans la séquence FIFO dans la séquence de priorité. Si cet attribut est défini sur MQMDS_PRIORITY, les messages sont mis en file d'attente avec la priorité spécifiée dans la zone *Priority* de leurs descripteurs de message ; mais s'il est défini sur MQMDS_FIFO, les messages sont mis en file d'attente avec la priorité par défaut de la file d'attente. Les messages de priorité égale sont stockés dans la file d'attente par ordre d'arrivée.

L'attribut **DefPriority** d'une file d'attente définit la valeur de priorité par défaut pour les messages placés dans cette file d'attente. Cette valeur est définie lors de la création de la file d'attente, mais elle peut être modifiée par la suite. Les files d'attente alias et les définitions locales des files d'attente éloignées peuvent avoir des priorités par défaut différentes des files d'attente de base dans lesquelles elles sont résolues. S'il existe plusieurs définitions de file d'attente dans le chemin de résolution (voir «Résolution de nom», à la page 762), la priorité par défaut est extraite de la valeur (au moment de l'opération d'insertion) de l'attribut **DefPriority** de la file d'attente spécifiée dans la commande d'ouverture.

La valeur de l'attribut **MaxPriority** du gestionnaire de files d'attente correspond à la priorité maximale que vous pouvez affecter à un message traité par ce gestionnaire de files d'attente. Vous ne pouvez pas modifier la valeur de cet attribut. Dans IBM MQ, l'attribut a la valeur 9 ; vous pouvez créer des messages dont les priorités sont comprises entre 0 (la plus faible) et 9 (la plus élevée).

Propriétés des messages

Les propriétés de message permettent à une application de sélectionner des messages à traiter ou d'extraire des informations sur un message sans accéder aux en-têtes MQMD ou MQRFH2 . Ils facilitent également la communication entre les applications IBM MQ et JMS .

Une propriété de message est une donnée associée à un message, constituée d'un nom textuel et d'une valeur d'un type particulier. Les propriétés de message sont utilisées par les sélecteurs de message pour filtrer les publications dans les rubriques ou pour extraire de manière sélective les messages des files d'attente. Les propriétés de message peuvent être utilisées pour inclure des données métier ou des informations d'état sans avoir à les stocker dans les données d'application. Les applications n'ont pas besoin d'accéder aux données dans les en-têtes MQ Message Descriptor (MQMD) ou MQRFH2 car les zones de ces structures de données sont accessibles en tant que propriétés de message à l'aide des appels de fonction MQI (Message Queue Interface).

L'utilisation des propriétés de message dans IBM MQ imite l'utilisation des propriétés dans JMS. Cela signifie que vous pouvez définir des propriétés dans une application JMS et les extraire dans une application IBM MQ procédurale, ou l'inverse. Pour mettre une propriété à la disposition d'une application JMS , affectez-lui le préfixe "usr" ; il est alors disponible (sans le préfixe) en tant que propriété utilisateur de message JMS . Par exemple, la propriété IBM MQ *usr.myproperty* (chaîne de caractères) est accessible à une application JMS à l'aide de l'appel JMS `message.getStringProperty('myproperty')`. Notez que les applications JMS ne peuvent pas accéder aux propriétés avec le préfixe "usr" si elles contiennent au moins deux U+002E (".") caractères. Une propriété sans préfixe et sans U+002E (".") est traité comme s'il avait le préfixe "usr". A l'inverse, une propriété utilisateur définie dans une application JMS est accessible dans une application IBM MQ en ajoutant "usr". préfixe du nom de propriété demandé dans un appel MQINQMP.

Propriétés de message et longueur de message

Utilisez l'attribut de gestionnaire de files d'attente *MaxPropertiesLongueur* pour contrôler la taille des propriétés pouvant circuler avec n'importe quel message dans un gestionnaire de files d'attente IBM MQ .

En général, lorsque vous utilisez MQSETMP pour définir des propriétés, la taille d'une propriété correspond à la longueur du nom de propriété en octets, plus la longueur de la valeur de propriété en octets transmise dans l'appel MQSETMP. Il est possible que le jeu de caractères du nom de la propriété et la valeur de la propriété changent lors de la transmission du message à sa destination car ils peuvent être convertis en Unicode ; dans ce cas, la taille de la propriété peut changer.

Dans un appel MQPUT ou MQPUT1 , les propriétés du message ne sont pas prises en compte dans la longueur du message pour la file d'attente et le gestionnaire de files d'attente, mais elles sont prises en compte dans la longueur des propriétés telles qu'elles sont perçues par le gestionnaire de files d'attente (qu'elles aient été définies à l'aide des appels MQI de propriété de message ou non).

Si la taille des propriétés dépasse la longueur maximale des propriétés, le message est rejeté avec MQRC_PROPERTIES_TOO_BIG. Etant donné que la taille des propriétés dépend de leur représentation, vous devez définir la longueur maximale des propriétés à un niveau brut.

Il est possible pour une application d'insérer avec succès un message avec une mémoire tampon supérieure à la valeur de *MaxMsgLength*, si la mémoire tampon inclut des propriétés. En effet, même lorsqu'elles sont représentées en tant qu'éléments MQRFH2 , les propriétés de message ne sont pas prises en compte dans la longueur du message. Les zones d'en-tête MQRFH2 sont ajoutées à la longueur des propriétés uniquement si un ou plusieurs dossiers sont contenus et si chaque dossier de l'en-tête contient des propriétés. Si un ou plusieurs dossiers sont contenus dans l'en-tête MQRFH2 et qu'un dossier ne contient pas de propriétés, les zones d'en-tête MQRFH2 sont prises en compte dans la longueur du message.

Dans un appel MQGET, les propriétés du message ne sont pas prises en compte dans la longueur du message en ce qui concerne la file d'attente et le gestionnaire de files d'attente. Toutefois, comme les propriétés sont comptées séparément, il est possible que la mémoire tampon renvoyée par un appel MQGET soit supérieure à la valeur de l'attribut *MaxMsgLength* .

Ne demandez pas à vos applications d'interroger la valeur de *MaxMsgLength* , puis d'allouer une mémoire tampon de cette taille avant d'appeler MQGET. Au lieu de cela, allouez une mémoire tampon que vous

considérez comme suffisamment grande. En cas d'échec de MQGET, allouez une mémoire tampon guidée par la taille du paramètre *DataLength* .

Le paramètre *DataLength* de l'appel MQGET renvoie la longueur en octets des données d'application et toutes les propriétés renvoyées dans la mémoire tampon que vous avez fournie, si un descripteur de message n'est pas spécifié dans la structure MQGMO.

Le paramètre *Buffer* de l'appel MQPUT contient les données de message d'application à envoyer et toutes les propriétés représentées dans les données de message.

Il existe une limite de longueur de 100 Mo pour les propriétés de message, à l'exclusion du descripteur de message ou de l'extension pour chaque message.

La taille d'une propriété dans sa représentation interne correspond à la longueur du nom, plus la taille de sa valeur, plus des données de contrôle pour la propriété. Il existe également des données de contrôle pour l'ensemble de propriétés après l'ajout d'une propriété au message.

Noms de propriétés

Un nom de propriété est une chaîne de caractères. Certaines restrictions s'appliquent à sa longueur et à l'ensemble de caractères pouvant être utilisés.

Un nom de propriété est une chaîne de caractères sensible à la casse, limitée à +4095 caractères, sauf indication contraire du contexte. Cette limite est contenue dans la constante MQ_MAX_PROPERTY_NAME_LENGTH.

Si vous dépassez cette longueur maximale lors de l'utilisation d'un appel MQI de propriété de message, l'appel échoue avec le code anomalie MQRC_PROPERTY_NAME_LENGTH_ERR.

Etant donné qu'il n'existe pas de longueur maximale de nom de propriété dans JMS, il est possible pour une application JMS de définir un nom de propriété JMS valide qui n'est pas un nom de propriété IBM MQ valide lorsqu'il est stocké dans une structure MQRFH2 .

Dans ce cas, lors de l'analyse syntaxique, seuls les 4095 premiers caractères du nom de propriété sont utilisés ; les caractères suivants sont tronqués. Une application utilisant des sélecteurs risque alors de ne pas correspondre à une chaîne de sélection ou de ne pas correspondre à une chaîne alors qu'elle ne s'y attend pas, car plusieurs propriétés risquent d'être tronquées au même nom. Lorsqu'un nom de propriété est tronqué, WebSphereMQ émet un message de journal des erreurs.

Tous les noms de propriété doivent respecter les règles définies par la spécification de langage Java pour les identificateurs Java , à l'exception du caractère Unicode U+002E (.) qui est autorisé dans le nom-mais pas le début. Les règles pour les identificateurs Java sont identiques à celles contenues dans la spécification JMS pour les noms de propriété.

Les espaces et les opérateurs de comparaison sont interdits. Les valeurs nulles imbriquées sont autorisées dans un nom de propriété mais ne sont pas recommandées. Si vous utilisez des valeurs nulles imbriquées, cela empêche l'utilisation de la constante MQVS_NULL_TERMINATED lorsqu'elle est utilisée avec la structure MQCHARV pour spécifier des chaînes de longueur variable.

Gardez les noms de propriété simples car les applications peuvent sélectionner des messages en fonction des noms de propriété et la conversion entre le jeu de caractères du nom et du sélecteur peut entraîner un échec inattendu de la sélection.

Les noms de propriété IBM MQ utilisent le caractère U+002E (.) pour le regroupement logique des propriétés. Cela divise l'espace de nom pour les propriétés. Les propriétés avec les préfixes suivants, dans n'importe quel mélange de minuscules ou de majuscules, sont réservées à l'utilisation par le produit:

- `mcd`
- `jms`
- `usr`
- `mq`
- `sib`

- `wmq`
- `Root`
- `Body`
- `Properties`

Un bon moyen d'éviter les conflits de noms est de s'assurer que toutes les applications préfixent leurs propriétés de message avec leur nom de domaine Internet. Par exemple, si vous développez une application à l'aide du nom de domaine `ourcompany.com` , vous pouvez nommer toutes les propriétés avec le préfixe `com.ourcompany`. Cette convention de dénomination permet également de sélectionner facilement des propriétés ; par exemple, une application peut demander toutes les propriétés de message en commençant par `com.ourcompany.%`.

Pour plus d'informations sur l'utilisation des noms de propriété, voir Restrictions relatives aux noms de propriété .

Restrictions de nom de propriété
Lorsque vous nommez une propriété, vous devez respecter certaines règles.

Les restrictions suivantes s'appliquent aux noms de propriété:

1. Une propriété ne doit pas commencer par les chaînes suivantes:
   - "JMS"-réservé à l'utilisation par IBM MQ classes for JMS.
   - "usr.JMS"-non valide.

   Les seules exceptions sont les propriétés suivantes qui fournissent des synonymes pour les propriétés JMS :

Propriété	**synonyme de**
JMSCorrelationID	Root.MQMD.CorrelId ou jms.Cid
JMSDeliveryMode	Root.MQMD.Persistence ou jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Root.MQMD.Expiry ou jms.Exp
JMSMessageID	Root.MQMD.MsgId
JMSPriority	Root.MQMD.Priority ou jms.Pri
JMSRedelivered	Root.MQMD.BackoutCount
JMSReplyTo (chaîne codée en tant qu'URI)	Root.MQMD.ReplyToQ ou Root.MQMD.ReplyToQMgr ou jms.Rto
JMSTimestamp	Root.MQMD.PutDate ou Root.MQMD.PutTime ou jms.Tms
JMSType	mcd.Type ou mcd.Set ou mcd.Fmt
JMSXAppID	Root.MQMD.PutApplName
JMSXDeliveryCount	Root.MQMD.BackoutCount
JMSXGroupID	Root.MQMD.GroupId ou jms.Gid
JMSXGroupSeq	Root.MQMD.MsgSeqNumber ou jms.Seq
JMSXUserID	Root.MQMD.UserIdentifier

   Ces synonymes permettent à une application MQI d'accéder aux propriétés JMS de la même manière qu'une application client IBM MQ classes for JMS . Parmi ces propriétés, seules JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupIDet JMSXGroupSeq peuvent être définies à l'aide de l'interface MQI.

Notez que les propriétés JMS_IBM_ * disponibles dans IBM MQ classes for JMS ne sont pas disponibles à l'aide de l'interface MQI. Les zones référencées par les propriétés JMS_IBM_ * sont accessibles d'une autre manière par les applications MQI.

2. Une propriété ne doit pas être appelée, dans une combinaison de minuscules ou de majuscules, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" et "ESCAPE". Il s'agit des noms des mots clés SQL utilisés dans les chaînes de sélection.
3. Un nom de propriété commençant par " mq " dans n'importe quel mélange de minuscules ou de majuscules et ne commençant pas par "mq_usr" ne peut contenir qu'un seul "." (U+002E). Plusieurs "." Les caractères ne sont pas autorisés dans les propriétés avec ces préfixes.
4. Deux "." Les caractères doivent contenir d'autres caractères entre eux ; vous ne pouvez pas avoir de point vide dans la hiérarchie. De même, un nom de propriété ne peut pas se terminer par un "." (?).
5. Si une application définit la propriété "a.b", puis la propriété "a.b.c", il n'est pas clair si la hiérarchie "b" contient une valeur ou un autre regroupement logique. Une telle hiérarchie est un "contenu mixte", ce qui n'est pas pris en charge. La définition d'une propriété qui provoque un contenu mixte n'est pas autorisée.

Ces restrictions sont appliquées par le mécanisme de validation comme suit:

- Les noms de propriété sont validés lors de la définition d'une propriété à l'aide de l'appel MQSETMP- Définition de la propriété de message , si la validation a été demandée lors de la création du descripteur de message. Si une tentative de validation d'une propriété est effectuée et échoue en raison d'une erreur dans la spécification du nom de la propriété, le code achèvement est MQCC_FAILED avec la raison suivante:
  - MQRC_PROPERTY_NAME_ERROR pour les raisons 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED pour la raison 5.
- Il n'est pas garanti que les noms des propriétés spécifiées directement en tant qu'éléments MQRFH2 soient validés par l'appel MQPUT.

Zones de descripteur de message en tant que propriétés
La plupart des zones de descripteur de message peuvent être traitées comme des propriétés. Le nom de la propriété est construit en ajoutant un préfixe au nom de la zone de descripteur de message.

Si une application MQI souhaite identifier une propriété de message contenue dans une zone de descripteur de message, par exemple, dans une chaîne de sélecteur ou à l'aide des API de propriété de message, utilisez la syntaxe suivante:

Nom de la propriété	**Zone de descripteur de message**
Root.MQMD.*Zone*	*Zone*

Spécifiez *Field* avec la même casse que pour les zones de structure MQMD dans la déclaration de langage C. Par exemple, le nom de propriété `Root.MQMD.AccountingToken` accède à la zone `AccountingToken` du descripteur de message.

Les zones `StrucId` et `Version` du descripteur de message ne sont pas accessibles à l'aide de la syntaxe indiquée.

Les zones de descripteur de message ne sont jamais représentées dans un en-tête MQRFH2 comme pour les autres propriétés.

Si les données de message démarrent avec un MQMDE qui est honoré par le gestionnaire de files d'attente, les zones MQMDE sont accessibles à l'aide de la notation `Root.MQMD.`*`Field`* décrite. Dans ce cas, les zones MQMDE sont traitées comme faisant logiquement partie du MQMD du point de vue des propriétés. Voir Présentation de MQMDE.

Types et valeurs de données de propriété

Une propriété peut être une valeur booléenne, une chaîne d'octets, une chaîne de caractères ou un nombre à virgule flottante ou entier. La propriété peut stocker toute valeur valide dans la plage du type de données, sauf indication contraire du contexte.

Le type de données d'une valeur de propriété doit être l'une des valeurs suivantes:

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

Une propriété peut exister mais n'a pas de valeur définie ; il s'agit d'une propriété null. Une propriété nulle est différente d'une propriété d'octet (MQBYTE [ ]) ou d'une propriété de chaîne de caractères (MQCHAR [ ]) dans la mesure où elle possède une valeur définie mais vide, c'est-à-dire une valeur de longueur nulle.

La chaîne d'octets n'est pas un type de données de propriété valide dans JMS ou XMS. Il est conseillé de ne pas utiliser les propriétés de chaîne d'octets dans le dossier *usr* .

Sélection de messages dans les files d'attente

Vous pouvez sélectionner des messages dans des files d'attente à l'aide des zones MsgId et CorrelId dans un appel MQGET ou à l'aide d'une chaîne de sélection (SelectionString) dans un appel MQOPEN ou MQSUB.

Sélecteurs

Un sélecteur de message est une chaîne de longueur variable utilisée par une application pour enregistrer son intérêt uniquement dans les messages dont les propriétés correspondent à la requête SQL (Structured Query Language) représentée par la chaîne de sélection.

Sélection à l'aide des appels de fonction MQSUB et MQOPEN

Vous utilisez *SelectionString*, qui est une structure de type MQCHARV, pour effectuer des sélections à l'aide des appels MQSUB et MQOPEN.

La structure *SelectionString* permet de transmettre une chaîne de sélection de longueur variable au gestionnaire de files d'attente.

Le CCSID associé à la chaîne de sélecteur est défini via la zone VSCCSID de la structure MQCHARV. La valeur utilisée doit être un CCSID pris en charge pour les chaînes de sélecteur. Voir Conversion de page de codes pour obtenir la liste des pages de codes prises en charge.

La spécification d'un CCSID pour lequel il n'existe aucune conversion Unicode prise en charge par IBM MQ génère une erreur MQRC_SOURCE_CCSID_ERROR. Cette erreur est renvoyée lorsque le sélecteur est présenté au gestionnaire de files d'attente, c'est-à-dire sur l'appel MQSUB, MQOPEN ou MQPUT1 .

La valeur par défaut de la zone *VSCCSID* est MQCCSI_APPL, ce qui indique que le CCSID de la chaîne de sélection est égal au CCSID du gestionnaire de files d'attente ou au CCSID du client s'il est connecté via un client. La constante MQCCSI_APPL peut toutefois être remplacée par une application qui la redéfinit avant la compilation.

Si le sélecteur MQCHARV représente une chaîne NULL, aucune sélection n'est effectuée pour ce consommateur de message et les messages sont distribués comme si un sélecteur n'avait pas été utilisé.

La longueur maximale d'une chaîne de sélection est limitée uniquement par ce qui peut être décrit par la zone MQCHARV *VSLength*.

SelectionString est renvoyé dans la sortie d'un appel MQSUB à l'aide de l'option d'abonnement MQSO_RESUME, si vous avez fourni une mémoire tampon et qu'il existe une longueur de mémoire tampon positive dans VSBufSize. Si vous ne fournissez pas de mémoire tampon, seule la longueur de la chaîne de

sélection est renvoyée dans la zone VSLength de MQCHARV. Si la mémoire tampon fournie est inférieure à l'espace requis pour renvoyer la zone, seuls les octets VSBufSize sont renvoyés dans la mémoire tampon fournie.

Une application ne peut pas modifier une chaîne de sélection sans fermer au préalable l'identificateur de la file d'attente (pour MQOPEN) ou l'abonnement (pour MQSUB). Une nouvelle chaîne de sélection peut ensuite être spécifiée lors d'un appel MQOPEN ou MQSUB ultérieur.

MQOPEN
Utilisez MQCLOSE pour fermer le descripteur ouvert, puis indiquez une nouvelle chaîne de sélection lors d'un appel MQOPEN ultérieur.

MQSUB
Utilisez MQCLOSE pour fermer le descripteur d'abonnement renvoyé (hSub), puis indiquez une nouvelle chaîne de sélection lors d'un appel MQSUB ultérieur.

La Figure 3, à la page 34 montre le processus de sélection à l'aide de l'appel MQSUB.

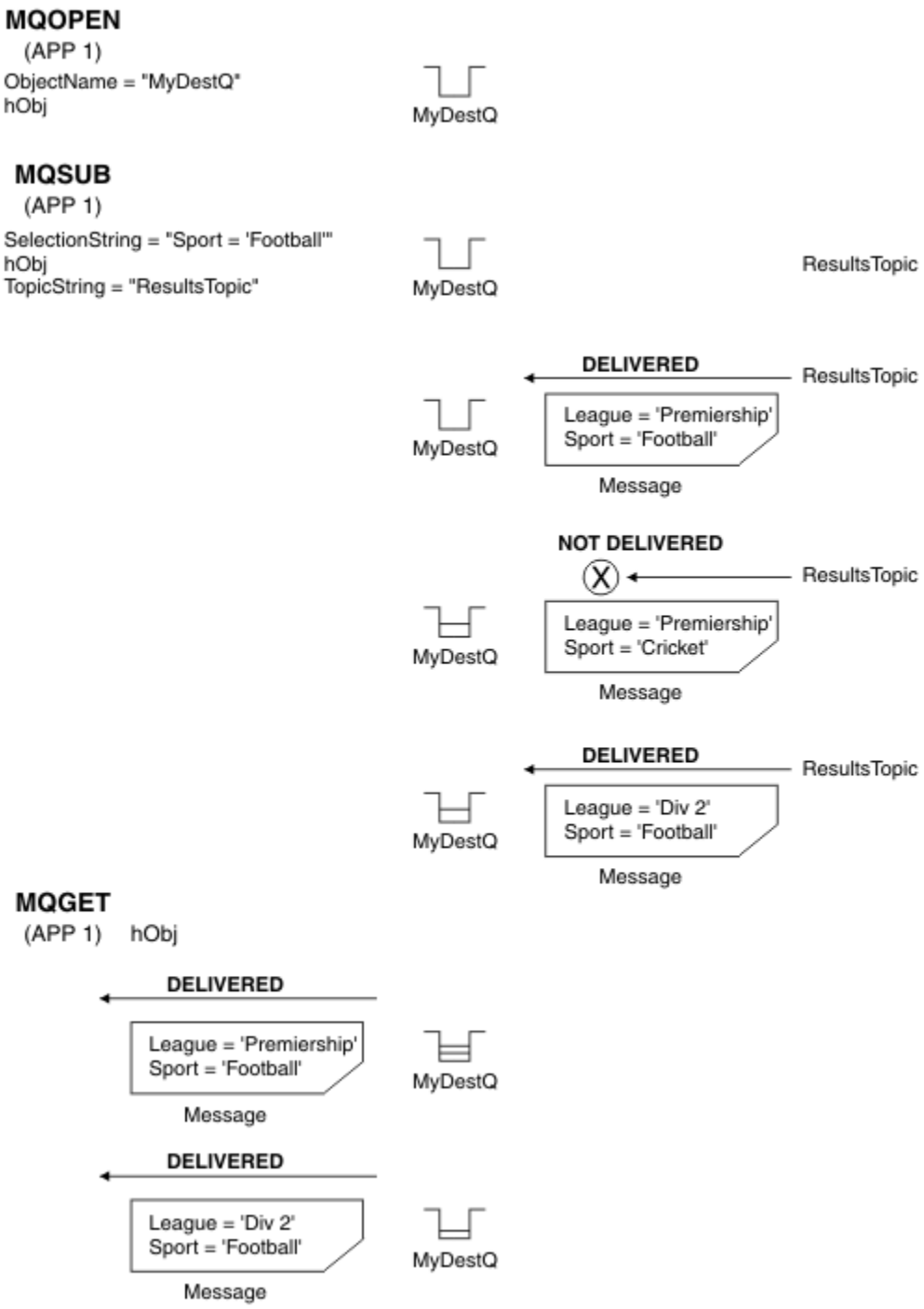


Figure 3. Sélection à l'aide de l'appel MQSUB

Un sélecteur peut être transmis lors de l'appel à MQSUB à l'aide de la zone *SelectionString* de la structure MQSD. La transmission d'un sélecteur sur MQSUB a pour effet que seuls les messages publiés dans la rubrique à laquelle vous êtes abonné, qui correspondent à une chaîne de sélection fournie, sont rendus disponibles dans la file d'attente de destination.

La Figure 4, à la page 35 montre le processus de sélection à l'aide de l'appel MQOPEN.

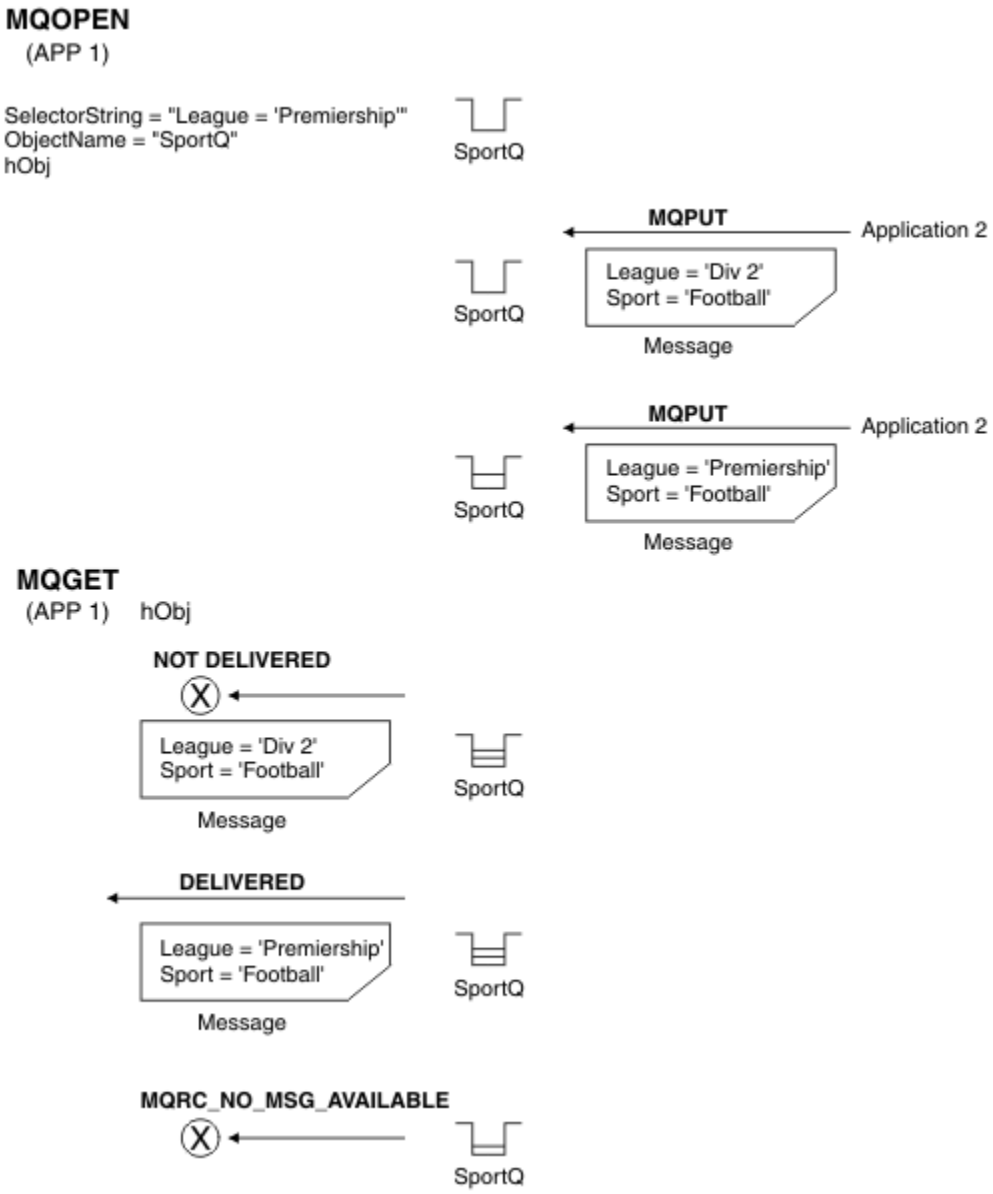


Figure 4. Sélection à l'aide de l'appel MQOPEN

Un sélecteur peut être transmis lors de l'appel à MQOPEN à l'aide de la zone *SelectionString* de la structure MQOD. La transmission d'un sélecteur sur l'appel MQOPEN a pour effet que seuls les messages de la file d'attente ouverte, qui correspondent à un sélecteur, sont distribués au consommateur de message.

L'utilisation principale du sélecteur dans l'appel MQOPEN concerne le cas point à point où une application peut choisir de recevoir uniquement les messages d'une file d'attente qui correspondent à un sélecteur. L'exemple précédent illustre un scénario simple dans lequel deux messages sont insérés dans une file d'attente ouverte par MQOPEN, mais un seul est reçu par l'application qui l'obtient, car il s'agit du seul qui correspond à un sélecteur.

Notez que les appels MQGET suivants génèrent MQRC_NO_MSG_AVAILABLE car il n'existe aucun autre message dans la file d'attente correspondant au sélecteur indiqué.

Concepts associés

«Règles et restrictions des chaînes de sélection», à la page 42
Familiarisez-vous avec ces règles sur la façon dont les chaînes de sélection sont interprétées et les restrictions de caractères pour éviter les problèmes potentiels lors de l'utilisation de sélecteurs.

Comportement de la sélection

Présentation du comportement de la sélection IBM MQ .

Les zones d'une structure MQMDE sont considérées comme étant les propriétés de message pour les propriétés de descripteur de message correspondantes si MQMD:

- A le format MQFMT_MD_EXTENSION
- Est immédiatement suivi d'une structure MQMDE valide
- Correspond à la version 1 ou contient uniquement les zones de la version 2 par défaut

Il est possible qu'une chaîne de sélection soit résolue en TRUE ou FALSE avant toute correspondance avec les propriétés de message. Par exemple, cela peut être le cas si la chaîne de sélection est définie sur `"TRUE <> FALSE"`. Cette évaluation précoce est garantie uniquement lorsqu'il n'y a pas de références de propriété de message dans la chaîne de sélection.

Si une chaîne de sélection est résolue en TRUE avant que des propriétés de message ne soient prises en compte, tous les messages publiés dans la rubrique à laquelle le consommateur est abonné sont distribués. Si une chaîne de sélection est résolue en FALSE avant que des propriétés de message ne soient prises en compte, le code anomalie MQRC_SELECTOR_ALWAYS_FALSE et le code achèvement MQCC_FAILED sont renvoyés sur l'appel de fonction qui a présenté le sélecteur.

Même si un message ne contient pas de propriétés de message (autres que les propriétés d'en-tête), il peut être sélectionné. Si une chaîne de sélection fait référence à une propriété de message qui n'existe pas, cette propriété est supposée avoir la valeur NULL ou'Inconnu'.

Par exemple, un message peut toujours satisfaire une chaîne de sélection telle que `'Color IS NULL'`, où `'Color'` n'existe pas en tant que propriété de message dans le message.

La sélection ne peut être effectuée que sur les propriétés associées à un message, et non sur le message lui-même, sauf si un fournisseur de sélection de message étendu est disponible. La sélection peut être effectuée sur la charge de message uniquement si un fournisseur de sélection de message étendu est disponible.

Chaque propriété de message est associée à un type. Lorsque vous effectuez une sélection, vous devez vous assurer que les valeurs utilisées dans les expressions pour tester les propriétés de message sont du type correct. Si une non-concordance de type se produit, l'expression en question se résout en FALSE.

Il est de votre responsabilité de vous assurer que la chaîne de sélection et les propriétés de message utilisent des types compatibles.

Les critères de sélection continuent d'être appliqués pour le compte des abonnés durables inactifs, de sorte que seuls les messages correspondant à la chaîne de sélection fournie à l'origine soient conservés.

Les chaînes de sélection ne peuvent pas être modifiées lorsqu'un abonnement durable est repris avec alter (MQSO_ALTER). Si une chaîne de sélection différente est présentée lorsqu'un abonné durable reprend l'activité, MQRC_SELECTOR_NOT_ALTERABLE est renvoyé à l'application.

Les applications reçoivent le code retour MQRC_NO_MSG_AVAILABLE si aucune file d'attente ne répond aux critères de sélection.

Si une application a spécifié une chaîne de sélection contenant des valeurs de propriété, seuls les messages contenant des propriétés correspondantes peuvent être sélectionnés. Par exemple, un abonné spécifie une chaîne de sélection de "a = 3" et un message est publié ne contenant aucune propriété, ou des propriétés où'a'n'existe pas ou n'est pas égal à 3. L'abonné ne reçoit pas ce message dans sa file d'attente de destination.

Performances de messagerie

La sélection de messages dans une file d'attente nécessite que IBM MQ inspecte séquentiellement chaque message de la file d'attente. Les messages sont inspectés jusqu'à ce qu'un message correspondant aux critères de sélection soit trouvé ou qu'il n'y ait plus de messages à examiner. Par conséquent, les performances de la messagerie sont affectées si la sélection de messages est utilisée dans les files d'attente profondes.

Pour optimiser la sélection de messages dans les files d'attente profondes lorsque la sélection est basée sur JMSCorrelationID ou JMSMessageID, utilisez une chaîne de sélection au format suivant:

- JMSCorrelationID = 'ID:*id_corrélation*'
- JMSMessageID= 'ID:*id_message*'

où :

- *correlation_id* est une chaîne contenant un identificateur de corrélation IBM MQ standard.
- *message_id* est une chaîne contenant un identificateur de message IBM MQ standard.

Remarque : Le sélecteur ne doit faire référence qu'à l'une des propriétés. L'utilisation d'un sélecteur doté de l'un de ces formats offre une amélioration significative des performances lors de la sélection sur JMSCorrelationID et une amélioration marginale des performances pour JMSMessageID. Pour plus d'informations, voir «Sélecteurs de message dans JMS», à la page 149.

Utilisation de sélecteurs complexes

Les sélecteurs peuvent contenir de nombreux composants, par exemple:

a et b ou c et d ou e et f ou g et h ou i et j... ou y et z

L'utilisation de ces sélecteurs complexes peut avoir des conséquences graves sur les performances et des besoins excessifs en ressources. En tant que tel, IBM MQ protège le système en ne traitant pas les sélecteurs trop complexes qui pourraient entraîner une pénurie de ressources système. La protection peut se produire sur les chaînes de sélection contenant plus de 100 tests ou lorsque IBM MQ détecte que la limite de taille de la pile du système d'exploitation est atteinte. Vous devez essayer de tester en détail l'utilisation de chaînes de sélection avec de nombreux composants, sur les plateformes appropriées, afin de vous assurer que les limites de protection ne sont pas atteintes.

Les performances et la complexité des sélecteurs peuvent être améliorées en les simplifiant à l'aide de parenthèses supplémentaires pour combiner des composants. Exemple :

(a et b ou c et d) ou (e et f ou g et h) ou (i et j) ...

Concepts associés

«Règles et restrictions des chaînes de sélection», à la page 42
Familiarisez-vous avec ces règles sur la façon dont les chaînes de sélection sont interprétées et les restrictions de caractères pour éviter les problèmes potentiels lors de l'utilisation de sélecteurs.

Syntaxe du sélecteur de message

Un sélecteur de message IBM MQ est une chaîne dont la syntaxe est basée sur un sous-ensemble de la syntaxe d'expression conditionnelle SQL92 .

L'ordre dans lequel un sélecteur de message est évalué est de gauche à droite dans un niveau de priorité. Vous pouvez utiliser des parenthèses pour modifier cet ordre. Les littéraux de sélecteur et les noms d'opérateur prédéfinis sont écrits ici en majuscules ; toutefois, ils ne sont pas sensibles à la casse.

Si le sélecteur est fourni via l'API, IBM MQ vérifie l'exactitude syntaxique d'un sélecteur de message au moment de sa présentation. Si la syntaxe de la chaîne de sélection est incorrecte ou si un nom de propriété n'est pas valide et qu'un fournisseur de sélection de message étendu n'est pas disponible, MQRC_SELECTION_NOT_AVAILABLE est renvoyé à l'application. Si la syntaxe de la chaîne de sélection est incorrecte ou qu'un nom de propriété n'est pas valide lors de la reprise d'un abonnement, un MQRC_SELECTOR_SYNTAX_ERROR est renvoyé à l'application. Si la validation de nom de propriété a été désactivée lorsque la propriété a été définie (en définissant MQCMHO_NONE à la place de MQCMHO_VALIDATE) et qu'une application insère ensuite un message avec un nom de propriété non valide, ce message n'est jamais sélectionné.

Aucune erreur n'est renvoyée lorsque le sélecteur est présenté si IBM MQ détermine qu'un sélecteur d'abonnement défini par l'administrateur utilise la syntaxe de message étendue, comme indiqué par le paramètre **DISPLAY SUB SELTYPE** ayant la valeur EXTENDED. Dans ce cas, la

vérification de la syntaxe de la chaîne de sélection est différée jusqu'à l'heure de publication (voir MQRC_SELECTION_NOT_AVAILABLE).

Un sélecteur peut contenir:

- Littéraux :
  - Les littéraux chaîne sont placés entre apostrophes. Deux guillemets simples consécutifs représentent un guillemet simple. Par exemple,'literal'et'literal''s'. Comme les littéraux chaîne Java , ils utilisent le codage de caractères Unicode. Vous ne pouvez pas utiliser de guillemets pour entourer un littéral chaîne. Toute séquence d'octets peut être utilisée entre les apostrophes.
  - Une chaîne d'octets est une ou plusieurs paires de caractères hexadécimaux placées entre guillemets et préfixées par 0x. Les exemples sont "0x2F1C" ou "0XD43A". La longueur d'une chaîne d'octets doit être d'au moins un octet. Si une chaîne d'octets de sélecteur est mise en correspondance avec une propriété de message de type MQTYPE_BYTE_STRING, aucune action spéciale n'est effectuée sur le zéro de début ou de fin. Les octets sont traités comme un autre caractère. L'endiannité n'est pas non plus prise en compte. La longueur des chaînes d'octets de sélecteur et de propriété doit être égale et la séquence d'octets doit être la même.

    Voici des exemples de sélections de chaînes d'octets (par exemple, *myBytes* = 0AFC23) qui correspondent:

    - "myBytes = "0x0AFC23"" = TRUE

    Les sélections de chaîne suivantes ne correspondent pas:

    - "myBytes = "0xAFC23"" = MQRC_SELECTOR_SYNTAX_ERROR (car le nombre d'octets n'est pas multiple de deux)
    - "myBytes = "0x0AFC2300"" = FALSE (car le zéro de fin est significatif dans la comparaison)
    - "myBytes = "0x000AFC23"" = FALSE (car le zéro non significatif est significatif dans la comparaison)
    - "myBytes = "0x23FC0A"" = FALSE (car endianness n'est pas pris en compte)
  - Les nombres hexadécimaux commencent par un zéro, suivi d'une majuscule ou d'une minuscule x. Le reste du littéral contient un ou plusieurs caractères hexadécimaux admis. Exemples: 0xA, 0xAF, 0X2020.
  - Un zéro de début suivi d'un ou de plusieurs chiffres compris entre 0 et 7 est toujours interprété comme étant le début d'un nombre octal. Vous ne pouvez pas représenter un nombre décimal à préfixe zéro comme celui-ci. Par exemple, 09 renvoie une erreur de syntaxe car 9 n'est pas un chiffre octal valide. Exemples de nombres octaux: 0177, 0713.
  - Un littéral numérique exact est une valeur numérique sans séparateur décimal, telle que 57, -957et +62. Un littéral numérique exact peut avoir un L majuscule ou minuscule de fin ; cela n'affecte pas la façon dont le nombre est stocké ou interprété. IBM MQ prend en charge les chiffres exacts compris entre -9,223,372,036,854,775,808 et 9,223,372,036,854,775,807.
  - Un littéral numérique approximatif est une valeur numérique en notation scientifique, telle que 7E3 ou -57.9E2, ou une valeur numérique avec une décimale, telle que 7., -95.7ou +6.2. IBM MQ prend en charge les nombres compris entre -1.797693134862315E+308 et 1.797693134862315E+308.

    La signification doit suivre un signe facultatif (+ ou -). Le significande doit être un entier ou une fraction. Une partie fractionnaire de la signification n'a pas besoin d'un chiffre de tête.

    Une valeur en majuscule ou en minuscule E indique le début d'un exposant facultatif. L'exposant a une base décimale et la partie numérique de l'exposant peut être préfixée par un caractère de signe facultatif.

    Les littéraux numériques approximatifs peuvent être terminés par un caractère F ou D (non sensible à la casse). Cette syntaxe permet de prendre en charge la méthode multilingue de balisage des nombres à précision simple ou double. Ces caractères sont facultatifs et n'affectent pas la façon dont un littéral numérique approximatif est stocké ou traité. Ces nombres sont toujours stockés et traités avec une double précision.

- Les littéraux booléens TRUE et FALSE.

Remarque : Les représentations IEEE-754 non finies telles que `NaN`, `+Infinity`, `-Infinity` ne sont pas prises en charge dans les chaînes de sélection. Il n'est donc pas possible d'utiliser ces valeurs comme opérandes dans une expression. Le zéro négatif est traité de la même manière que le zéro positif pour les opérations mathématiques.

- Identificateurs :

  Un identificateur est une séquence de caractères de longueur variable qui doit commencer par un caractère de début d'identificateur valide, suivi de zéro ou plusieurs caractères de partie d'identificateur valides. Les règles pour les noms d'identificateur sont les mêmes que celles pour les noms de propriété de message. Pour plus d'informations, voir «Noms de propriétés», à la page 29 et «Restrictions de nom de propriété», à la page 30 .

  Remarque : La sélection peut être effectuée sur la charge de message uniquement si un fournisseur de sélection de message étendu est disponible.

  Les identificateurs sont des références de zone d'en-tête ou des références de propriété. Le type d'une valeur de propriété dans un sélecteur de message doit correspondre au type utilisé pour définir la propriété, bien que la promotion numérique soit effectuée dans la mesure du possible. Si une non-concordance de type se produit, le résultat de l'expression est FALSE. Si une propriété qui n'existe pas dans un message est référencée, sa valeur est `NULL`.

  Les conversions de type qui s'appliquent aux méthodes get pour les propriétés ne s'appliquent pas lorsqu'une propriété est utilisée dans une expression de sélecteur de message. Par exemple, si vous définissez une propriété en tant que valeur de chaîne, puis que vous utilisez un sélecteur pour l'interroger en tant que valeur numérique, l'expression renvoie FALSE.

  Les noms de zone et de propriété JMS qui sont mappés à des noms de propriété ou des noms de zone `MQMD` sont également des identificateurs valides dans une chaîne de sélection. IBM MQ mappe les noms de zone et de propriété JMS reconnus aux valeurs de propriété de message. Pour plus d'informations, voir «Sélecteurs de message dans JMS», à la page 149. Par exemple, la chaîne de sélection `"JMSPriority >="` est sélectionnée dans la propriété `Pri` qui se trouve dans le dossier `jms` du message en cours.

- Dépassement / dépassement négatif:

  Pour les nombres décimaux et les nombres approximatifs, les conditions suivantes ne sont pas définies:

  - Spécification d'un nombre en dehors de la plage définie
  - Spécification d'une expression arithmétique qui provoquerait un dépassement ou un dépassement négatif

  Aucune vérification n'est effectuée pour ces conditions.

- Blanc :

  Défini sous la forme d'un espace, d'un saut de page, d'une nouvelle ligne, d'un retour chariot, d'une tabulation horizontale ou d'une tabulation verticale. Les caractères Unicode suivants sont reconnus comme des espaces:

  - `\u0009 to \u000D`
  - `\u0020`
  - `\u001C`
  - `\u001D`
  - `\u001E`
  - `\u001F`
  - `\u1680`
  - `\u180E`
  - `\u2000` à `\u200A`
  - `\u2028`

  - \u2029
  - \u202F
  - \u205F
  - \u3000
- Expressions :
  - Un sélecteur est une expression conditionnelle. Un sélecteur qui a la valeur true correspond ; un sélecteur qui a la valeur false ou unknown ne correspond pas.
  - Les expressions arithmétiques sont composées d'elles-mêmes, d'opérations arithmétiques, d'identificateurs (la valeur d'identificateur est traitée comme un littéral numérique) et de littéraux numériques.
  - Les expressions conditionnelles sont composées d'elles-mêmes, d'opérations de comparaison et d'opérations logiques.
- La méthode standard bracketing (), qui permet de définir l'ordre dans lequel les expressions sont évaluées, est prise en charge.
- Opérateurs logiques dans l'ordre de priorité: NOT, AND, OR.
- Opérateurs de comparaison: =, >, >=, <, <=, <> (différent de).
  - Deux chaînes d'octets sont égales uniquement si les chaînes sont de même longueur et si la séquence d'octets est égale.
  - Seules les valeurs de même type peuvent être comparées. Une exception est qu'il est valide pour comparer des valeurs numériques exactes et des valeurs numériques approximatives (la conversion de type requise est définie par les règles de la promotion numérique Java). S'il y a une tentative de comparaison de différents types, le sélecteur est toujours faux.
  - La comparaison des chaînes et des booléens est limitée à = et <>. Deux chaînes sont égales uniquement si elles contiennent la même séquence de caractères.
- Opérateurs arithmétiques dans l'ordre de priorité:
  - +, - unaire.
  - Multiplication * et division / .
  - + addition et - soustraction.
  - Les opérations arithmétiques sur une valeur NULL ne sont pas prises en charge. S'ils sont tentés, le sélecteur complet est toujours faux.
  - Les opérations arithmétiques doivent utiliser la promotion numérique Java .
- Opérateur de comparaison arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 et arithmetic-expr3 :
  - `Age BETWEEN 15 and 19` est équivalent à `age>= 15 AND age <= 19`.
  - `Age NOT BETWEEN 15 and 19` est équivalent à `age < 15 OR age > 19`.
  - Si l'une des expressions d'une opération BETWEEN est NULL, la valeur de l'opération est false. Si l'une des expressions d'une opération `NOT BETWEEN` est NULL, la valeur de l'opération est true.
- Opérateur de comparaison d'identificateur `[NOT] IN (string-literal1, string-literal2,...)` dans lequel l'identificateur a une chaîne ou une valeur NULL .
  - `Country IN ('UK', 'US', 'France')` est true pour `'UK'` et false pour `'Peru'`. Elle est équivalente à l'expression `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` est false pour `'UK'` et true pour `'Peru'`. Elle est équivalente à l'expression `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Si l'identificateur d'une opération IN ou `NOT IN` est NULL, la valeur de l'opération est inconnue.
- Opérateur de comparaison `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` , où `identifier` a une valeur de chaîne. *pattern-value* est un littéral chaîne, où _

représente un caractère unique et % représente une séquence de caractères (y compris la séquence vide). Tous les autres personnages se tiennent pour eux-mêmes. Le *caractère d'échappement* facultatif est un littéral chaîne de caractères unique utilisé pour mettre en échappement la signification spéciale de _ et % dans *valeur-modèle*. L'opérateur LIKE doit être utilisé uniquement pour comparer deux valeurs de chaîne.

- `phone LIKE '12%3'` est vrai pour 123 et 12993 et faux pour 1234.
- `word LIKE 'l_se'` est vrai pour la perte et faux pour la perte.
- `underscored LIKE '\_%' ESCAPE '\'` est true pour `_foo` et false pour `bar`.
- `phone NOT LIKE '12%3'` est false pour 123 et 12993 et true pour 1234.
- Si l'identificateur d'une opération LIKE ou `NOT LIKE` est NULL, la valeur de l'opération est inconnue.

Remarque : L'opérateur LIKE doit être utilisé pour comparer deux valeurs de chaîne. La valeur de `Root.MQMD.CorrelId` est un tableau de 24 octets et non une chaîne de caractères. La chaîne de sélecteur `Root.MQMD.CorrelId LIKE 'ABC%'` est acceptée par l'analyseur syntaxique comme étant syntaxiquement valide, mais elle est évaluée à false. Lorsque vous comparez un tableau d'octets à une chaîne de caractères, LIKE ne peut donc pas être utilisé.

- L'opérateur de comparaison `identifier IS NULL` teste une valeur de zone d'en-tête NULL ou une valeur de propriété manquante.
- L'opérateur de comparaison `identifier IS NOT NULL` teste l'existence d'une valeur de zone d'en-tête non nulle ou d'une valeur de propriété.
- Valeurs nulles

  L'évaluation des expressions de sélecteur qui contiennent des valeurs NULL est définie par la sémantique SQL 92 NULL , en résumé:

  - SQL traite une valeur NULL comme inconnue.
  - La comparaison ou l'arithmétique avec une valeur inconnue donne toujours une valeur inconnue.
  - Les opérateurs `IS NULL` et `IS NOT NULL` convertissent une valeur inconnue en valeurs TRUE et FALSE .

  Les opérateurs booléens utilisent une logique à trois valeurs (T=TRUE, F=FALSE, U=UNKNOWN)

Tableau 1. Valeur du résultat de l'opérateur booléen lorsque la logique est A AND B

Opérateur A	**Opérateur B**	**Résultat (A et 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

Tableau 2. Valeur du résultat de l'opérateur booléen lorsque la logique est A OR B

Opérateur A	**Opérateur B**	**Résultat (A OR B)**
T	F	T
T	U	T

Tableau 2. Valeur du résultat de l'opérateur booléen lorsque la logique est A OR B (suite)		
Opérateur A	**Opérateur B**	**Résultat (A OR B)**
T	T	T
F	T	T
F	U	U
F	F	F
U	T	T
U	U	U
U	F	U

Tableau 3. Valeur du résultat de l'opérateur booléen lorsque la logique est NOT A	
Opérateur A	**Résultat (NOT A)**
T	F
F	T
U	U

Le sélecteur de message suivant sélectionne les messages dont le type de message est voiture, la couleur bleue et le poids est supérieur à 2500 lbs:

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

Bien que SQL prenne en charge la comparaison décimale fixe et l'arithmétique, les sélecteurs de message ne le font pas. C'est pourquoi les littéraux numériques exacts sont limités à ceux sans décimale. C'est également la raison pour laquelle il existe des valeurs numériques avec une décimale comme représentation alternative pour une valeur numérique approximative.

Les commentaires SQL ne sont pas pris en charge.

Concepts associés

«Propriétés des messages», à la page 28
Les propriétés de message permettent à une application de sélectionner des messages à traiter ou d'extraire des informations sur un message sans accéder aux en-têtes MQMD ou MQRFH2 . Ils facilitent également la communication entre les applications IBM MQ et JMS .

Référence associée

MsgHandle

MQBUFMH-Conversion de la mémoire tampon en descripteur de message

Règles et restrictions des chaînes de sélection

Familiarisez-vous avec ces règles sur la façon dont les chaînes de sélection sont interprétées et les restrictions de caractères pour éviter les problèmes potentiels lors de l'utilisation de sélecteurs.

- La sélection de messages pour la messagerie de publication / abonnement a lieu sur le message envoyé par le diffuseur de publications. Voir Chaînes de sélection.
- L'équivalence est testée à l'aide d'un seul caractère égal à ; par exemple, a = b est correct, alors que a == b est incorrect.
- Un opérateur utilisé par de nombreux langages de programmation pour représenter'différent de'est !=. Cette représentation n'est pas un synonyme valide pour <> ; par exemple, a <> b est valide, alors que a != b n'est pas valide.

- Les apostrophes ne sont reconnues que si les' (U+0027) est utilisé. De même, les guillemets, valides uniquement lorsqu'ils sont utilisés pour entourer des chaînes d'octets, doivent utiliser le caractère " (U+0022).
- Les symboles &, &&, | et | | ne sont pas synonymes de conjonction logique / disjonction; par exemple, a && b doit être spécifié en tant que a AND b.
- Les caractères génériques * et ? ne sont pas synonymes de % et _.
- Les sélecteurs contenant des expressions composées telles que 20 < b < 30 ne sont pas valides. L'analyseur syntaxique évalue les opérateurs qui ont la même priorité de gauche à droite. L'exemple deviendrait donc (20 < b) < 30, ce qui n'a pas de sens. A la place, l'expression doit être écrite sous la forme (b > 20) AND (b < 30).
- Les chaînes d'octets doivent être placées entre guillemets ; si des guillemets simples sont utilisés, la chaîne d'octets est considérée comme un littéral chaîne. Le nombre de caractères (et non le nombre que les caractères représentent) qui suit le 0x doit être un multiple de deux.
- Le mot clé IS n'est pas synonyme du caractère égal à. Par conséquent, les chaînes de sélection a IS 3 et b IS 'red' ne sont pas valides. Le mot clé IS existe uniquement pour la prise en charge des cas IS NULL et IS NOT NULL .

Concepts associés

«Comportement de la sélection», à la page 36
Présentation du comportement de la sélection IBM MQ .

Référence associée

Chaînes de sélection

Remarques relatives à UTF-8 et à Unicode lors de l'utilisation de sélecteurs de message

Les caractères, qui ne sont pas placés entre guillemets simples, qui constituent les mots clés réservés d'une chaîne de sélection doivent être entrés en Unicode latin de base (du caractère U+0000 à U+0007F). Il n'est pas possible d'utiliser d'autres représentations de point de code de caractères alphanumériques. Par exemple, le nombre 1 doit être exprimé sous la forme U+0031 en Unicode, il n'est pas valide d'utiliser l'équivalent en chiffres pleine largeur U+FF11 ou l'équivalent en arabe U+0661.

Les noms de propriété de message peuvent être spécifiés à l'aide de n'importe quelle séquence valide de caractères Unicode. Les noms de propriété de message contenus dans les chaînes de sélection codées en UTF-8 seront validés même s'ils contiennent des caractères multi-octets. La validation de UTF-8 multi-octet est stricte et vous devez vous assurer que des séquences UTF-8 valides sont utilisées pour les noms de propriété de message. Les caractères au-delà du plan multilingue de base Unicode (ceux au-dessus de U + FFFF), représentés en UTF-16 par des points de code de substitution (X'D800'à X'DFFF'), ou quatre octets en UTF-8, ne sont pas pris en charge dans les noms de propriété de message.

Aucun traitement supplémentaire n'est effectué sur les noms ou les valeurs de propriété lors de la comparaison pour l'égalité. Cela signifie par example qu'il n'y a pas de pré / décomposition et que les ligatures n'ont pas de signification particulière. Par exemple, le caractère de tréma précomposé U+00FC n'est pas considéré comme équivalent à U+0075 + U+0308 et la séquence de caractères ff n'est pas considérée comme équivalente à Unicode U+FB00 (LATIN SMALL LIGATURE FF)

Les données de propriété placées entre apostrophes peuvent être représentées par n'importe quelle séquence d'octets et ne sont pas validées.

Sélection du contenu d'un message

Il est possible de s'abonner en fonction d'une sélection de contenu de message (également appelé filtrage de contenu), mais la décision concernant les messages à distribuer à un abonnement de ce type ne peut pas être effectuée directement par IBM MQ; à la place, un fournisseur de sélection de message étendu, par exemple IBM Integration Bus, est requis pour traiter les messages.

Lorsqu'une application publie sur une chaîne de rubrique, où un ou plusieurs abonnés ont une chaîne de sélection sélectionnée sur le contenu du message, IBM MQ demande que le fournisseur de sélection de message étendu analyse la publication et indique à IBM MQ si la publication correspond aux critères de sélection spécifiés par chaque abonné avec un filtre de contenu.

Si le fournisseur de sélection de message étendu détermine que la publication correspond à la chaîne de sélection de l'abonné, le message continue d'être distribué à l'abonné.

Si le fournisseur de sélection de message étendu détermine que la publication ne correspond pas, le message n'est pas distribué à l'abonné. Cela peut entraîner l'échec de l'appel MQPUT ou MQPUT1 avec le code anomalie MQRC_PUBLICATION_FAILURE. Si le fournisseur de sélection de message étendu ne parvient pas à analyser la publication, le code anomalie MQRC_CONTENT_ERROR est renvoyé et l'appel MQPUT ou MQPUT1 échoue.

Si le fournisseur de sélection de message étendu n'est pas disponible ou ne parvient pas à déterminer si l'abonné doit recevoir la publication, le code anomalie MQRC_SELECTION_NOT_AVAILABLE est renvoyé et l'appel MQPUT ou MQPUT1 échoue.

Lorsqu'un abonnement est créé avec un filtre de contenu et que le fournisseur de sélection de message étendu n'est pas disponible, l'appel MQSUB échoue avec le code anomalie MQRC_SELECTION_NOT_AVAILABLE. Si un abonnement avec un filtre de contenu est repris et que le fournisseur de sélection de message étendu n'est pas disponible, l'appel MQSUB renvoie un avertissement de MQRC_SELECTION_NOT_AVAILABLE, mais l'abonnement peut être repris.

Référence associée

Chaînes de sélection

Consommation asynchrone de messages IBM MQ

La consommation asynchrone utilise un ensemble d'extensions MQI (Message Queue Interface), MQI appelle MQCB et MQCTL, qui permettent à une application MQI d'être écrite pour consommer des messages à partir d'un ensemble de files d'attente. Les messages sont distribués à l'application en appelant une "unité de code", identifiée par l'application qui transmet le message ou un jeton représentant le message.

Dans les environnements d'application les plus simples, l'unité de code est définie par un pointeur de fonction, mais dans d'autres environnements, l'unité de code peut être définie par un nom de programme ou de module.

Dans la consommation asynchrone des messages, les termes suivants sont utilisés:

Consommateur de message
: Construction de programmation qui permet de définir un programme, ou une fonction, à appeler avec un message lorsqu'un message correspondant aux exigences des applications devient disponible.

Gestionnaire d'événements
: Construction de programmation qui permet de définir un programme ou une fonction à appeler lorsqu'un événement asynchrone, tel qu'une mise au repos du gestionnaire de files d'attente, se produit.

Rappel
: Terme générique utilisé pour désigner une routine de consommateur de message ou de gestionnaire d'événements.

La consommation asynchrone peut simplifier la conception et l'implémentation de nouvelles applications, en particulier celles qui traitent plusieurs files d'attente d'entrée ou abonnements. Toutefois, si vous utilisez plusieurs files d'attente d'entrée et que vous traitez les messages par ordre de priorité, la séquence de priorité est observée indépendamment dans chaque file d'attente: vous pouvez obtenir des messages de priorité basse d'une file d'attente avant des messages de priorité élevée d'une autre. L'ordre des messages dans plusieurs files d'attente n'est pas garanti. Notez également que si vous utilisez des exits API, vous devrez peut-être les modifier pour inclure les appels MQCB et MQCTL.

Les illustrations suivantes donnent un exemple de la façon dont vous pouvez utiliser cette fonction.

La Figure 5, à la page 45 illustre une application à unités d'exécution multiples qui consomme des messages provenant de deux files d'attente. L'exemple montre tous les messages distribués à une seule fonction.

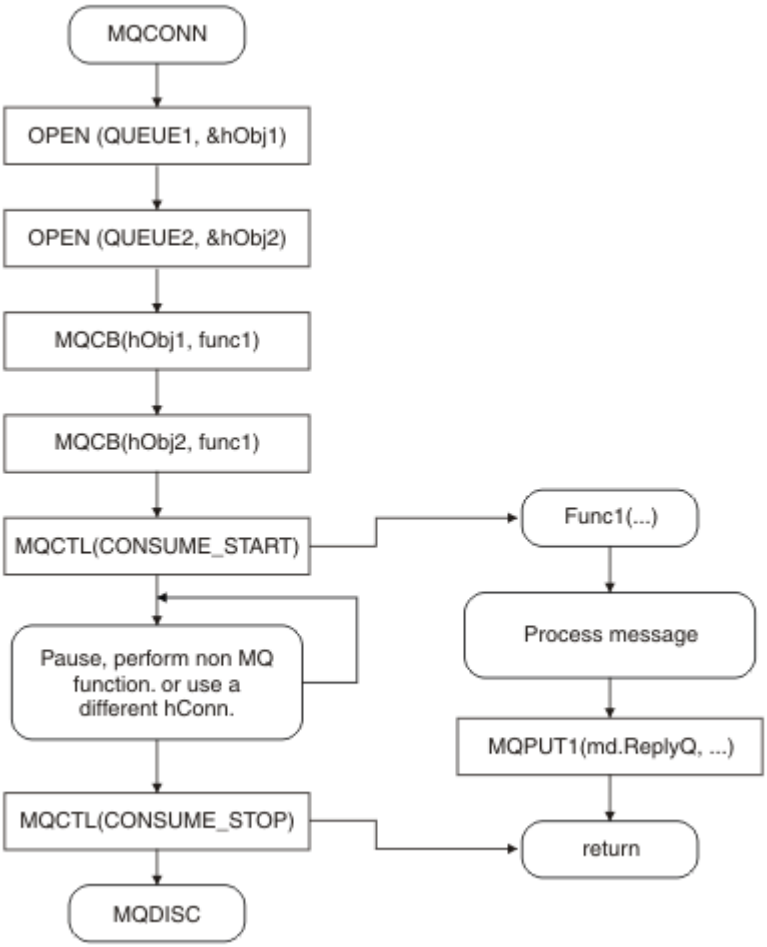


Figure 5. Application gérée par message standard consommant à partir de deux files d'attente

z/OS Sous z/OS, l'unité d'exécution de contrôle principale doit émettre un appel MQDISC avant de s'arrêter. Cela permet à toutes les unités d'exécution de rappel d'arrêter et de libérer des ressources système.

Figure 6, à la page 46 Cet exemple de flux montre une application à unité d'exécution unique consommant des messages provenant de deux files d'attente. L'exemple montre tous les messages distribués à une seule fonction.

La différence par rapport au cas asynchrone est que le contrôle ne revient pas à l'émetteur de MQCTL tant que tous les consommateurs ne se sont pas désactivés ; c'est-à-dire qu'un consommateur a émis une demande MQCTL STOP ou que le gestionnaire de files d'attente est mis au repos.

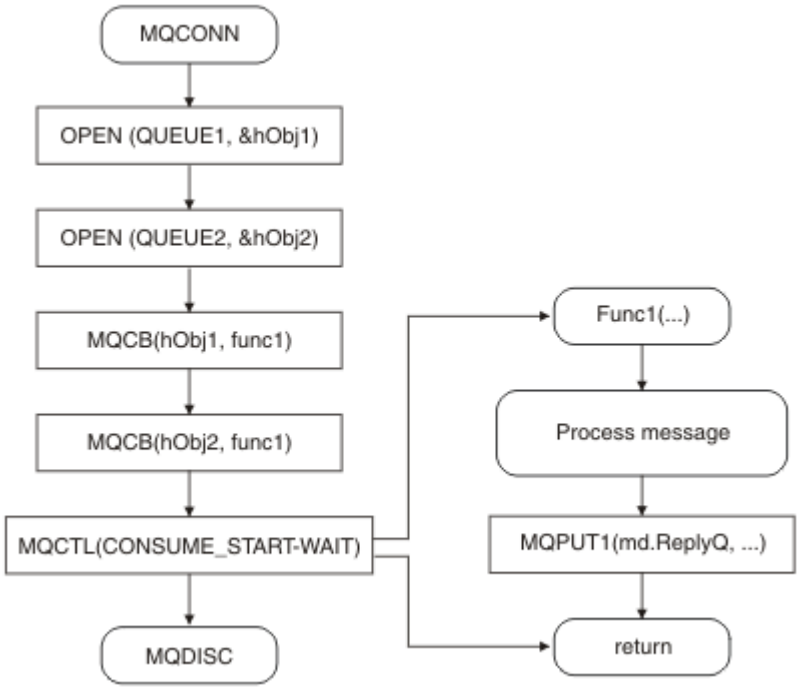


Figure 6. Application gérée par message à unité d'exécution unique consommant à partir de deux files d'attente

Groupes de messages

Les messages peuvent se produire au sein des groupes pour permettre l'ordre des messages.

Les groupes de messages permettent de marquer plusieurs messages comme étant liés les uns aux autres et d'appliquer un ordre logique au groupe (voir «Ordre logique et physique», à la page 793). Sur Multiplateformes, la segmentation de message permet de fractionnés les messages de grande taille en segments plus petits. Vous ne pouvez pas utiliser de messages groupés ou segmentés lors de l'insertion dans une rubrique.

La hiérarchie au sein d'un groupe est la suivante:

Groupe
Il s'agit du niveau le plus élevé de la hiérarchie et il est identifié par un *GroupId*. Il se compose d'un ou de plusieurs messages qui contiennent le même *GroupId*. Ces messages peuvent être stockés n'importe où dans la file d'attente.

Remarque : Le terme *message* est utilisé ici pour désigner un élément d'une file d'attente, tel qu'il serait renvoyé par une seule instruction MQGET qui ne spécifie pas MQGMO_COMPLETE_MSG.

La Figure 7, à la page 46 présente un groupe de messages logiques:

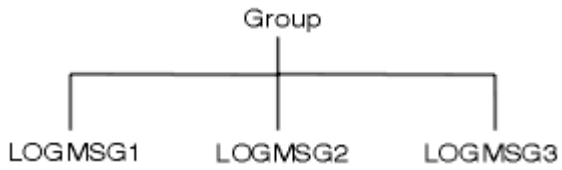


Figure 7. Groupe de messages logiques

En ouvrant une file d'attente et en spécifiant MQOO_BIND_ON_GROUP, vous forcez tous les messages d'un groupe envoyés à cette file d'attente à être envoyés à la même instance de la file d'attente. Pour plus d'informations sur l'option BIND_ON_GROUP, voir Gestion des affinités de message.

Message logique

Les messages logiques d'un groupe sont identifiés par les zones *GroupId* et *MsgSeqNumber* . Le *MsgSeqNumber* commence à 1 pour le premier message d'un groupe et, si un message n'est pas dans un groupe, la valeur de la zone est 1.

Utilisez les messages logiques d'un groupe pour:

- Assurer l'ordre (si cela n'est pas garanti dans les circonstances dans lesquelles le message est transmis).
- Permet aux applications de regrouper des messages similaires (par exemple, ceux qui doivent tous être traités par la même instance de serveur).

Chaque message d'un groupe se compose d'un message physique, sauf s'il est divisé en segments. Chaque message est logiquement un message distinct, et seules les zones *GroupId* et *MsgSeqNumber* du MQMD doivent comporter une relation avec les autres messages du groupe. Les autres zones du MQMD sont indépendantes ; certaines peuvent être identiques pour tous les messages du groupe alors que d'autres peuvent être différentes. Par exemple, les messages d'un groupe peuvent avoir des noms de format, des CCSID et des codages différents.

Segment

Les segments sont utilisés pour traiter les messages qui sont trop volumineux pour l'application d'insertion ou d'extraction ou le gestionnaire de files d'attente (y compris les gestionnaires de files d'attente intermédiaires via lesquels le message est transmis). Pour plus d'informations, voir «Segmentation des messages», à la page 812.

Un message individuel est divisé en messages plus petits appelés *segments*. Un segment d'un message est identifié par les zones *GroupId*, *MsgSeqNumber*et *Offset* . La zone *Offset* commence à zéro pour le premier segment d'un message.

Chaque segment se compose d'un message physique qui peut appartenir à un groupe (Figure 8, à la page 47 montre un exemple de messages au sein d'un groupe). Un segment fait logiquement partie d'un seul message, de sorte que seules les zones *MsgId*, *Offset*et *MsgFlags* du MQMD doivent différer entre les segments distincts du même message. Si un segment n'arrive pas, le code anomalie MQRC_INCOMPLETE_GROUP ou MQRC_INCOMPLETE_MSG est renvoyé comme il convient.

La Figure 8, à la page 47 montre un groupe de messages logiques, dont certains sont segmentés:

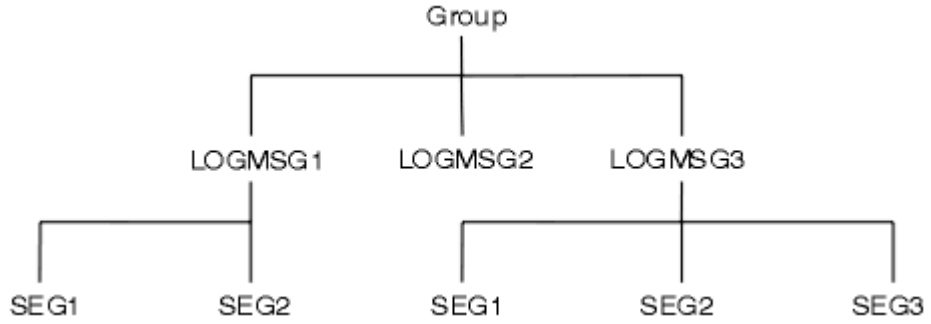


Figure 8. Messages segmentés

z/OS La segmentation n'est pas prise en charge sous IBM MQ for z/OS.

Vous ne pouvez pas utiliser de messages segmentés ou groupés avec la fonction de publication / abonnement.

Concepts associés

«Segmentation des messages», à la page 812
Utilisez ces informations pour en savoir plus sur la segmentation des messages. Cette fonction n'est pas prise en charge sous IBM MQ for z/OS ou par les applications utilisant IBM MQ classes for JMS.

Référence associée

«Ordre logique et physique», à la page 793

Dans chaque niveau de priorité, les messages des files d'attente peuvent apparaître dans l'ordre *physique* ou *logique* .

MQMD-Descripteur de message

Persistance des messages

Les messages persistants sont écrits dans les journaux et les fichiers de données de file d'attente. Si un gestionnaire de files d'attente est redémarré après un incident, il récupère ces messages persistants à partir des données consignées. Les messages qui ne sont pas persistants sont supprimés si un gestionnaire de files d'attente s'arrête, que l'arrêt soit dû à une commande de l'opérateur ou à l'échec d'une partie de votre système.

z/OS Les messages non persistants stockés dans une unité de couplage (CF) sous z/OS constituent une exception à cette règle. Ils persistent aussi longtemps que les FC restent disponibles.

Lorsque vous créez un message, si vous initialisez le descripteur de message (MQMD) à l'aide des valeurs par défaut, la persistance du message est extraite de l'attribut **DefPersistence** de la file d'attente indiquée dans la commande MQOPEN. Vous pouvez également définir la persistance du message à l'aide de la zone *Persistence* de la structure MQMD pour définir le message comme persistant ou non persistant.

Les performances de votre application sont affectées lorsque vous utilisez des messages persistants ; l'étendue de l'effet dépend des caractéristiques de performances du sous-système d'E-S de la machine et de la manière dont vous utilisez les options de point de synchronisation sur chaque plateforme:

- Un message persistant, en dehors de l'unité de travail en cours, est écrit sur le disque à chaque opération d'insertion et d'extraction. Voir «Validation et annulation d'unités de travail», à la page 876.
- z/OS ALW Pour toutes les plateformes à l'exception de IBM i, un message persistant dans l'unité de travail en cours est consigné uniquement lorsque l'unité de travail est validée et que l'unité de travail peut contenir de nombreuses opérations de file d'attente.

Les messages non persistants peuvent être utilisés pour la messagerie rapide. Pour plus d'informations sur les messages rapides, voir Sécurité des messages .

Remarque : Une combinaison de l'écriture de messages persistants dans une unité de travail et de l'écriture de messages persistants en dehors d'une unité ou d'un travail peut entraîner des problèmes de performances potentiellement graves pour vos applications. Cela est particulièrement vrai lorsque la même file d'attente cible est utilisée pour les deux opérations.

Messages dont la distribution a échoué

Lorsqu'un gestionnaire de files d'attente ne peut pas placer un message dans une file d'attente, vous disposez de différentes options.

Vous pouvez :

- Essayez à nouveau d'insérer le message dans la file d'attente.
- Demande de renvoi du message à l'expéditeur.
- Placez le message dans la file d'attente des messages non livrés.

Pour plus d'informations, voir«Traitement des erreurs de programme de procédure», à la page 1065.

Messages annulés

Lors du traitement de messages à partir d'une file d'attente sous le contrôle d'une unité de travail, l'unité de travail peut être constituée d'un ou de plusieurs messages. Si une annulation se produit, les messages qui ont été extraits de la file d'attente sont réintégrés dans la file d'attente et peuvent être traités à nouveau dans une autre unité d'oeuvre. Si le traitement d'un message particulier est à l'origine de l'incident, l'unité d'oeuvre est à nouveau annulée. Cela peut entraîner une boucle de traitement. Les messages qui ont été insérés dans une file d'attente sont supprimés de la file d'attente.

Une application peut détecter les messages qui sont interceptés dans une telle boucle en testant la zone *BackoutCount* de MQMD. L'application peut soit corriger la situation, soit envoyer un avertissement à un opérateur.

Multi Le nombre d'annulations survit toujours aux redémarrages du gestionnaire de files d'attente. Toute modification apportée à l'attribut **HardenGetBackout** est ignorée.

z/OS Pour les files d'attente partagées, le nombre d'annulations survit toujours aux redémarrages du gestionnaire de files d'attente. Pour toutes les autres configurations sous z/OS, pour vous assurer que le nombre d'annulations des files d'attente privées survit aux redémarrages du gestionnaire de files d'attente, définissez l'attribut *HardenGetBackout* sur MQQA_BACKOUT_HARDENED; sinon, si le gestionnaire de files d'attente doit redémarrer, il ne conserve pas un nombre d'annulations précis pour chaque message. La définition de l'attribut de cette manière ajoute le coût du traitement supplémentaire.

Pour plus d'informations sur la validation et l'annulation des messages, voir «Validation et annulation d'unités de travail», à la page 876.

File d'attente de réponse et gestionnaire de files d'attente

Dans certains cas, vous pouvez recevoir des messages en réponse à un message que vous envoyez:

- Un message de réponse en réponse à un message de demande
- Un message de rapport sur un événement inattendu ou une expiration
- Un message de rapport sur un événement COA (Confirmation d'arrivée) ou COD (Confirmation de livraison)
- Un message de rapport sur un événement PAN (Positive Action Notification) ou NAN (Negative Action Notification)

A l'aide de la structure MQMD, indiquez le nom de la file d'attente à laquelle vous souhaitez envoyer les messages de réponse et de rapport, dans la zone *ReplyToQ* . Indiquez le nom du gestionnaire de files d'attente propriétaire de la file d'attente de réponse dans la zone *ReplyToQMgr* .

Si vous laissez la zone *ReplyToQMgr* vide, le gestionnaire de files d'attente définit le contenu des zones suivantes dans le descripteur de message de la file d'attente:

ReplyToQ
Si *ReplyToQ* est une définition locale d'une file d'attente éloignée, la zone *ReplyToQ* est définie sur le nom de la file d'attente éloignée ; sinon, cette zone n'est pas modifiée.

ReplyToQMgr
Si *ReplyToQ* est une définition locale d'une file d'attente éloignée, la zone *ReplyToQMgr* est définie sur le nom du gestionnaire de files d'attente propriétaire de la file d'attente éloignée ; sinon, la zone *ReplyToQMgr* est définie sur le nom du gestionnaire de files d'attente auquel votre application est connectée.

Remarque : Vous pouvez demander qu'un gestionnaire de files d'attente effectue plusieurs tentatives de distribution d'un message et vous pouvez demander que le message soit supprimé en cas d'échec. Si le message, après l'échec de sa distribution, ne doit pas être supprimé, le gestionnaire de files d'attente éloignées le place dans sa file d'attente de rebut (voir «Utilisation de la file d'attente de rebut (messages non livrés)», à la page 1068).

Contexte de message

Les informations de *contexte de message* permettent à l'application qui extrait le message de trouver l'émetteur du message.

L'application d'extraction peut souhaiter:

- Vérifiez que l'application émettrice dispose du niveau de droits correct
- Effectuer une fonction de comptabilité afin qu'elle puisse facturer l'application émettrice pour tout travail qu'elle doit effectuer

- Conserver une trace d'audit de tous les messages qu'elle a traités

Lorsque vous utilisez l'appel MQPUT ou MQPUT1 pour placer un message dans une file d'attente, vous pouvez indiquer que le gestionnaire de files d'attente doit ajouter des informations de contexte par défaut au descripteur de message. Les applications disposant du niveau de droits approprié peuvent ajouter des informations de contexte supplémentaires. Pour plus d'informations sur la spécification des informations de contexte, voir «Contrôle des informations de contexte de message», à la page 779.

Le contexte utilisateur est utilisé par le gestionnaire de files d'attente lors de la génération des types de message de rapport suivants:

- Confirmer à expédition
- Expiration

Lorsque ces messages de rapport sont générés, le contexte utilisateur est vérifié pour les droits + put et + passid sur la destination du rapport. Lorsque le contexte utilisateur ne dispose pas des droits suffisants, le message de rapport est placé dans la file d'attente des messages non livrés si celle-ci a été définie. Lorsqu'il n'y a pas de file d'attente de rebut, le message de rapport est supprimé.

Toutes les informations de contexte sont stockées dans les zones de contexte du descripteur de message. Le type d'informations correspond aux informations d'identité, d'origine et de contexte utilisateur.

contexte d'identité

Les informations de *contexte d'identité* identifient l'utilisateur de l'application qui a d'abord inséré le message dans une file d'attente. Les applications dûment autorisées peuvent définir les zones suivantes:

- Le gestionnaire de files d'attente remplit la zone *UserIdentifier* avec un nom qui identifie l'utilisateur ; la manière dont le gestionnaire de files d'attente peut effectuer cette opération dépend de l'environnement dans lequel l'application s'exécute.
- Le gestionnaire de files d'attente remplit la zone *AccountingToken* avec un jeton ou un numéro déterminé à partir de l'application qui a inséré le message.
- Les applications peuvent utiliser la zone *ApplIdentityData* pour toute information supplémentaire qu'elles souhaitent inclure sur l'utilisateur (par exemple, un mot de passe chiffré).

Un identificateur de sécurité des systèmes Windows (SID) est stocké dans la zone *AccountingToken* lorsqu'un message est créé sous IBM MQ for Windows. Le SID peut être utilisé pour compléter la zone *UserIdentifier* et pour établir les données d'identification d'un utilisateur.

Pour plus d'informations sur la façon dont le gestionnaire de files d'attente remplit les zones *UserIdentifier* et *AccountingToken* , voir les descriptions de ces zones dans UserIdentifier et AccountingToken.

Les applications qui transmettent des messages d'un gestionnaire de files d'attente à un autre doivent également transmettre les informations de contexte d'identité afin que les autres applications connaissent l'identité de l'émetteur du message.

Contexte d'origine

Les informations de *contexte d'origine* décrivent l'application qui a inséré le message dans la file d'attente dans laquelle le message est actuellement stocké. Le descripteur de message contient les zones suivantes pour les informations de contexte d'origine:

- *PutApplType* définit le type d'application qui a inséré le message (par exemple, une transaction CICS).
- *PutApplName* définit le nom de l'application qui a inséré le message (par exemple, le nom d'un travail ou d'une transaction).
- *PutDate* définit la date à laquelle le message a été inséré dans la file d'attente.
- *PutTime* définit l'heure à laquelle le message a été inséré dans la file d'attente.

- *ApplOriginData* définit les informations supplémentaires qu'une application souhaite inclure sur l'origine du message. Par exemple, il peut être défini par des applications dûment autorisées pour indiquer si les données d'identité sont dignes de confiance.

Les informations de contexte d'origine sont généralement fournies par le gestionnaire de files d'attente. Le temps moyen de Greenwich (GMT) est utilisé pour les zones *PutDate* et *PutTime* . Consultez les descriptions de ces zones dans PutDate et PutTime.

Une application disposant de droits suffisants peut fournir son propre contexte. Cela permet de conserver les informations comptables lorsqu'un seul utilisateur possède un ID utilisateur différent sur chacun des systèmes qui traitent un message qu'ils ont émis.

Objets IBM MQ

Ces informations fournissent des détails sur les objets IBM MQ , notamment les gestionnaires de files d'attente, les groupes de partage de files d'attente, les files d'attente, les objets de rubrique d'administration, les listes de noms, les définitions de processus, les objets d'informations d'authentification, les canaux, les classes de stockage, les programmes d'écoute et les services.

Les gestionnaires de files d'attente définissent les propriétés (appelées attributs) de ces objets. Les valeurs de ces attributs affectent la manière dont IBM MQ traite ces objets. A partir de vos applications, vous utilisez l'interface MQI (Message Queue Interface) pour contrôler ces objets. Les objets sont identifiés par un *descripteur d'objet* (MQOD) lorsqu'ils sont traités à partir d'un programme.

Lorsque vous utilisez des commandes IBM MQ pour définir, modifier ou supprimer des objets, par exemple, le gestionnaire de files d'attente vérifie que vous disposez du niveau de droits requis pour effectuer ces opérations. De même, lorsqu'une application utilise l'appel MQOPEN pour ouvrir un objet, le gestionnaire de files d'attente vérifie que l'application dispose du niveau de droits d'accès requis avant qu'elle soit autorisée à accéder à cet objet. Les vérifications s'effectuent au niveau du nom de l'objet à ouvrir.

Concepts associés

«Contrôle des informations de contexte de message», à la page 779
Lorsque vous utilisez l'appel MQPUT ou MQPUT1 pour placer un message dans une file d'attente, vous pouvez indiquer que le gestionnaire de files d'attente doit ajouter des informations de contexte par défaut au descripteur de message. Les applications disposant du niveau de droits approprié peuvent ajouter des informations de contexte supplémentaires. Vous pouvez utiliser la zone d'options de la structure MQPMO pour contrôler les informations de contexte.

Référence associée

«Options MQOPEN relatives au contexte de message», à la page 769
Si vous souhaitez pouvoir associer des informations de contexte à un message lorsque vous le placez dans une file d'attente, vous devez utiliser l'une des options de contexte de message lorsque vous ouvrez la file d'attente.

Windows Préparation et exécution des applications Microsoft Transaction Server

Pour préparer une application MTS à s'exécuter en tant qu'application IBM MQ MQI client , suivez ces instructions en fonction de votre environnement.

Pour des informations générales sur le développement d'applications Microsoft Transaction Server (MTS) qui accèdent aux ressources IBM MQ , voir la section relative à MTS dans le centre d'aide IBM MQ .

Pour préparer une application MTS à s'exécuter en tant qu'application IBM MQ MQI client , effectuez l'une des opérations suivantes pour chaque composant de l'application:

- Si le composant utilise les liaisons de langage C pour l'interface MQI, suivez les instructions de la rubrique «Préparation des programmes C dans Windows», à la page 1044 mais liez le composant à la bibliothèque mqicxa.lib à la place de mqic.lib.

- Si le composant utilise les classes C++ IBM MQ , suivez les instructions de la rubrique «Génération de programmes C++ sur Windows», à la page 563 mais liez le composant à la bibliothèque imqx23vn.lib à la place de imqc23vn.lib.
- Si le composant utilise les liaisons de langage Visual Basic pour l'interface MQI, suivez les instructions de la rubrique «Préparation des programmes Visual Basic dans Windows», à la page 1047 , mais lorsque vous définissez le projet Visual Basic, entrez MqType=3 dans la zone **Arguments de compilation conditionnels** .

Remarques sur la conception des applications IBM MQ

Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

Lors de la conception d'une application IBM MQ , tenez compte des questions et des options suivantes:

Type d'application

Quel est le but de votre demande? Pour plus d'informations sur les différents types d'application que vous pouvez développer, voir les liens suivants:

- serveur
- Environnement
- Publication/abonnement
- Services Web
- Exits utilisateur, exits API et services installables

En outre, vous pouvez également écrire vos propres applications pour automatiser l'administration d' IBM MQ. Pour plus d'informations, voir IBM MQ Administration Interface (MQAI) et Automatisation des tâches d'administration.

Langage de programmation

IBM MQ prend en charge un certain nombre de langages de programmation différents pour l'écriture d'applications. Pour plus d'informations, voir «Développement d'applications pour IBM MQ», à la page 5.

Applications pour plusieurs plateformes

Votre application s'exécutera-t-elle sur plusieurs plateformes? Avez-vous une stratégie pour passer à une plateforme différente de celle que vous utilisez aujourd'hui? Si la réponse à l'une de ces questions est oui, veillez à coder vos programmes pour l'indépendance de la plateforme.

Par exemple, si vous utilisez C, code dans la norme ANSI C. Utilisez une fonction de bibliothèque C standard plutôt qu'une fonction équivalente spécifique à la plateforme, même si la fonction spécifique à la plateforme est plus rapide ou plus efficace. L'exception est lorsque l'efficacité dans le code est primordiale, lorsque vous devez coder pour les deux situations à l'aide de #ifdef. Exemple :

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

Types de files d'attente

Voulez-vous créer une file d'attente chaque fois que vous en avez besoin, ou souhaitez-vous utiliser des files d'attente qui ont déjà été configurées? Voulez-vous supprimer une file d'attente une fois que vous avez fini de l'utiliser ou va-t-elle être réutilisée? Voulez-vous utiliser les files d'attente alias pour l'indépendance de l'application? Pour connaître les types de files d'attente pris en charge, voir Files d'attente.

z/OS Utilisation de files d'attente partagées, de groupes de partage de files d'attente et de clusters de groupes de partage de files d'attente (IBM MQ for z/OS uniquement)

Vous pouvez tirer parti de l'augmentation de la disponibilité, de l'évolutivité et de l'équilibrage de la charge de travail qui sont possibles lorsque vous utilisez des files d'attente partagées avec des groupes de partage de files d'attente. Pour plus d'informations, voir Files d'attente partagées et groupes de partage de files d'attente .

Vous pouvez également estimer les flux de messages moyens et de pointe et envisager d'utiliser des clusters de groupes de partage de files d'attente pour répartir la charge de travail. Pour plus d'informations, voir Files d'attente partagées et groupes de partage de files d'attente .

Utilisation des clusters de gestionnaires de files d'attente

Vous souhaiterez peut-être tirer parti de l'administration système simplifiée et de l'augmentation de la disponibilité, de l'évolutivité et de l'équilibrage de la charge de travail qui sont possibles lorsque vous utilisez des clusters.

Types de messages

Vous pouvez utiliser des datagrammes pour des messages simples, mais des messages de demande (pour lesquels vous attendez des réponses) pour d'autres situations. Vous pouvez affecter des priorités différentes à certains de vos messages. Pour plus d'informations sur la conception des messages, voir «Techniques de conception pour les messages», à la page 61.

Utilisation de la messagerie de publication / abonnement ou point à point

A l'aide de la messagerie de publication / abonnement, une application d'envoi envoie les informations qu'elle souhaite partager dans un message IBM MQ à une destination standard gérée par IBM MQ publish? subscribe, et permet à IBM MQ de gérer la distribution de ces informations. L'application cible n'a pas besoin de connaître la source des informations qu'elle reçoit, elle enregistre simplement un intérêt dans un ou plusieurs sujets et reçoit ces informations lorsqu'elles sont disponibles. Pour plus d'informations sur la messagerie de publication/abonnement, voir Publication/ abonnement.

A l'aide de la messagerie point-à-point, une application émettrice envoie un message à une file d'attente spécifique, à partir de laquelle elle sait qu'une application réceptrice l'extraira. Une application réceptrice obtient des messages d'une file d'attente spécifique et agit sur leur contenu. Une application fonctionne souvent à la fois comme un expéditeur et un récepteur, envoyant une requête à une autre application et recevant une réponse.

Contrôle de vos programmes IBM MQ

Vous pouvez démarrer certains programmes automatiquement ou faire en sorte que les programmes attendent qu'un message particulier arrive dans une file d'attente (à l'aide de la fonction de IBM MQ *déclenchement* , voir «Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888). Vous pouvez également démarrer une autre instance d'une application lorsque les messages d'une file d'attente ne sont pas traités assez rapidement (à l'aide de la fonction IBM MQ Evénements d'instrumentation , comme décrit dans Evénements d'instrumentation).

Exécution de votre application sur un client IBM MQ

L'interface MQI complète est prise en charge dans l'environnement client et presque toutes les applications IBM MQ écrites dans un langage procédural peuvent être réliées pour s'exécuter sur un IBM MQ MQI client. Liez l'application sur le IBM MQ MQI client à la bibliothèque MQIC plutôt qu'à la bibliothèque MQI. z/OS Get (signal) on z/OS n'est pas pris en charge.

Remarque : Une application exécutée sur un client IBM MQ peut se connecter à plusieurs gestionnaires de files d'attente simultanément ou utiliser un nom de gestionnaire de files d'attente avec un astérisque (*) sur un appel MQCONN ou MQCONNX. Modifiez l'application si vous souhaitez établir un lien vers les bibliothèques du gestionnaire de files d'attente à la place des bibliothèques client, car cette fonction ne sera pas disponible.

Pour plus d'informations, voir «Exécution d'applications dans l'environnement IBM MQ MQI client», à la page 946.

Performance des applications
Les décisions de conception peuvent avoir un impact sur les performances de vos applications. Pour des suggestions d'amélioration des performances des applications IBM MQ , voir «Considérations relatives à la conception et aux performances des applications», à la page 63 IBM i et «Remarques sur la conception et les performances des applications IBM i», à la page 66 .

Techniques IBM MQ avancées
Pour les applications plus avancées, vous pouvez utiliser des techniques IBM MQ avancées telles que la corrélation des réponses et la génération et l'envoi d'informations de contexte IBM MQ . Pour plus d'informations, voir «Techniques de conception pour les applications avancées», à la page 65.

Sécurisation de vos données et maintien de leur intégrité
Vous pouvez utiliser les informations de contexte transmises avec un message pour vérifier que le message a été envoyé à partir d'une source acceptable. Vous pouvez utiliser les fonctions de point de synchronisation fournies par IBM MQ ou votre système d'exploitation pour vous assurer que vos données restent cohérentes avec d'autres ressources (voir «Validation et annulation d'unités de travail», à la page 876 pour plus de détails). Vous pouvez utiliser la fonction de *persistance* des messages IBM MQ pour assurer la distribution des messages importants.

Test des applications IBM MQ

L'environnement de développement d'application pour les programmes IBM MQ n'étant pas différent de celui d'une autre application, vous pouvez utiliser les mêmes outils de développement et les mêmes fonctions de trace IBM MQ .

z/OS Lors du test d'applications CICS avec IBM MQ for z/OS, vous pouvez utiliser la fonction CEDF (CICS Execution Diagnostic Facility). CEDF intercepte l'entrée et la sortie de chaque appel MQI ainsi que les appels à tous les services CICS . En outre, dans l'environnement CICS , vous pouvez écrire un programme d'exit de croisement d'API pour fournir des informations de diagnostic avant et après chaque appel MQI. Pour savoir comment procéder, voir «Using and writing applications on IBM MQ for z/OS», à la page 914.

IBM i Lorsque vous testez des applications IBM i , vous pouvez utiliser le débogueur standard. Pour ce faire, utilisez la commande STRDBG.

Traitement des exceptions et des erreurs
Vous devez savoir comment traiter les messages qui ne peuvent pas être distribués et comment résoudre les situations d'erreur qui vous sont signalées par le gestionnaire de files d'attente. Pour certains rapports, vous devez définir des options de rapport sur MQPUT.

Concepts associés
Présentation technique du IBM MQ
«Design and performance considerations for z/OS applications», à la page 68
Application design is one of the most important factors affecting performance. Use this topic to understand some of the design factors involved in performance.

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Ecriture d'une application de procédure pour la mise en file d'attente», à la page 738
La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

«Ecriture d'applications de procédure client», à la page 937
Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

«Développement d'applications C++», à la page 539
IBM MQ fournit des classes C++ équivalentes à des objets IBM MQ et des classes supplémentaires équivalentes aux types de données de tableau. Il fournit un certain nombre de fonctions qui ne sont pas disponibles via l'interface MQI.

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85
IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

«Utilisation IBM MQ classes for Java», à la page 356
Utilisez IBM MQ dans un environnement Java . IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

Tâches associées

«Développement d'applications .NET», à la page 567
Les applications IBM MQ classes for .NET permettent aux applications .NET de se connecter à IBM MQ en tant que IBM MQ MQI client ou de se connecter directement à un serveur IBM MQ .

Spécification du nom d'application dans les langages de programmation pris en charge

Avant IBM MQ 9.2.0, vous pouviez déjà spécifier un nom d'application sur les applications client Java ou JMS . Depuis IBM MQ 9.2.0 , cette fonction est étendue à d'autres langages de programmation sous IBM MQ for Multiplatforms.

Utilisation du nom d'application

Le nom de l'application est généré à partir de:

- runmqsc AFFICHE APPLTAG CONN
- runmqsc AFFICHAGE DU TYPE QSTATUS (DESCRIPTEUR) APPLTAG
- runmqsc AFFICHAGE DE LA BALISE RAPPLTAG CHSTATUS
- MQMD.PutApplName
- Trace de l'activité d'application

Le nom de l'application est également utilisé lors de la configuration de la trace de l'activité de l'application. Le nom d'application par défaut des applications nonJava est le nom tronqué de l'exécutable, sauf sous Windows et IBM i.

Windows Sous Windows, le nom par défaut est le nom complet de l'exécutable, tronqué à 28 caractères sur la gauche.

IBM i Sous IBM i, le nom par défaut est le nom du travail.

Pour les applications Java , il s'agit du nom de classe précédé du nom de package tronqué à gauche à 28 caractères.

Pour plus d'informations, voir PutApplName.

Les applications sur IBM MQ for Multiplatforms peuvent définir leurs noms d'application de manière administrative ou à l'aide de diverses méthodes de programmation. Cela permet aux applications de fournir un nom plus significatif indépendant de la plateforme, lorsque vous configurez la trace de l'activité d'application ou lorsque la sortie est générée à partir de diverses commandes **runmqsc** .

Vous pouvez rééquilibrer les applications dans un cluster uniforme. Des noms d'application significatifs sont utilisés pour y parvenir.

Caractères pris en charge

Pour plus d'informations sur la manière dont vous spécifiez le nom de l'application, voir «Caractères de nom d'application recommandés», à la page 56 .

Langages de programmation

Pour plus d'informations sur la façon dont les applications qui se résolvent dans les bibliothèques IBM MQ en C et dans d'autres langages de programmation peuvent fournir le nom de l'application, voir «Connexions de langage de programmation», à la page 58 .

Applications .NET gérées

Pour plus d'informations sur la façon dont les applications .NET gérées peuvent fournir le nom de l'application, voir «Applications .NET gérées», à la page 59 .

Applications XMS

Pour plus d'informations sur la façon dont les applications XMS peuvent fournir le nom de l'application, voir «Applications XMS», à la page 60 .

Applications de liaisons Java et JMS

ALW

Pour plus d'informations sur la manière dont les applications Java et JMS peuvent fournir le nom de l'application, voir «Applications de liaisons Java et JMS», à la page 60 .

Concepts associés

Trace de l'activité d'application

A propos des clusters uniformes

Référence associée

MQCNO

MQCNO sur IBM i

Utilisation du nom d'application dans les langages de programmation pris en charge

Utilisez ces informations pour savoir comment le nom de l'application est sélectionné dans les différentes langues prises en charge par IBM MQ .

Caractères de nom d'application recommandés

Les noms d'application doivent figurer dans le jeu de caractères indiqué par l'attribut **CodedCharSetId** de la zone du gestionnaire de files d'attente. Pour plus d'informations sur cet attribut, voir Attributs du gestionnaire de files d'attente.

Toutefois, si l'application s'exécute en tant que IBM MQ MQI client, le nom de l'application doit être dans le jeu de caractères et le codage du client.

Pour assurer une transition harmonieuse du nom d'application entre les gestionnaires de files d'attente et pour permettre la surveillance des ressources d'application via les rubriques de surveillance des ressources, les noms d'application doivent contenir uniquement des caractères imprimables mono-octet.

Remarques :

- Vous devez également éviter d'utiliser des barres obliques et des perluètes dans les noms d'application.

- Vous devez éviter d'utiliser le caractère perluète dans les noms d'application. Les métriques STATAPP de la rubrique système pour les noms d'application contenant une perluète ne seront pas générées.

Cela limite le nom à:

- Caractères alphanumériques: `A-Z, a-zet 0-9`

  Remarque : Vous ne devez pas utiliser les caractères en minuscules `a-z` dans les noms d'application sur les systèmes utilisant EBCDIC Katakana.
- le caractère espace
- Caractères imprimables invariants en EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`
- Le caractère /. Lorsque vous vous abonnez aux métriques de trace d'activité ou de rubrique système STATAPP pour une application dont le nom contient une barre oblique, vous devez remplacer les barres obliques par une perluète. Par exemple, pour recevoir des métriques STATAPP pour une application appelée "DEPT1/APPS/STOCKQUOTE", vous devez vous abonner à la chaîne de rubrique "$SYS/MQ/INFO/QMGR/QMBASIC/Monitor/STATAPP/DEPT1&APPS&STOCKQUOTE/INSTANCE". Les modèles d'application amqsact et amqsrua convertissent automatiquement les barres obliques en perluètes lors de la création de leurs abonnements.

Comment définir les caractères

Le tableau suivant récapitule les moyens par lesquels le nom d'application est choisi dans les différentes langues prises en charge par IBM MQ . Le moyen par lequel le nom est choisi est par ordre de priorité, le plus élevé en premier.

	Liaisons C et client	**Liaisons et client Java**	**Liaisons et client JMS**	**Client .NET géré**	**Liaisons .NET non gérées et client**	**Client XMS géré**	**Liaisons et clientXMS non gérés**
Remplacement de la propriété de connexion		Remplacement de la propriété de connexionJava		:Remplacement de la propriété de connexion .NET	:Remplacement de la propriété de connexion .NET		
Propriété remplacée		Java propriété remplacée		:Propriété NET remplacée	:Propriété NET remplacée		
Environnement MQ		Java EnvironnementMQ		:EnvironnementMQEnvironment .NET	:EnvironnementMQEnvironment .NET		
Propriété de fabrique de connexions			Propriété de fabrique de connexions			Propriété de fabrique de connexions	Propriété de fabrique de connexions

	Liaisons C et client	Liaisons et client Java	Liaisons et client JMS	Client .NET géré	Liaisons .NET non gérées et client	Client XMS géré	Liaisons et clientXMS non gérés
JMSAdmin			JMSAdmin			JMSAdmin	JMSAdmin
MQCNO	Options de connexion						
Variable d'environnement	Variables d'environnement				Variables d'environnement		Variables d'environnement
mqclient.ini (Applicable aux connexions client uniquement)	Connexions client				Connexions client		Connexions client
Java nom de classe		Nom de classeJava	Nom de classeJava				
Nom par défaut	Nom par défaut			:NET Nom par défaut	:NET Nom par défaut	:NET Nom par défaut	:NET Nom par défaut

Remarque : Les liaisons C et la colonne client s'appliquent également aux langages de programmation suivants:

- COBOL
- Assembler
- Visual Basic
- Report Program Generator

Connexions de langage de programmation

Les applications qui se résolvent dans les bibliothèques IBM MQ en C et dans d'autres langages de programmation peuvent fournir le nom de l'application de l'une des manières suivantes.

Les méthodes de connexion sont répertoriées par ordre de priorité, en commençant par la plus élevée.

Multi **Options de connexion**

- ALW MQCNO

  Remarque : z/OS Lors de la connexion à un gestionnaire de files d'attente IBM MQ for z/OS , vous pouvez uniquement définir le nom de l'application à l'aide de connexions en mode client ou d'applications IBM MQ classes for JMS ou IBM MQ classes for Java .

- IBM i MQCNO sur IBM i

ALW Variables d'environnement

Si vous n'avez pas encore sélectionné de nom d'application, vous pouvez utiliser la variable d'environnement **MQAPPLNAME** pour identifier la connexion au gestionnaire de files d'attente. Exemple :

```
export MQAPPLNAME=ExampleApplName
```

Notez que les 28 premiers caractères seulement sont utilisés et que ces caractères ne doivent pas être des blancs ou des valeurs nulles.

Remarque : L'attribut s'applique uniquement aux langages de programmation pris en charge, aux connexions .NETnon gérées et aux connexions XMS non gérées.

ALW Fichier de configuration du client

Si vous n'avez pas encore sélectionné de nom d'application et que la connexion est une connexion client, vous pouvez spécifier les informations suivantes dans le fichier de configuration client (par exemple, `mqclient.ini`) pour identifier la connexion au gestionnaire de files d'attente.

```
Connection:
    ApplName=ExampleApplName
```

Remarques :

1. Les 28 premiers caractères seulement sont utilisés, et ces caractères ne doivent pas être des blancs ou des valeurs nulles.
2. L'attribut s'applique uniquement aux connexions client sur les langages de programmation pris en charge, aux connexions .NETnon gérées et aux connexions XMS non gérées uniquement.

Pour plus d'informations, voir le fichier de configuration IBM MQ MQI client , `mqclient.ini`.

Nom par défaut

Si vous n'avez toujours pas choisi le nom de l'application, le nom par défaut continue d'être utilisé, qui contient autant de chemin et de nom d'exécutable que le système d'exploitation l'affiche. Pour plus d'informations, voir PutApplName.

Applications .NET gérées

Les applications .NET gérées peuvent fournir le nom de l'application de l'une des manières suivantes.

Les méthodes de connexion sont répertoriées par ordre de priorité, en commençant par la plus élevée.

Remplacement des propriétés de connexion

Vous pouvez fournir un fichier de substitution des détails de connexion aux applications de la manière suivante:

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

Le fichier spécifié par `overrideConnectionDetailsFile` contient une liste de propriétés préfixées par `mqj`. Les applications doivent définir la propriété **mqj.APPNAME** dans laquelle la valeur de la propriété **mqj.APPNAME** spécifie le nom utilisé pour identifier la connexion au gestionnaire de files d'attente.

Seuls les 28 premiers caractères du nom sont utilisés. Exemple :

```
mqj.APPNAME=ExampleApplName
```

Propriété remplacée

Une constante **MQC.APPNAME_PROPERTY** a été définie avec la valeur *APPNAME*. Vous pouvez désormais transmettre cette propriété au constructeur **MQQueueManager** , en utilisant uniquement les 28 premiers caractères du nom. Exemple :

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

Pour plus d'informations, voir «Opérations gérées et non gérées dans .NET», à la page 649.

Environnement MQ

La propriété *AppName* est ajoutée à la classe **MQEnvironment**et seuls les 28 premiers caractères sont utilisés. Exemple :

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

Nom par défaut

Si vous n'avez pas fourni le nom de l'application par l'un des moyens décrits dans le texte précédent, le nom de l'application est automatiquement défini pour être le nom de l'exécutable (et la partie du chemin qui convient).

Applications XMS

Les méthodes de connexion sont répertoriées par ordre de priorité, en commençant par la plus élevée.

Propriété de fabrique de connexions

Les applications XMS peuvent fournir le nom d'application sur la fabrique de connexions à l'aide de la propriété **XMSC.WMQ_APPLICATIONNAME** ("*XMSC_WMQ_APPNAME*") d'une manière similaire à JMS. Vous pouvez indiquer jusqu'à 28 caractères.

Pour plus d'informations, reportez-vous aux sections «XMS .NET Création d'objets gérés», à la page 677 et «Propriétés d'un message XMS», à la page 685.

JMSAdmin

Dans les outils d'administration, la propriété est appelée "**APPLICATIONNAME**" ou "**APPNAME**" en abrégé.

Applications de liaisons Java et JMS

Les méthodes de connexion sont répertoriées par ordre de priorité, en commençant par la plus élevée.

ALW Les applications client Java et JMS peuvent déjà spécifier un nom d'application, qui a été étendu sur IBM MQ for Multiplatforms aux applications de liaisons, en utilisant la zone MQCNO **ApplName** .

Remplacement des propriétés de connexion

La propriété **Application name** a été ajoutée à la liste des propriétés de connexion que vous pouvez remplacer. Pour plus d'informations, voir Utilisation de la substitution de propriété de connexion IBM MQ.

Avertissement : Les propriétés de connexion et la façon d'utiliser le fichier de remplacement des propriétés de connexion sont identiques pour IBM MQ classes for Java et .NET.

Propriété remplacée

Une constante **MQC.APPNAME_PROPERTY** a été définie avec la valeur *APPNAME*. Vous pouvez désormais transmettre cette propriété au constructeur **MQQueueManager** , en utilisant uniquement les 28 premiers caractères du nom. Pour plus d'informations, voir Utilisation du remplacement de propriété de connexion dans IBM MQ classes for Java.

Environnement MQ

La propriété *AppName* est ajoutée à la classe **MQEnvironment** et seuls les 28 premiers caractères sont utilisés.

Pour plus d'informations, voir «Configuration de l'environnement IBM MQ pour IBM MQ classes for Java», à la page 384.

Nom de classeJava

Si vous n'avez pas indiqué le nom de l'application par l'un des moyens indiqués dans le texte précédent, le nom de l'application est dérivé du nom de la classe principale.

Pour plus d'informations, voir «Configuration de l'environnement IBM MQ pour IBM MQ classes for Java», à la page 384.

Avertissement : IBM i Sous IBM i , il n'est pas possible d'interroger le nom de la classe principale. Par conséquent, IBM MQ client for Java est utilisé à la place.

Concepts associés

«Configuration de l'environnement IBM MQ pour IBM MQ classes for Java», à la page 384
Pour qu'une application puisse se connecter à un gestionnaire de files d'attente en mode client, elle doit spécifier le nom de canal, le nom d'hôte et le numéro de port.

Référence associée

MQCNO
MQCNO sur IBM i

Techniques de conception pour les messages

Remarques pour vous aider à concevoir des messages, y compris des remarques pour les sélecteurs et les propriétés de message.

Éléments à prendre en compte au stade de la conception

Vous créez un message lorsque vous utilisez un appel MQI pour placer le message dans une file d'attente. En entrée de l'appel, vous fournissez des informations de contrôle dans un *descripteur de message* (MQMD) et les données que vous souhaitez envoyer à un autre programme. Mais au stade de la conception, vous devez prendre en compte les éléments suivants, car ils affectent la façon dont vous créez vos messages:

Type de message à utiliser

Concevez-vous une application simple dans laquelle vous pouvez envoyer un message, puis n'effectuez aucune autre action? Ou demandes-tu une réponse à une question? Si vous posez une question, vous pouvez inclure dans le descripteur de message le nom de la file d'attente dans laquelle vous souhaitez recevoir la réponse.

Voulez-vous que vos messages de demande et de réponse soient synchrones? Cela implique que vous définissez un délai d'attente pour la réponse à votre demande et que si vous ne recevez pas la réponse dans ce délai, elle est traitée comme une erreur.

Ou préférez-vous travailler de manière asynchrone, de sorte que vos processus n'aient pas à dépendre de l'occurrence d'événements spécifiques, tels que des signaux de synchronisation communs?

Une autre considération est de savoir si vous avez tous vos messages à l'intérieur d'une unité de travail.

Affectation de priorités différentes aux messages

Vous pouvez affecter une valeur de priorité à chaque message et définir la file d'attente de sorte qu'elle conserve ses messages par ordre de priorité. Dans ce cas, lorsqu'un autre programme extrait un message de la file d'attente, il obtient toujours le message avec la priorité la plus élevée. Si la file d'attente ne conserve pas ses messages dans l'ordre de priorité, un programme qui extrait les messages de la file d'attente les extrait dans l'ordre dans lequel ils ont été ajoutés à la file d'attente.

Les programmes peuvent également sélectionner un message à l'aide de l'identificateur attribué par le gestionnaire de files d'attente lors de l'insertion du message dans la file d'attente. Vous pouvez également générer vos propres identificateurs pour chacun de vos messages.

Effet du redémarrage du gestionnaire de files d'attente sur les messages

Le gestionnaire de files d'attente conserve tous les messages persistants, en les récupérant si nécessaire à partir des fichiers journaux IBM MQ , lorsqu'il est redémarré. Les messages non persistants et les files d'attente dynamiques temporaires ne sont pas conservés. Les messages que vous ne souhaitez pas supprimer doivent être définis comme persistants lors de leur création. Lorsque vous écrivez une application pour IBM MQ for Windows ou IBM MQ sur des systèmes AIX and Linux , assurez-vous que vous savez comment votre système a été configuré en ce qui concerne l'allocation des fichiers journaux afin de réduire le risque de concevoir une application qui s'exécutera dans les limites des fichiers journaux.

z/OS Les messages des files d'attente partagées (disponibles uniquement sous IBM MQ for z/OS) sont conservés dans l'unité de couplage (CF), les messages non persistants sont conservés lors des redémarrages d'un gestionnaire de files d'attente tant que l'unité de couplage reste disponible. En cas d'échec de l'unité de couplage, les messages non persistants sont perdus.

Donner des informations sur vous-même au destinataire des messages

Généralement, le gestionnaire de files d'attente définit l'ID utilisateur, mais les applications disposant des droits appropriés peuvent également définir cette zone, afin que vous puissiez inclure votre propre ID utilisateur et d'autres informations que le programme de réception peut utiliser à des fins de comptabilité ou de sécurité.

Nombre de files d'attente de réception

Multi Si un message doit être placé dans plusieurs files d'attente, vous pouvez le publier dans une rubrique ou dans une liste de distribution.

z/OS Si un message doit être inséré dans plusieurs files d'attente, vous pouvez le publier dans une rubrique.

Sélecteurs et propriétés de message

Les messages peuvent être associés à des métadonnées en même temps que la charge de message principale. Ces propriétés de message peuvent être utiles pour fournir des données supplémentaires.

Il y a deux aspects à ces données supplémentaires qu'il est important de connaître:

- Les propriétés ne sont pas soumises à la protection Advanced Message Security (AMS). Si vous souhaitez utiliser AMS pour protéger vos données, placez-les dans le contenu et non dans les propriétés de message.
- Les propriétés peuvent être utilisées pour sélectionner des messages.

Il est important de noter que l'utilisation de sélecteurs rompt la convention de message standard du premier entré, premier sorti. Etant donné que le gestionnaire de files d'attente est optimisé pour cette charge de travail, la fourniture de sélecteurs complexes n'est pas recommandée pour des raisons de performances. Le gestionnaire de files d'attente ne stocke pas les index des propriétés de message. Par conséquent, la recherche d'un message doit être une recherche linéaire. Plus la file d'attente est profonde, plus le sélecteur est complexe et plus la probabilité que le sélecteur correspondant à un message soit faible aura un impact négatif sur les performances.

Si une sélection complexe est requise, il est conseillé de filtrer les messages à l'aide d'une application ou d'un moteur de traitement, tel que IBM Integration Bus, vers des destinations différentes. L'utilisation d'une hiérarchie de rubriques peut également être utile.

Remarque : IBM MQ classes for Java ne prennent pas en charge l'utilisation de sélecteurs. Si vous souhaitez utiliser des sélecteurs, vous devez les utiliser via l'API JMS .

Considérations relatives à la conception et aux performances des applications

Il existe un certain nombre de façons dont une mauvaise conception du programme peut affecter les performances. Ils peuvent être difficiles à détecter car le programme peut sembler bien fonctionner lui-même, mais affecter l'exécution d'autres tâches. Plusieurs problèmes spécifiques aux programmes effectuant des appels IBM MQ sont décrits dans cette rubrique.

Voici quelques idées pour vous aider à concevoir des applications efficaces:

- Concevez votre application de sorte que le traitement se déroule en parallèle avec le temps de réflexion d'un utilisateur:
  - Affiche un panneau et permet à l'utilisateur de commencer à taper pendant que l'application est encore en cours d'initialisation.
  - Obtenez les données dont vous avez besoin en parallèle à partir de différents serveurs.
- Gardez les connexions et les files d'attente ouvertes si vous prévoyez de les réutiliser au lieu de les ouvrir et de les fermer, de les connecter et de les déconnecter à plusieurs reprises.
- Toutefois, une application serveur qui ne place qu'un seul message doit utiliser MQPUT1.
- Les gestionnaires de files d'attente sont optimisés pour les messages dont la taille est comprise entre 4 et 100 Ko. Les messages de très grande taille sont inefficaces ; il est probablement préférable d'envoyer 100 messages de 1 Mo chacun plutôt qu'un seul message de 100 Mo. Les très petits messages sont également inefficaces. Le gestionnaire de files d'attente effectue la même quantité de travail pour un message mono-octet que pour un message de 4 Ko.
- Conservez vos messages dans une unité de travail afin qu'ils puissent être validés ou annulés simultanément.
- Utilisez l'option tampon pour les messages qui n'ont pas besoin d'être récupérables.
- Si vous devez envoyer un message à un certain nombre de files d'attente cible, envisagez d'utiliser une liste de distribution.

Effet de la longueur du message

La quantité de données d'un message peut affecter les performances de l'application qui traite le message. Pour obtenir les meilleures performances de votre application, envoyez uniquement les données essentielles d'un message. Par exemple, dans une demande de débit d'un compte bancaire, les seules informations qui doivent être transmises du client à l'application serveur sont le numéro de compte et le montant du débit.

Effet de la persistance des messages

Les messages persistants sont généralement consignés. La consignation des messages réduit les performances de votre application. Par conséquent, utilisez les messages persistants uniquement pour les données essentielles. Si les données d'un message peuvent être supprimées en cas d'arrêt ou d'échec du gestionnaire de files d'attente, utilisez un message non persistant.

z/OS Les opérations MQPUT et MQGET pour les messages persistants sont bloquées lorsque l'espace du journal de reprise est insuffisant pour enregistrer les opérations. Cette condition est indiquée dans le journal des travaux du gestionnaire de files d'attente par les messages CSQJ110E et CSQJ111A. S'assurer que des processus de surveillance sont en place afin que ces conditions soient gérées et évitées.

Recherche d'un message particulier

L'appel MQGET extrait généralement le premier message d'une file d'attente. Si vous utilisez les identificateurs de message et de corrélation (*MsgId* et *CorrelId*) dans le descripteur de message pour spécifier un message particulier, le gestionnaire de files d'attente doit effectuer une recherche dans la file d'attente jusqu'à ce qu'il trouve ce message. L'utilisation de l'appel MQGET de cette manière affecte les performances de votre application.

Files d'attente contenant des messages de longueurs différentes

Si votre application ne peut pas utiliser des messages de longueur fixe, agrandissez et réduisez les tampons de manière dynamique pour qu'ils correspondent à la taille de message standard. Si l'application émet un appel MQGET qui échoue car la mémoire tampon est trop petite, la taille des données de message est renvoyée. Ajoutez du code à votre application afin que la mémoire tampon soit redimensionnée en conséquence et que l'appel MQGET soit réémis.

Remarque : Si vous ne définissez pas explicitement l'attribut **MaxMsgLength** , la valeur par défaut est 4 Mo, ce qui peut être très inefficace si cette valeur est utilisée pour influencer la taille de la mémoire tampon de l'application.

Fréquence des points de synchronisation

Les programmes qui émettent un très grand nombre d'appels MQPUT ou MQGET au sein d'un point de synchronisation, sans les valider, peuvent entraîner des problèmes de performances. Les files d'attente affectées peuvent être saturées avec des messages actuellement inaccessibles, tandis que d'autres tâches peuvent être en attente pour obtenir ces messages. Cela a des implications en termes de stockage et en termes d'unités d'exécution liées à des tâches qui tentent d'obtenir des messages.

Utilisation de l'appel MQPUT1

Utilisez l'appel MQPUT1 uniquement si vous avez un seul message à insérer dans une file d'attente. Si vous souhaitez insérer plusieurs messages, utilisez l'appel MQOPEN, suivi d'une série d'appels MQPUT et d'un appel MQCLOSE unique.

Nombre d'unités d'exécution utilisées

Windows Pour IBM MQ for Windows, une application peut nécessiter un grand nombre d'unités d'exécution. Un nombre maximal d'unités d'exécution d'application est alloué à chaque processus de gestionnaire de files d'attente.

Les applications peuvent utiliser trop d'unités d'exécution. Déterminez si l'application prend en compte cette possibilité et qu'elle prend des mesures pour arrêter ou signaler ce type d'occurrence.

Placer les messages persistants sous le point de synchronisation

Les messages persistants doivent être insérés et placés sous le point de synchronisation. En effet, lors de l'obtention d'un message persistant en dehors du point de synchronisation, si l'extraction échoue, l'application ne peut pas savoir si le message a été obtenu de la file d'attente ou non et si, si le message a été obtenu, il a également été perdu. Lors de l'obtention de messages persistants sous le point de synchronisation, si quelque chose échoue, la transaction est annulée et le message persistant n'est pas perdu car il se trouve toujours dans la file d'attente.

De même, lors de l'insertion de messages persistants, placez-les sous le point de synchronisation. Une autre raison de placer et d'obtenir des messages persistants sous le point de synchronisation est que le code de message persistant dans IBM MQ est fortement optimisé pour le point de synchronisation. Ainsi, l'insertion et l'obtention de messages persistants sous le point de synchronisation sont plus rapides que l'insertion et l'obtention de messages persistants en dehors du point de synchronisation.

Si votre application place des messages persistants en dehors du point de synchronisation, le gestionnaire de files d'attente vérifie s'il peut créer un point de synchronisation implicite pour le compte

de l'application. Si le gestionnaire de files d'attente peut le faire, il inclut l'insertion dans ce point de synchronisation et le valide automatiquement. Pour une description plus détaillée, voir «Point de synchronisation implicite sur Multiplatforms», à la page 885 .

Cependant, il est plus rapide d'insérer et d'extraire des messages non persistants en dehors du point de synchronisation car le code non persistant dans IBM MQ est optimisé pour être en dehors du point de synchronisation. L'insertion et l'obtention de messages persistants s'effectue à des vitesses de disque car le message persistant est conservé sur le disque. Cependant, l'insertion et l'obtention de messages non persistants se fait à des vitesses d'UC car il n'y a pas d'écriture de disque impliquée, même pas lors de l'utilisation d'un point de synchronisation.

Si une application reçoit des messages et ne sait pas à l'avance s'ils sont persistants ou non, l'option OGM MQGMO_SYNCPOINT_IF_PERSISTENT peut être utilisée.

Techniques de conception pour les applications avancées

Lors de la conception d'applications plus avancées, vous pouvez envisager certaines techniques, telles que l'attente de messages, la corrélation des réponses, la définition et l'utilisation des informations de contexte, le démarrage automatique des applications, la génération de rapports et la suppression des affinités de message lors de l'utilisation de la mise en cluster.

Pour une application IBM MQ simple, vous devez choisir les objets IBM MQ à utiliser dans votre application et les types de message à utiliser. Pour une application plus avancée, vous pouvez utiliser certaines des techniques introduites dans les sections suivantes.

En attente de messages

Un programme qui sert une file d'attente peut attendre des messages en:

- Attente jusqu'à l'arrivée d'un message ou jusqu'à l'expiration d'un intervalle de temps spécifié (voir «En attente de messages», à la page 817).
- z/OS Sous IBM MQ for z/OS uniquement, définition d'un signal pour que le programme soit informé de l'arrivée d'un message. Pour plus d'informations, voir «Signaling», à la page 818.
- Etablissement d'un exit de rappel à utiliser lorsqu'un message arrive ; voir «Consommation asynchrone de messages IBM MQ», à la page 44.
- Effectuer des appels périodiques dans la file d'attente pour voir si un message est arrivé (*interrogation*). Cela n'est généralement pas conseillé car cela peut avoir des conséquences sur les performances.

Corrélation des réponses

Dans les applications IBM MQ , lorsqu'un programme reçoit un message lui demandant d'effectuer un travail, il envoie généralement un ou plusieurs messages de réponse au demandeur.

Pour aider le demandeur à associer ces réponses à sa demande d'origine, une application peut définir une zone *identificateur de corrélation* dans le descripteur de chaque message. Les programmes copient ensuite l'identificateur de message du message de demande dans la zone d'identificateur de corrélation de leurs messages de réponse.

Définition et utilisation des informations de contexte

Les *informations de contexte* sont utilisées pour associer des messages à l'utilisateur qui les a générés et pour identifier l'application qui a généré le message. Ces informations sont utiles pour la sécurité, la comptabilité, l'audit et l'identification des problèmes.

Lorsque vous créez un message, vous pouvez spécifier une option qui demande au gestionnaire de files d'attente d'associer des informations de contexte par défaut à votre message.

Pour plus d'informations sur l'utilisation et la définition des informations de contexte, voir «Contexte de message», à la page 49.

Démarrage automatique des programmes IBM MQ

Utilisez IBM MQ *déclenchement* pour démarrer automatiquement un programme lorsque des messages arrivent dans une file d'attente.

Vous pouvez définir des conditions de déclenchement sur une file d'attente afin qu'un programme commence à traiter cette file d'attente:

- Chaque fois qu'un message arrive dans la file d'attente
- Lorsque le premier message arrive dans la file d'attente
- Lorsque le nombre de messages dans la file d'attente atteint un nombre prédéfini

Pour plus d'informations sur le déclenchement, voir «Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888. Le déclenchement est juste une façon de démarrer un programme automatiquement. Par exemple, vous pouvez démarrer automatiquement un programme sur un temporisateur à l'aide de fonctions nonIBM MQ .

Sous Multiplateformes, IBM MQ peut définir des objets de service pour démarrer les programmes IBM MQ au démarrage du gestionnaire de files d'attente ; voir Objets de service.

Génération de rapports IBM MQ

Vous pouvez demander les rapports suivants dans une application:

- Rapports d'exception
- Rapports d'expiration
- Rapports de confirmation à l'arrivée (COA)
- Rapports de confirmation de livraison (COD)
- Rapports de notification d'action positive (PAN)
- Rapports de notification d'action négative (NAN)

Ces dernières sont décrites dans la rubrique «Messages de rapport», à la page 21.

Clusters et affinités de messages

Avant de commencer à utiliser des clusters avec plusieurs définitions pour la même file d'attente, examinez vos applications pour voir s'il existe des clusters nécessitant un échange de messages associés.

Dans un cluster, un message peut être acheminé vers n'importe quel gestionnaire de files d'attente qui héberge une instance de la file d'attente appropriée. Par conséquent, la logique des applications avec des affinités de message peut être perturbée.

Par exemple, vous pouvez avoir deux applications qui s'appuient sur une série de messages qui circulent entre elles sous la forme de questions et de réponses. Il peut être important que toutes les questions soient envoyées au même gestionnaire de files d'attente et que toutes les réponses soient renvoyées à l'autre gestionnaire de files d'attente. Dans ce cas, il est important que la routine de gestion de la charge de travail n'envoie pas les messages à un gestionnaire de files d'attente qui vient d'héberger une instance de la file d'attente appropriée.

Si possible, supprimez les affinités. La suppression des affinités de messages améliore la disponibilité et l'évolutivité des applications.

Pour plus d'informations, voir Gestion des affinités de message.

Remarques sur la conception et les performances des applications IBM i

Utilisez ces informations pour comprendre comment la conception de l'application, les unités d'exécution et le stockage peuvent affecter les performances.

Ces informations sont divisées en deux sections:

- «Considérations relatives à la conception des applications», à la page 67
- «Problèmes de performances spécifiques», à la page 68

Considérations relatives à la conception des applications

Il existe un certain nombre de façons dont une mauvaise conception du programme peut affecter les performances. Ces problèmes peuvent être difficiles à détecter car le programme peut sembler bien fonctionner, tout en affectant l'exécution d'autres tâches. Plusieurs problèmes spécifiques aux programmes qui font des appels IBM MQ for IBM i sont décrits dans les sections suivantes.

Pour plus d'informations sur la conception d'application, voir «Remarques sur la conception des applications IBM MQ», à la page 52.

Effet de la longueur du message
Bien que IBM MQ for IBM i autorise les messages à contenir jusqu'à 100 Mo de données, la quantité de données d'un message affecte les performances de l'application qui traite le message. Pour obtenir les meilleures performances de votre application, envoyez uniquement les données essentielles d'un message ; par exemple, dans une demande de débit d'un compte bancaire, les seules informations qui doivent être transmises du client à l'application serveur sont le numéro de compte et le montant du débit.

Effet de la persistance des messages
Les messages persistants sont journalisés. La journalisation des messages réduit les performances de votre application. Par conséquent, utilisez les messages persistants pour les données essentielles uniquement. Si les données d'un message peuvent être supprimées en cas d'arrêt ou d'échec du gestionnaire de files d'attente, utilisez un message non persistant.

Recherche d'un message particulier
L'appel MQGET extrait généralement le premier message d'une file d'attente. Si vous utilisez les identificateurs de message et de corrélation (*MsgId* et *CorrelId*) dans le descripteur de message pour spécifier un message particulier, le gestionnaire de files d'attente doit effectuer une recherche dans la file d'attente jusqu'à ce qu'il trouve ce message. L'utilisation de l'appel MQGET de cette manière affecte les performances de votre application.

Files d'attente contenant des messages de longueurs différentes
Si les messages d'une file d'attente ont des longueurs différentes, pour déterminer la taille d'un message, votre application peut utiliser l'appel MQGET avec la zone *BufferLength* définie sur zéro de sorte que, même si l'appel échoue, elle renvoie la taille des données de message. L'application peut alors répéter l'appel en spécifiant l'identifiant du message qu'elle a mesuré dans son premier appel et une mémoire tampon de taille correcte. Toutefois, si d'autres applications servent la même file d'attente, vous pouvez constater que les performances de votre application sont réduites car son deuxième appel MQGET passe du temps à rechercher un message qu'une autre application a extrait entre vos deux appels.

Si votre application ne peut pas utiliser des messages de longueur fixe, une autre solution à ce problème consiste à utiliser l'appel MQINQ pour trouver la taille maximale des messages que la file d'attente peut accepter, puis à utiliser cette valeur dans votre appel MQGET. La taille maximale des messages d'une file d'attente est stockée dans l'attribut **MaxMsgLen** de la file d'attente. Cette méthode peut toutefois utiliser de grandes quantités de stockage, car la valeur de cet attribut de file d'attente peut être la valeur maximale autorisée par IBM MQ for IBM i, qui peut être supérieure à 2 Go.

Fréquence des points de synchronisation
Les programmes qui émettent de nombreux appels MQPUT dans le point de synchronisation, sans les valider, peuvent entraîner des problèmes de performances. Les files d'attente affectées peuvent être saturées avec des messages actuellement inutilisables, tandis que d'autres tâches peuvent attendre d'obtenir ces messages. Ce problème a des implications en termes de stockage et en termes d'unités d'exécution liées à des tâches qui tentent d'obtenir des messages.

Utilisation de l'appel MQPUT1
Utilisez l'appel MQPUT1 uniquement si vous avez un seul message à insérer dans une file d'attente. Si vous souhaitez insérer plusieurs messages, utilisez l'appel MQOPEN, suivi d'une série d'appels MQPUT et d'un appel MQCLOSE unique.

Nombre d'unités d'exécution utilisées
Une application peut nécessiter de nombreuses unités d'exécution. Un nombre maximal d'unités d'exécution est alloué à chaque processus de gestionnaire de files d'attente. Si certaines applications sont gênantes, cela peut être dû à leur conception qui utilise un trop grand nombre d'unités d'exécution. Déterminez si l'application prend en compte cette possibilité et qu'elle prend des mesures pour arrêter ou signaler ce type d'occurrence. Le nombre maximal d'unités d'exécution autorisé par IBM i est de 4 095. Toutefois, la valeur par défaut est 64. IBM MQ met à disposition jusqu'à 63 unités d'exécution pour ses processus.

Problèmes de performances spécifiques
Cette section explique les problèmes de stockage et les performances médiocres.

Incidents de stockage
Si vous recevez le message système `CPF0907. Serious storage condition may exist.`, il est possible que l'espace associé aux gestionnaires de files d'attente IBM MQ for IBM i soit saturé.

Votre application ou IBM MQ for IBM i s'exécute-t-elle lentement?
Si votre application s'exécute lentement, elle peut indiquer qu'elle se trouve dans une boucle ou qu'elle attend une ressource qui n'est pas disponible. Cette lenteur peut également être due à un problème de performances. C'est peut-être parce que votre système fonctionne près des limites de sa capacité. Ce type de problème est probablement le plus grave aux heures de pointe de la charge du système, généralement à la mi-matin et au milieu de l'après-midi. (Si votre réseau s'étend sur plusieurs fuseaux horaires, la charge maximale du système peut vous sembler se produire à un autre moment.)

Si vous constatez que la dégradation des performances ne dépend pas du chargement du système, mais qu'elle se produit parfois lorsque le système est légèrement chargé, un programme d'application mal conçu est probablement à blâmer. Ce problème peut se manifester comme un problème qui ne se produit que lorsque certaines files d'attente sont consultées.

QTOTJOB et QADLTOTJ sont des valeurs système qui méritent d'être explorées.

Les symptômes suivants peuvent indiquer que IBM MQ for IBM i s'exécute lentement:

- Si votre système est lent à répondre aux commandes MQSC.
- Si des affichages répétés de la longueur de la file d'attente indiquent que la file d'attente est traitée lentement pour une application avec laquelle vous vous attendez à une grande quantité d'activité de file d'attente.
- La trace IBM MQ est-elle en cours d'exécution?

Linux Remarques sur la conception des applications Linux on Power Systems - Little Endian

Linux on Power Systems - Little Endian prenant en charge uniquement les applications 64 bits, aucune prise en charge n'est fournie dans IBM MQ pour les applications 32 bits.

Concepts associés

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

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

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

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

For more information about application design, see “Remarques sur la conception des applications IBM MQ” on page 52.

Effect of message length

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

Effect of message persistence

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

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

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

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

Searching for a particular message

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

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

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

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

Queues that contain messages of different lengths

Get a message, using a buffer size matching the expected size of the message. If you receive the return code indicating that the message is too long, get a bigger buffer. When the get fails in this way, the data

length returned is the size of the unconverted message data. If you specify MQGMO_CONVERT on the MQGET call, and the data expands during conversion, it still might not fit in the buffer, in which case you need to further increase the size of the buffer.

If you issue the MQGET with a buffer length of zero, it returns the size of the message and the application can then get a buffer of this size and reissue the get. If you have multiple applications processing the queue, another application might have already processed the message when the original application reissued the get. If you occasionally have large messages, you might need to get a large buffer just for these messages, and release it after the message has been processed. This should help reduce virtual storage problems if all applications have large buffers.

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

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

Frequency of syncpoints

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

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

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

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

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

Advantages of the MQPUT1 call

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

How many messages can a queue manager contain

Local Queues

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

Shared Queues

The capacity for shared queues depends on the size of the coupling facility (CF). IBM MQ uses CF list structures where fundamental storage units are entries and elements. Each message is stored as 1

entry and multiple elements containing the associated MQMD and other message data. The number of elements consumed by a single message depends on the size of the message and, for CFLEVEL(5), the offload rules in effect at MQPUT time. Fewer elements are needed when message data is offloaded to either Db2 or SMDS. Message data access is slower when the message has been offloaded. See Performance Supportpac MP1H for further comparison of performance and CPU overhead associated with message offload.

What affects performance

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

What affects how fast messages can be processed

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

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

Possible reasons for this can be:

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

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

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

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

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

How much CPU per message

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

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

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

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

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

Triggering in CICS

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

How many connections or concurrent users are supported

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

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

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

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

Other things affecting performance

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

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

Related information

MP1K: IBM MQ for z/OS 9.0 Performance Report

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

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

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 73.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 77.

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

- “Writing IMS applications using IBM MQ” on page 73

- “Writing IMS bridge applications” on page 77

Related concepts

“Présentation de l'interface de file d'attente de messages” on page 739
Découvrez les composants MQI (Message Queue Interface).

“Connexion et déconnexion d'un gestionnaire de files d'attente” on page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

“Ouverture et fermeture d'objets” on page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

“Insertion de messages dans une file d'attente” on page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

“Obtention de messages à partir d'une file d'attente” on page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

“Interrogation et définition des attributs d'objet” on page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

“Validation et annulation d'unités de travail” on page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

“Démarrage des applications IBM MQ à l'aide de déclencheurs” on page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

“Utilisation de l'interface MQI et des clusters” on page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

“Using and writing applications on IBM MQ for z/OS” on page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS Writing IMS applications using IBM MQ

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

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

- “Syncpoints in IMS applications” on page 73
- “MQI calls in IMS applications” on page 74

Restrictions

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

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

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

“Writing IMS bridge applications” on page 77
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

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

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

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

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

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

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

z/OS *MQI calls in IMS applications*

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

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

- “Server applications” on page 74
- “Inquiry applications” on page 76

Server applications

Here is an outline of the MQI server application model:

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

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

```
main()
----
Call InitIMS
If IMS initialization successful
Call InitMQM
```

```
If IBM MQ initialization successful
Call ProcessRequests
Call EndMQM
End-if
End-if

Return
```

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

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

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

Return to calling function
```

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

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

Return to calling function
```

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

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

  This involves:

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

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

  Only certain types of application (for example, a simple database update or inquiry) can be approached in this way. Although the MQI reply messages can be put with the authority of the originator of the MQI message being handled, the security implications of any IMS resource updates need to be addressed carefully.
- **Processing one message per invocation of the MPP and ensuring multiple scheduling of the MPP to process all available messages.**

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

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

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

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

Inquiry applications

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

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

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

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

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

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

z/OS Writing IMS bridge applications

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

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

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

- “How the IMS bridge deals with messages” on page 77
- “Writing IMS transaction programs through IBM MQ” on page 936

Related concepts

“Writing IMS applications using IBM MQ” on page 73
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

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

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

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

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

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

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

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

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

Note:

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

- Messages containing IMS transaction data but no MQIIH structure:

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

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

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

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

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

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

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

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

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

- “Mapping IBM MQ messages to IMS transaction types” on page 79
- “If the message cannot be put to the IMS queue” on page 79
- “IMS bridge feedback codes” on page 80
- “The MQMD fields in messages from the IMS bridge” on page 80
- “The MQIIH fields in messages from the IMS bridge” on page 81
- “Reply messages from IMS” on page 82
- “Using alternate response PCBs in IMS transactions” on page 82
- “Sending unsolicited messages from IMS” on page 82
- “Message segmentation” on page 83
- “Data conversion for messages to and from the IMS bridge” on page 83

Related concepts

"Writing IMS transaction programs through IBM MQ" on page 936
The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

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

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

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

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

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

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

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

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

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

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

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

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

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

z/OS *IMS bridge feedback codes*

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

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

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

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

z/OS *The MQMD fields in messages from the IMS bridge*

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

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

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

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

Feedback
MQFB_NONE

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

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

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

Priority
MQMD.Priority of the input message

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

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

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

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

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

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized

Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Ecriture des exits de conversion de données” on page 1010 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.
- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

Développement d'applications JMS/Jakarta Messaging et Java

IBM MQ fournit trois interfaces de langage Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSet IBM MQ classes for Java.

Pourquoi et quand exécuter cette tâche

JM 3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging est un fournisseur Jakarta Messaging qui implémente les interfaces Jakarta Messaging pour IBM MQ en tant que système de messagerie. Jakarta Connectors Architecture fournit un moyen standard de connecter des applications s'exécutant dans un environnement Jakarta EE à un système d'information d'entreprise (EIS) tel que IBM MQ ou Db2.

Pour plus d'informations, reportez-vous aux sections «Pourquoi utiliser IBM MQ classes for Jakarta Messaging ?», à la page 86 et «Accès à IBM MQ depuis Java -Choix de l'API», à la page 89.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ en tant que système de messagerie. Java Platform, Enterprise Edition Connector Architecture (JCA) fournit une méthode standard de connexion des applications en cours d'exécution dans un environnement Java EE à un système d'information d'entreprise (EIS) tel qu'IBM MQ ou Db2.

Pour plus d'informations, reportez-vous aux sections «Pourquoi utiliser IBM MQ classes for JMS ?», à la page 87 et «Accès à IBM MQ depuis Java -Choix de l'API», à la page 89.

IBM MQ classes for Java

IBM MQ classes for Java vous permet d'utiliser IBM MQ dans un environnement Java. IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

IBM MQ classes for Java encapsule l'interface MQI (Message Queue Interface), l'API IBM MQ native, et utilise le même modèle d'objet que les autres interfaces orientées objet, tandis que IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging implémentent les interfaces de messagerie Java d' Oracle et Java Community Process respectivement.

Pour plus d'informations, reportez-vous aux sections «Pourquoi utiliser IBM MQ classes for Java ?», à la page 358 et «Accès à IBM MQ depuis Java -Choix de l'API», à la page 89.

Remarque :

Stabilized IBM n'apportera pas d'autres améliorations à IBM MQ classes for Java ; cette fonctionnalité est stabilisée au niveau livré dans IBM MQ 8.0. Les applications existantes qui utilisent IBM MQ classes for Java continuent d'être entièrement prises en charge, mais les nouvelles fonctions ne seront pas ajoutées et les demandes d'amélioration seront rejetées. "Intégralement prises en charge" signifie que les incidents seront corrigés et que toute modification nécessaire suite à la modification de la configuration système requise pour IBM MQ sera apportée.

Les IBM MQ classes for Java ne sont pas pris en charge dans IMS.

Les IBM MQ classes for Java ne sont pas pris en charge dans WebSphere Liberty. Ils ne doivent pas être utilisés avec la fonction de messagerie IBM MQ Liberty ni avec le support JCA générique. Pour plus d'informations, voir Utilisation des interfaces WebSphere MQ Java dans les environnements J2EE/JEE.

Utilisation de IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 continue de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API JMS 2.0 et de l'API Jakarta Messaging 3.0 dans la même application n'est pas prise en charge.

Remarque : Pour Jakarta Messaging 3.0, le contrôle de la spécification JMS passe de Oracle à Java Community Process. Toutefois, Oracle conserve le contrôle du nom "javax", qui est utilisé dans d'autres technologies Java qui n'ont pas été déplacées vers le processus de communauté Java . Ainsi, alors que Jakarta Messaging 3.0 est fonctionnellement équivalent à JMS 2.0 , il existe des différences de dénomination:

- Le nom officiel de la version 3.0 est Jakarta Messaging et non Java Message Service.
- Les noms de pack et de constante sont précédés de `jakarta` et non de `javax`. Par exemple, dans JMS 2.0 , la connexion initiale à un fournisseur de messagerie est un objet `javax.jms.Connection` et dans Jakarta Messaging 3.0 , il s'agit d'un objet `jakarta.jms.Connection` .

JMS 2.0 Les packages javax.jms définissent les interfaces JMS et un fournisseur JMS implémente ces interfaces pour un produit de messagerie spécifique. IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ.

JM 3.0 Les packages jakarta.jms définissent les interfaces Jakarta Messaging et un fournisseur Jakarta Messaging implémente ces interfaces pour un produit de messagerie spécifique. IBM MQ classes for Jakarta Messaging est un fournisseur Jakarta Messaging qui implémente les interfaces Jakarta Messaging pour IBM MQ.

Les spécifications JMS et Jakarta Messaging s'attendent à ce que les objets ConnectionFactory et Destination soient des objets gérés. Un administrateur crée et gère des objets gérés dans un référentiel central, et une application JMS ou Jakarta Messaging extrait ces objets à l'aide de Java Naming Directory Interface (JNDI).

JMS 2.0 Pour JMS 2.0, un administrateur peut utiliser l'outil d'administration IBM MQ JMS **JMSAdmin**ou IBM MQ Explorerpour créer et gérer des objets gérés dans un référentiel central.

JM 3.0 Pour Jakarta Messaging 3.0, vous ne pouvez pas administrer JNDI à l'aide de IBM MQ Explorer. L'administration JNDI est prise en charge par la variante Jakarta Messaging 3.0 de **JMSAdmin**, qui est **JMS30Admin**.

Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

IBM MQ classes for JMS fournit également deux ensembles d'extensions à l'API JMS . Le principal objectif de ces extensions concerne la création et la configuration dynamiques des fabriques de connexions et des destinations lors de la phase d'exécution, mais elles offrent aussi une fonction qui n'est pas directement liée à la messagerie, comme l'identification des problèmes.

Les extensions IBM MQ JMS

IBM MQ classes for JMS contient des extensions implémentées dans des objets tels que des objets MQConnectionFactory, MQQueue et MQTopic. Ces objets ont des propriétés et des méthodes spécifiques à IBM MQ. Les objets peuvent être des objets administrés ou une application peut créer les objets de façon dynamique lors de la phase d'exécution. Ces extensions sont appelées extensions IBM MQ JMS .

Les extensions IBM JMS

IBM MQ classes for JMS fournit également un ensemble plus générique d'extensions à l'API JMS , qui ne sont pas spécifiques à IBM MQ en tant que système de messagerie ou à Java en tant que langage de programmation utilisé. Ces extensions sont appelées extensions IBM JMS et ont les objectifs généraux suivants:

- Fournir un niveau de cohérence plus élevé entre les fournisseurs IBM JMS .
- Pour faciliter l'écriture d'une application de pont entre deux systèmes de messagerie IBM .
- Pour faciliter le port d'une application d'un fournisseur IBM JMS à un autre.

Les extensions fournissent une fonction similaire à celle fournie dans IBM MQ Message Service Client (XMS) for C/C++ et IBM MQ Message Service Client (XMS) for .NET.

Concepts associés

Interfaces de langage IBM MQ Java

Tâches associées

«Ecriture d'applications IBM MQ classes for JMS/Jakarta Messaging», à la page 144

Après une brève introduction au modèle JMS , cette section fournit des conseils détaillés sur la façon d'écrire des applications IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging .

JM 3.0 Pourquoi utiliser IBM MQ classes for Jakarta Messaging ?

L'utilisation d' IBM MQ classes for Jakarta Messaging offre un certain nombre d'avantages, notamment la possibilité de réutiliser les compétences Jakarta Messaging existantes dans votre organisation, ainsi que des applications plus indépendantes du fournisseur Jakarta Messaging et de la configuration IBM MQ sous-jacente.

Récapitulatif des avantages de l'utilisation de IBM MQ classes for Jakarta Messaging

L'utilisation de IBM MQ classes for Jakarta Messaging vous permet de réutiliser des compétences Jakarta Messaging existantes et d'assurer l'indépendance des applications.

- Vous pouvez réutiliser des compétences Jakarta Messaging .

  IBM MQ classes for Jakarta Messaging est un fournisseur Jakarta Messaging qui implémente les interfaces Jakarta Messaging pour IBM MQ en tant que système de messagerie. Si votre organisation est nouvelle dans IBM MQ, mais qu'elle possède déjà des compétences en développement d'application Jakarta Messaging (ou JMS), il peut être plus facile d'utiliser l'API Jakarta Messaging familière pour accéder aux ressources IBM MQ plutôt que l'une des autres API fournies avec IBM MQ.
- Jakarta Messaging est une partie intégrante de Jakarta EE.

  Jakarta Messaging est l'API naturelle à utiliser pour la messagerie sur la plateforme Jakarta EE . Chaque serveur d'applications compatible avec Jakarta EE doit inclure un fournisseur Jakarta Messaging . Vous pouvez utiliser Jakarta Messaging dans les clients d'application, les servlets, les pages JSP (Java Server Pages), les beans enterprise Java (EJB) et les beans gérés par message (MDB). Notez en particulier que les applications Jakarta EE utilisent des beans gérés par message pour traiter les messages de manière asynchrone et que tous les messages sont distribués aux beans gérés par message en tant que messages Jakarta Messaging .
- Les fabriques de connexions et les destinations peuvent être stockées en tant qu'objets administrés Jakarta Messaging dans un référentiel central au lieu d'être codées en dur dans une application.

  Un administrateur peut créer et gérer des objets administrés Jakarta Messaging dans un référentiel central, et les applications IBM MQ classes for Jakarta Messaging peuvent extraire ces objets à l'aide de Java Naming Directory Interface (JNDI). Les fabriques de connexions et les destinations Jakarta Messaging encapsulent des informations spécifiques à IBM MQ, telles que les noms de gestionnaire de files d'attente, les noms de canal, les options de connexion, les noms de file d'attente et les noms de rubrique. Si les fabriques de connexions et les destinations sont stockées en tant qu'objets

gérés, ces informations ne sont pas codées en dur dans une application. Cette disposition offre donc à l'application une certaine indépendance par rapport à la configuration IBM MQ sous-jacente.

- Jakarta Messaging est une API standard qui peut fournir la portabilité des applications.

  Une application Jakarta Messaging peut utiliser JNDI pour extraire des fabriques de connexions et des destinations stockées en tant qu'objets gérés et utiliser uniquement les interfaces définies dans le package `jakarta.jms` (Jakarta Messaging 3.0) pour effectuer des opérations de messagerie. L'application est alors entièrement indépendante de tout fournisseur Jakarta Messaging , tel que IBM MQ classes for Jakarta Messaging, et peut être portée d'un fournisseur Jakarta Messaging à un autre sans modification de l'application.

  Si JNDI n'est pas disponible dans un environnement d'application particulier, une application IBM MQ classes for Jakarta Messaging peut utiliser des extensions de l'API Jakarta Messaging pour créer et configurer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution. L'application est alors entièrement autonome, mais elle est liée à IBM MQ classes for Jakarta Messaging en tant que fournisseur Jakarta Messaging .

- Les applications de pont peuvent être plus faciles à écrire à l'aide de Jakarta Messaging.

  Une application de pont est une application qui reçoit des messages d'un système de messagerie et les envoie à un autre système de messagerie. L'écriture d'une application pont peut être compliquée à l'aide d'API et de formats de message spécifiques au produit. A la place, vous pouvez écrire une application pont en utilisant deux fournisseurs Jakarta Messaging , un pour chaque système de messagerie. L'application utilise ensuite une seule API, l'API Jakarta Messaging , et traite uniquement les messages Jakarta Messaging .

Environnements déployables

Pour fournir l'intégration à un serveur d'applications Jakarta EE, les normes Jakarta EE exigent des fournisseurs de messagerie qu'ils offrent un adaptateur de ressources. Conformément à la spécification Jakarta Connectors Architecture , IBM MQ fournit un adaptateur de ressources qui utilise Jakarta Messaging pour fournir des fonctions de messagerie dans tout environnement Jakarta EE certifié. Pour plus d'informations, voir «Liberty et l'adaptateur de ressources IBM MQ», à la page 451.

Remarque : WebSphere Application Server traditional ne prend actuellement pas en charge Jakarta EE.

En dehors de l'environnement Jakarta EE, les fichiers OSGi et JAR sont fournis pour vous permettre d'obtenir plus facilement les seules IBM MQ classes for Jakarta Messaging. Ces fichiers JAR sont plus facilement déployables en mode autonome ou dans des infrastructures de gestion de logiciels telles que Maven. Pour plus d'informations, voir «Obtention du IBM MQ classes for JMS et du IBM MQ classes for Jakarta Messaging séparément», à la page 131.

Concepts associés

IBM MQ classes for Jakarta Messaging: présentation

«Accès à IBM MQ depuis Java -Choix de l'API», à la page 89

IBM MQ fournit trois interfaces de langage Java .

JMS 2.0 Pourquoi utiliser IBM MQ classes for JMS ?

L'utilisation d' IBM MQ classes for JMS offre un certain nombre d'avantages, notamment la possibilité de réutiliser les compétences JMS existantes dans votre organisation, ainsi que des applications plus indépendantes du fournisseur JMS et de la configuration IBM MQ sous-jacente.

Récapitulatif des avantages de l'utilisation de IBM MQ classes for JMS

L'utilisation de IBM MQ classes for JMS vous permet de réutiliser des compétences JMS existantes et d'assurer l'indépendance des applications.

Remarque : JMS 2.0 a été remplacé par Jakarta Messaging. IBM MQ classes for JMS continue de prendre en charge la norme JMS 2.0 , mais les futures améliorations apportées à la messagerie Java n'apparaîtront que dans Jakarta Messaging, et donc dans IBM MQ classes for Jakarta Messaging. Les IBM

MQ classes for JMS sont uniquement recommandées pour la maintenance et l'extension des applications JMS 2.0 existantes. IBM MQ classes for Jakarta Messaging doit être la technologie préférée pour les nouveaux développements.

- Vous pouvez réutiliser des compétences JMS .

  IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ en tant que système de messagerie. Si votre organisation est nouvelle pour IBM MQ, mais qu'elle possède déjà des compétences en développement d'application JMS , il peut être plus facile d'utiliser l'API JMS familière pour accéder aux ressources IBM MQ que l'une des autres API fournies avec IBM MQ.
- JMS est une partie intégrante de Java Platform, Enterprise Edition (Java EE).

  JMS est l'API naturelle à utiliser pour la messagerie sur la plateforme Java EE . Chaque serveur d'applications compatible avec Java EE doit inclure un fournisseur JMS . Vous pouvez utiliser JMS dans les clients d'application, les servlets, les pages JSP (Java Server Pages), les beans enterprise Java (EJB) et les beans gérés par message (MDB). Notez en particulier que les applications Java EE utilisent des beans gérés par message pour traiter les messages de manière asynchrone et que tous les messages sont distribués aux beans gérés par message en tant que messages JMS .
- Les fabriques de connexions et les destinations peuvent être stockées en tant qu'objets administrés JMS dans un référentiel central au lieu d'être codées en dur dans une application.

  Un administrateur peut créer et gérer des objets administrés JMS dans un référentiel central, et les applications IBM MQ classes for JMS peuvent extraire ces objets à l'aide de Java Naming Directory Interface (JNDI). Les fabriques de connexions et les destinations JMS encapsulent des informations spécifiques à IBM MQ, telles que les noms de gestionnaire de files d'attente, les noms de canal, les options de connexion, les noms de file d'attente et les noms de rubrique. Si les fabriques de connexions et les destinations sont stockées en tant qu'objets gérés, ces informations ne sont pas codées en dur dans une application. Cette disposition offre donc à l'application une certaine indépendance par rapport à la configuration IBM MQ sous-jacente.
- JMS est une API standard qui peut fournir la portabilité des applications.

  Une application JMS peut utiliser JNDI pour extraire des fabriques de connexions et des destinations qui sont stockées en tant qu'objets gérés et utiliser uniquement les interfaces définies dans le package `javax.jms` pour effectuer des opérations de messagerie. L'application est alors entièrement indépendante de tout fournisseur JMS , tel que IBM MQ classes for JMS, et peut être portée d'un fournisseur JMS à un autre sans modification de l'application.

  Si JNDI n'est pas disponible dans un environnement d'application particulier, une application IBM MQ classes for JMS peut utiliser des extensions de l'API JMS pour créer et configurer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution. L'application est alors entièrement autonome, mais elle est liée à IBM MQ classes for JMS en tant que fournisseur JMS .
- Les applications de pont peuvent être plus faciles à écrire à l'aide de JMS.

  Une application de pont est une application qui reçoit des messages d'un système de messagerie et les envoie à un autre système de messagerie. L'écriture d'une application pont peut être compliquée à l'aide d'API et de formats de message spécifiques au produit. A la place, vous pouvez écrire une application pont en utilisant deux fournisseurs JMS , un pour chaque système de messagerie. L'application utilise ensuite une seule API, l'API JMS , et traite uniquement les messages JMS .

Environnements déployables

Pour fournir l'intégration à un serveur d'applications Java EE, les normes Java EE exigent des fournisseurs de messagerie qu'ils offrent un adaptateur de ressources. Conformément à la spécification Java EE Connector Architecture (JCA), IBM MQ fournit un adaptateur de ressources qui utilise JMS pour offrir des fonctions de messagerie dans n'importe quel environnement Java EE certifié.

S'il est possible d'utiliser les IBM MQ classes for Java dans Java EE, cette API n'est pas conçue ni optimisée dans cette optique. Pour plus d'informations sur les remarques relatives à IBM MQ classes for Java dans Java EE, voir «Exécution d'applications IBM MQ classes for Java dans Java EE», à la page 359.

En dehors de l'environnement Java EE, les fichiers OSGi et JAR sont fournis pour vous permettre d'obtenir plus facilement les seules IBM MQ classes for JMS. Ces fichiers JAR sont plus facilement déployables en mode autonome ou dans des infrastructures de gestion de logiciels telles que Maven. Pour plus d'informations, voir «Obtention du IBM MQ classes for JMS et du IBM MQ classes for Jakarta Messaging séparément», à la page 131.

Concepts associés

IBM MQ classes for Jakarta Messaging: présentation

«Pourquoi utiliser IBM MQ classes for Jakarta Messaging ?», à la page 86
L'utilisation d' IBM MQ classes for Jakarta Messaging offre un certain nombre d'avantages, notamment la possibilité de réutiliser les compétences Jakarta Messaging existantes dans votre organisation, ainsi que des applications plus indépendantes du fournisseur Jakarta Messaging et de la configuration IBM MQ sous-jacente.

«Accès à IBM MQ depuis Java -Choix de l'API», à la page 89
IBM MQ fournit trois interfaces de langage Java .

Accès à IBM MQ depuis Java -Choix de l'API

IBM MQ fournit trois interfaces de langage Java .

- IBM MQ classes for Jakarta Messaging
- IBM MQ classes for JMS
- IBM MQ classes for Java

IBM MQ classes for Jakarta Messaging

IBM MQ classes for Jakarta Messaging permet aux applications écrites à l'aide des API Jakarta Messaging 3.0 d'utiliser IBM MQ comme fournisseur de messagerie.

Jakarta Messaging est la direction stratégique de la messagerie dans les applications Java .

Jakarta Messaging 3.0 étant fonctionnellement équivalent à JMS 2.0, pour plus d'informations, voir «Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85.

IBM MQ classes for JMS

IBM MQ classes for JMS permet aux applications écrites à l'aide des API JMS 2.0 d'utiliser IBM MQ comme fournisseur de messagerie.

Comme Jakarta Messaging remplace JMS, il est recommandé d'utiliser IBM MQ classes for JMS dans les applications existantes ou dans les environnements (par exemple, WebSphere Application Server) qui ne prennent pas en charge Jakarta Messaging.

Il n'est pas possible d'utiliser à la fois IBM MQ classes for Jakarta Messaging et IBM MQ classes for JMS dans la même application.

Pour plus d'informations, voir «Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85.

IBM MQ classes for Java

Stabilized L'autre API que les applications Java peuvent utiliser pour accéder aux ressources IBM MQ est IBM MQ classes for Java, qui fournit une API orientée IBM MQpermettant aux programmes d'utiliser IBM MQ comme fournisseur de messagerie. Toutefois, IBM MQ classes for Java est stabilisé fonctionnellement au niveau fourni dans IBM MQ 8.0. Pour plus d'informations, voir «Pourquoi utiliser IBM MQ classes for Java ?», à la page 358. Bien que les applications existantes qui utilisent IBM MQ classes for Java continuent d'être entièrement prises en charge, les nouvelles applications doivent utiliser IBM MQ classes for Jakarta Messaging.

Fonctions communes de IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging permettent d'accéder aux fonctions de messagerie point-à-point et de publication / abonnement d' IBM MQ. Outre l'envoi de messages JMS qui assurent la prise en charge du modèle de messagerie standard JMS, les applications peuvent également envoyer et recevoir des messages sans en-têtes supplémentaires et peuvent donc interopérer avec d'autres applications IBM MQ, comme les applications C MQI. Le contrôle complet des charges de message MQMD et MQ est disponible.

D'autres fonctions IBM MQ, comme la diffusion en flux des messages, et les messages d'insertion asynchrone et de rapport, sont disponibles.

A l'aide des classes auxiliaires PCF fournies, les messages d'administration PCF IBM MQ peuvent être envoyés et reçus via l'API JMS et être utilisés pour gérer les gestionnaires de files d'attente.

Les fonctions qui ont récemment été ajoutées à IBM MQ, telles que la consommation asynchrone et la reconnexion automatique, ne sont pas disponibles dans IBM MQ classes for Java, mais dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging.

Demande d'améliorations

Si vous avez besoin d'accéder à des fonctions IBM MQ qui ne sont pas disponibles via IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, vous pouvez vous faire une idée.

IBM peut ensuite indiquer si l'implémentation est possible dans l'implémentation IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , ou si une meilleure pratique peut être suivie.

IBM étant un contributeur à la norme standard, des fonctions supplémentaires peuvent être mises en oeuvre dans le cadre du processus JCP. Elles ne s'appliquent qu'à Jakarta Messaging.

Information associée

Bienvenue dans le portail IBM Ideas Portal

Processus de révision de la spécification JMS Java

Utilisation de JMS pour l'envoi de messages PCF

JM 3.0 Prérequis pour IBM MQ classes for Jakarta Messaging

Cette rubrique vous indique ce que vous devez savoir avant d'utiliser IBM MQ classes for Jakarta Messaging. Pour développer et exécuter des applications IBM MQ classes for Jakarta Messaging , vous avez besoin de certains composants logiciels comme prérequis.

Pour plus d'informations sur les prérequis pour IBM MQ classes for Jakarta Messaging, voir Configuration système requise pour IBM MQ.

Pour développer des applications IBM MQ classes for Jakarta Messaging , vous avez besoin d'un kit de développement de logiciels (SDK) Java SE . Pour plus d'informations sur les JDK pris en charge par votre système d'exploitation, voir Configuration système requise pour IBM MQ.

Pour exécuter des applications IBM MQ classes for Jakarta Messaging , vous avez besoin des composants logiciels suivants:

- Un gestionnaire de files d'attente IBM MQ .
- Un Java runtime environment (JRE), pour chaque système sur lequel vous exécutez des applications.
- IBM i Pour IBM i, Qshell, qui est l'option 30 du système d'exploitation.
- z/OS Pour z/OS, z/OS UNIX System Services (z/OS UNIX).

Le fournisseur JSSE IBM inclut un fournisseur cryptographique certifié FIPS. Il peut donc être configuré à l'aide d'un programme pour la conformité à la norme FIPS 140-2, prêt à être utilisé immédiatement. Par conséquent, la conformité à la norme FIPS 140-2 peut être prise en charge directement à partir de IBM MQ classes for Jakarta Messaging.

Le fournisseur JSSE d' Oracle peut disposer d'un fournisseur cryptographique certifié FIPS qui y est configuré, mais il n'est pas prêt à être utilisé immédiatement et n'est pas disponible pour la configuration par programmation. Par conséquent, dans ce cas, IBM MQ classes for Jakarta Messaging ne peut pas activer directement la conformité à la norme FIPS 140-2. Il se peut que vous puissiez activer manuellement cette conformité, mais IBM ne peut actuellement pas fournir de conseils à ce sujet.

Vous pouvez utiliser des adresses Internet Protocol version 6 (IPv6) dans vos applications IBM MQ classes for Jakarta Messaging si les adresses IPv6 sont prises en charge par votre machine virtuelle Java (JVM) et l'implémentation TCP/IP sur votre système d'exploitation. L'outil d'administration IBM MQ Jakarta Messaging , **JMS30Admin**, accepte également les adresses IPv6 . Pour plus d'informations sur cet outil, voir Configuration des objets JMS et Jakarta Messaging à l'aide des outils d'administration.

L'outil d'administration IBM MQ JMS et IBM MQ Explorer utilisent Java Naming Directory Interface (JNDI) pour accéder à un service d'annuaire qui stocke les objets gérés. Les applications IBM MQ classes for Jakarta Messaging peuvent également utiliser JNDI pour extraire des objets gérés d'un service d'annuaire.

Remarque : Pour Jakarta Messaging 3.0, vous ne pouvez pas administrer JNDI à l'aide de IBM MQ Explorer. L'administration JNDI est prise en charge par la variante Jakarta Messaging 3.0 de **JMSAdmin**, qui est **JMS30Admin**.

Un fournisseur de services est un code qui permet d'accéder à un service d'annuaire en mappant des appels JNDI au service d'annuaire. Un fournisseur de services de système de fichiers dans les fichiers `fscontext.jar` et `providerutil.jar` est fourni avec IBM MQ classes for Jakarta Messaging. Le fournisseur de services de système de fichiers permet d'accéder à un service d'annuaire basé sur le système de fichiers local.

Si vous prévoyez d'utiliser un service d'annuaire basé sur un serveur LDAP, vous devez installer et configurer un serveur LDAP ou avoir accès à un serveur LDAP existant. En particulier, vous devez configurer le serveur LDAP pour stocker les objets Java . Pour plus d'informations sur l'installation et la configuration de votre serveur LDAP, voir la documentation fournie avec le serveur.

JMS 2.0 Prérequis pour IBM MQ classes for JMS

Cette rubrique vous indique ce que vous devez savoir avant d'utiliser IBM MQ classes for JMS. Pour développer et exécuter des applications IBM MQ classes for JMS , vous avez besoin de certains composants logiciels comme prérequis.

Pour plus d'informations sur les prérequis pour IBM MQ classes for JMS, voir Configuration système requise pour IBM MQ.

Pour développer des applications IBM MQ classes for JMS , vous avez besoin d'un kit de développement de logiciels (SDK) Java SE . Pour plus d'informations sur les JDK pris en charge par votre système d'exploitation, voir Configuration système requise pour IBM MQ.

Pour exécuter des applications IBM MQ classes for JMS , vous avez besoin des composants logiciels suivants:

- Un gestionnaire de files d'attente IBM MQ .
- Un Java runtime environment (JRE), pour chaque système sur lequel vous exécutez des applications.
- IBM i Pour IBM i, Qshell, qui est l'option 30 du système d'exploitation.
- z/OS Pour z/OS, z/OS UNIX System Services (z/OS UNIX).

Le fournisseur JSSE IBM inclut un fournisseur cryptographique certifié FIPS. Il peut donc être configuré à l'aide d'un programme pour la conformité à la norme FIPS 140-2, prêt à être utilisé immédiatement. Par conséquent, la conformité à la norme FIPS 140-2 peut être prise en charge directement à partir de IBM MQ classes for Java et IBM MQ classes for JMS.

Le fournisseur JSSE d' Oracle peut disposer d'un fournisseur cryptographique certifié FIPS qui y est configuré, mais il n'est pas prêt à être utilisé immédiatement et n'est pas disponible pour la configuration par programmation. Par conséquent, dans ce cas, IBM MQ classes for Java et IBM MQ classes for JMS ne

peuvent pas activer directement la conformité à la norme FIPS 140-2. Il se peut que vous puissiez activer manuellement cette conformité, mais IBM ne peut actuellement pas fournir de conseils à ce sujet.

Vous pouvez utiliser des adresses Internet Protocol version 6 (IPv6) dans vos applications IBM MQ classes for JMS si les adresses IPv6 sont prises en charge par votre machine virtuelle Java (JVM) et l'implémentation TCP/IP sur votre système d'exploitation. L'outil d'administration IBM MQ JMS (voir Configuration des objets JMS à l'aide de l'outil d'administration) accepte également les adresses IPv6 .

L'outil d'administration IBM MQ JMS et IBM MQ Explorer utilisent Java Naming Directory Interface (JNDI) pour accéder à un service d'annuaire qui stocke les objets gérés. Les applications IBM MQ classes for JMS peuvent également utiliser JNDI pour extraire des objets gérés d'un service d'annuaire. Un fournisseur de services est un code qui permet d'accéder à un service d'annuaire en mappant des appels JNDI au service d'annuaire. Un fournisseur de services de système de fichiers dans les fichiers `fscontext.jar` et `providerutil.jar` est fourni avec IBM MQ classes for JMS. Le fournisseur de services de système de fichiers permet d'accéder à un service d'annuaire basé sur le système de fichiers local.

Si vous prévoyez d'utiliser un service d'annuaire basé sur un serveur LDAP, vous devez installer et configurer un serveur LDAP ou avoir accès à un serveur LDAP existant. En particulier, vous devez configurer le serveur LDAP pour stocker les objets Java . Pour plus d'informations sur l'installation et la configuration de votre serveur LDAP, voir la documentation fournie avec le serveur.

Installation et configuration de IBM MQ classes for JMS/Jakarta Messaging

Cette section décrit les répertoires et les fichiers créés lorsque vous installez IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, et explique comment configurer IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging après l'installation.

Concepts associés

«Utilisation de l'adaptateur de ressources IBM MQ», à la page 445
L'adaptateur de ressources permet aux applications qui s'exécutent dans un serveur d'applications d'accéder aux ressources IBM MQ . Il prend en charge les communications entrantes et sortantes.

Ce qui est installé pour IBM MQ classes for JMS

Un certain nombre de fichiers et de répertoires sont créés lorsque vous installez IBM MQ classes for JMS. Sous Windows, une partie de la configuration est effectuée lors de l'installation en définissant automatiquement des variables d'environnement. Sur d'autres plateformes et dans certains environnements Windows , vous devez définir des variables d'environnement avant de pouvoir exécuter des applications IBM MQ classes for JMS .

Pour la plupart des systèmes d'exploitation, les IBM MQ classes for JMS sont installés en tant que composant facultatif lorsque vous installez IBM MQ.

Pour plus d'informations sur l'installation de IBM MQ, voir:

Multi Installation de IBM MQ

z/OS Installation de IBM MQ for z/OS

Important : Outre les fichiers JAR relocalisables décrits dans «Fichiers JAR IBM MQ classes for JMS/Jakarta Messaging relocalisables», à la page 94, la copie des fichiers JAR ou des bibliothèques natives IBM MQ classes for JMS sur d'autres machines ou à un autre emplacement sur une machine où IBM MQ classes for JMS a été installé n'est pas prise en charge.

Répertoire d'installation

Tableau 5, à la page 93 indique où les fichiers IBM MQ classes for JMS sont installés sur chaque plateforme.

Tableau 5. Répertoires d'installation de IBM MQ classes for JMS	
Plateforme	**Répertoire**
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/java

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Le répertoire d'installation inclut le contenu suivant:

- Les fichiers JAR IBM MQ classes for JMS , y compris les fichiers JAR relocalisables, qui se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\lib .
- Les bibliothèques natives IBM MQ , qui sont utilisées par les applications qui utilisent l'interface native Java .

  Les bibliothèques natives 32 bits sont installées dans le répertoire *MQ_INSTALLATION_PATH*\java\lib et les bibliothèques natives 64 bits se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\lib64 .

  Pour plus d'informations sur les bibliothèques natives IBM MQ , voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 100.
- Scripts supplémentaires décrits dans «Scripts fournis avec IBM MQ classes for JMS/ Jakarta Messaging», à la page 127. Ces scripts se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\bin .
- Spécifications de l'API IBM MQ classes for JMS . L'outil Javadoc a été utilisé pour générer les pages HTML qui contiennent les spécifications de l'API.

  Les pages HTML se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses :

  – ALW Sous AIX, Linux, and Windows, ce sous-répertoire contient les pages HTML individuelles.

  – IBM i Sous IBM i, les pages HTML se trouvent dans un fichier appelé wmqjms_javadoc.jar.

  – z/OS Sous z/OS, les pages HTML se trouvent dans un fichier appelé wmqjms_javadoc.jar.
- Prise en charge de OGSi. Les bundles OSGi sont installés dans le répertoire java\lib\OSGi et décrits dans «Prise en charge d'OSGi avec IBM MQ classes for JMS», à la page 128.
- L'adaptateur de ressources IBM MQ , qui peut être déployé dans n'importe quel serveur d'applications compatible Java Platform, Enterprise Edition 7 (Java EE 7) ou Jakarta EE .

  L'adaptateur de ressources IBM MQ se trouve dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\jca ; pour plus d'informations, voir «Utilisation de l'adaptateur de ressources IBM MQ», à la page 445
- Windows Sous Windows, les symboles pouvant être utilisés pour le débogage sont installés dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\symbols .

Le répertoire d'installation inclut également certains fichiers appartenant à d'autres composants IBM MQ .

Modèles d'application

JMS 2.0 Certains modèles d'application sont fournis avec IBM MQ classes for JMS. Le Tableau 6, à la page 94 indique où les modèles d'application sont installés sur chaque plateforme.

JM 3.0 Pour IBM MQ classes for Jakarta Messaging, de nouveaux exemples sont en cours de préparation.

JMS 2.0

Tableau 6. Répertoires d'exemples pour IBM MQ classes for JMS

Plateforme	Répertoire
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/samp/jms
Windows Windows	*MQ_INSTALLATION_PATH*\tools\jms
IBM i IBM i	/QIBM/ProdData/mqm/java/samples/jms
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/samples/jms

Dans ce tableau, *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Après l'installation, vous devrez peut-être effectuer certaines tâches de configuration pour compiler et exécuter des applications.

«Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96 décrit le chemin d'accès aux classes requis pour exécuter des exemples d'applications IBM MQ classes for JMS . Cette rubrique décrit également les fichiers JAR supplémentaires qui doivent être référencés dans des circonstances particulières et les variables d'environnement que vous devez définir pour exécuter les scripts fournis avec IBM MQ classes for JMS.

Pour contrôler les propriétés, telles que le traçage et la consignation d'une application, vous devez fournir un fichier de propriétés de configuration. Le fichier de propriétés de configuration IBM MQ classes for JMS est décrit dans «Le fichier de configuration IBM MQ classes for JMS/Jakarta Messaging», à la page 102.

Concepts associés

Problèmes liés au déploiement de l'adaptateur de ressources

Tâches associées

«Utilisation des modèles d'application IBM MQ classes for JMS», à la page 124
Les modèles d'application IBM MQ classes for JMS fournissent une présentation des fonctions communes de l'API JMS . Vous pouvez les utiliser pour vérifier la configuration de votre installation et de votre serveur de messagerie et pour vous aider à créer vos propres applications.

Fichiers JAR IBM MQ classes for JMS/Jakarta Messaging relocalisables
Les fichiers JAR relocalisables peuvent être déplacés vers des systèmes qui doivent exécuter IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging.

Important :

- Outre les fichiers JAR rélocalisables décrits dans la rubrique Fichiers JAR rélocalisables, la copie des fichiers JAR IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ou des bibliothèques natives vers d'autres machines ou vers un autre emplacement sur une machine où IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ont été installés n'est pas prise en charge.
- N'incluez pas les fichiers JAR relocalisables dans les applications déployées sur les serveurs d'applications Java EE , tels que WebSphere Application Server ou WebSphere Liberty. Dans ces environnements, l'adaptateur de ressources IBM MQ doit être déployé et utilisé à la place. Notez que

WebSphere Application Server intègre l'adaptateur de ressources IBM MQ , il n'est donc pas nécessaire de le déployer manuellement dans cet environnement.

- Pour éviter les conflits de chargeur de classe, il n'est pas recommandé de regrouper les fichiers JAR relocalisables dans plusieurs applications au sein du même environnement d'exécution Java . Dans ce scénario, rendez les fichiers JAR relocalisables IBM MQ disponibles dans le chemin d'accès aux classes de l'environnement d'exécution Java .
- Si vous groupez les fichiers JAR relocalisables dans vos applications, veillez à inclure tous les fichiers JAR prérequis, comme décrit dans Fichiers JAR relocalisables. Vous devez également vous assurer que vous disposez des procédures appropriées pour mettre à jour les fichiers JAR intégrés dans le cadre de la maintenance de l'application, afin de vous assurer que IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging sont toujours à jour et que les problèmes connus font l'objet d'une nouvelle médiation.

Fichiers JAR relocalisables

Dans une entreprise, les fichiers suivants peuvent être déplacés vers des systèmes qui doivent exécuter IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging:

- `bcpkix-jdk15to18.jar` «4», à la page 95
- V 9.4.0 `bcpkix-jdk18on.jar` «3», à la page 95
- `bcprov-jdk15to18.jar` «4», à la page 95
- V 9.4.0 `bcprov-jdk18on.jar` «3», à la page 95
- `bcutil-jdk15to18.jar` «4», à la page 95
- V 9.4.0 `bcutil-jdk18on.jar` «3», à la page 95
- JMS 2.0 `com.ibm.mq.allclient.jar` «1», à la page 95
- JM 3.0 `com.ibm.mq.jakarta.client.jar` «2», à la page 95
- `fscontext.jar`
- `jakarta.jms-api.jar`
- `jms.jar`
- `org.json.jar`
- `providerutil.jar`

Remarques :

1. JMS 2.0 et JMS 1.1
2. Jakarta Messaging 3.0
3. Du IBM MQ 9.4.0
4. AvantIBM MQ 9.4.0

JMS Fichiers JAR

`jms.jar` contient les interfaces JMS 1.1 et JMS 2.0 -elles sont nommées `javax.jms.*`.

JM 3.0 `jakarta.jms-api.jar` contient les interfaces Jakarta Messaging 3.0 -elles sont nommées `jakarta.jms.*`.

fscontext.jar et providerutil.jar

Les fichiers `fscontext.jar` et `providerutil.jar` sont requis si votre application effectue des recherches JNDI à l'aide d'un contexte de système de fichiers.

Le fournisseur de sécurité Bouncy Castle et CMS prennent en charge les fichiers JAR

Le fournisseur de sécurité Bouncy Castle et les fichiers JAR de support CMS sont requis. Pour plus d'informations, voir Support for non-IBM JREs with AMS.

V 9.4.0 Les fichiers JAR suivants sont requis:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

Le fichier `org.json.jar` est requis si votre application IBM MQ classes for JMS utilise une table de définition de canal du client au format JSON.

com.ibm.mq.allclient.jar et com.ibm.mq.jakarta.client.jar

Les fichiers `com.ibm.mq.allclient.jar` et `com.ibm.mq.jakarta.client.jar` contiennent les classes IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging, IBM MQ classes for Java, PCF et Headers. Si vous déplacez ces fichiers JAR vers un nouvel emplacement, veillez à prendre les mesures nécessaires pour conserver ce nouvel emplacement avec les nouveaux groupes de correctifs IBM MQ . Assurez-vous également que l'utilisation des fichiers est connue du support IBM si vous obtenez un correctif temporaire.

Pour déterminer la version des fichiers `com.ibm.mq.allclient.jar` et `com.ibm.mq.jakarta.client.jar`, utilisez la commande suivante:

JM 3.0
```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
java -jar com.ibm.mq.allclient.jar
```

L'exemple suivant illustre un exemple de sortie de cette commande:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       WebSphere MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging
Avant de pouvoir compiler et exécuter des applications IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , le paramètre de votre variable d'environnement **CLASSPATH** doit inclure le fichier

JAR (Java archive) IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . En fonction de vos besoins, vous devrez peut-être ajouter d'autres fichiers JAR à votre chemin d'accès aux classes. Pour exécuter les scripts fournis avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, d'autres variables d'environnement doivent être définies.

Avant de commencer

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Important : La définition de l'option Java *-Xbootclasspath* pour inclure IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging n'est pas prise en charge.

Pourquoi et quand exécuter cette tâche

Pour compiler et exécuter des applications IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , utilisez le paramètre **CLASSPATH** pour votre plateforme et la version de messagerie Java , comme indiqué dans les tableaux suivants. Vous pouvez également spécifier le chemin d'accès aux classes dans la commande **java** au lieu d'utiliser la variable d'environnement.

JMS 2.0 Pour IBM MQ classes for JMS, le paramètre inclut le répertoire samples, afin que vous puissiez compiler et exécuter les exemples d'application IBM MQ classes for JMS .

JM 3.0 Pour IBM MQ classes for Jakarta Messaging, de nouveaux exemples sont en cours de préparation.

JM 3.0

*Tableau 7. Paramètres **CLASSPATH** pour Jakarta Messaging 3.0 pour compiler et exécuter des applications IBM MQ classes for Jakarta Messaging*

Plateforme	CLASSPATH définition
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.jakarta.client.jar;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jakarta.client.jar;

JMS 2.0

*Tableau 8. Paramètres **CLASSPATH** pour JMS 2.0 pour compiler et exécuter des applications IBM MQ classes for JMS , y compris les exemples d'application*

Plateforme	Paramètre CLASSPATH
AIX AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:

*Tableau 8. Paramètres **CLASSPATH** pour JMS 2.0 pour compiler et exécuter des applications IBM MQ classes for JMS , y compris les exemples d'application (suite)*	
Plateforme	**Paramètre CLASSPATH**
Linux Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mq.allclient.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;
z/OS z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.allclient.jar: *MQ_INSTALLATION_PATH*/java/samples/jms/samples:

Dans ces tableaux, *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Le manifeste du fichier JAR `com.ibm.mq.jakarta.client.jar` ou `com.ibm.mq.allclient.jar` contient des références à la plupart des autres fichiers JAR requis par les applications IBM MQ classes for JMS . Vous n'avez donc pas besoin d'ajouter ces fichiers JAR à votre chemin d'accès aux classes. Ces fichiers JAR incluent ceux requis par les applications qui utilisent l'interface JNDI (Java Naming Directory Interface) pour extraire les objets gérés d'un service d'annuaire et par les applications qui utilisent l'API JTA (Java Transaction API).

Toutefois, vous devez inclure des fichiers JAR supplémentaires dans votre chemin d'accès aux classes dans les cas suivants:

- Si vous utilisez des classes d'exit de canal qui implémentent les interfaces d'exit de canal définies dans le package `com.ibm.mq` , au lieu de celles définies dans le package `com.ibm.mq.exits` , vous devez ajouter le fichier JAR IBM MQ classes for Java , `com.ibm.mq.jar`, à votre chemin d'accès aux classes.
- Si votre application utilise JNDI pour extraire des objets gérés d'un service d'annuaire, vous devez également ajouter les fichiers JAR suivants à votre chemin d'accès aux classes:
  - `fscontext.jar`
  - `providerutil.jar`
- Si votre application utilise JTA, vous devez également ajouter `jta.jar` à votre chemin d'accès aux classes.

Remarque : Ces fichiers JAR supplémentaires sont requis uniquement pour la compilation de vos applications et non pour leur exécution.

Les scripts fournis avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging utilisent les variables d'environnement suivantes:

MQ_JAVA_DATA_PATH
Cette variable d'environnement indique le répertoire de la sortie de journal et de trace.

MQ_JAVA_INSTALL_PATH
Cette variable d'environnement indique le répertoire dans lequel IBM MQ classes for JMS est installé.

MQ_JAVA_LIB_PATH
Cette variable d'environnement indique le répertoire dans lequel les bibliothèques IBM MQ classes for JMS sont stockées, comme indiqué dans les tableaux précédents.

Procédure

- Windows

  Sous Windows, après avoir installé IBM MQ, exécutez la commande **setmqenv**.

  Si vous n'exécutez pas d'abord cette commande, le message d'erreur suivant peut s'afficher lorsque vous émettez une commande **dspmqver** :

  ```
  AMQ8351: IBM MQ n'a pas été configuré
  ou la fonction JRE IBM MQ n'a pas été installée.
  ```

  Remarque : Ce message est à attendre si vous n'avez pas installé IBM MQ Java runtime environment (JRE) (voir Vérification des prérequis pour les fonctions Windows supplémentaires).

- Linux AIX

  Sur les systèmes AIX and Linux , définissez les variables d'environnement vous-même:

  JMS 2.0 Pour JMS 2.0, utilisez l'un des scripts suivants pour définir les variables d'environnement:

  - Si vous utilisez une machine virtuelle Java 32 bits, utilisez le script `setjmsenv`.
  - Si vous utilisez une machine virtuelle Java 64 bits sur un système AIX ou Linux , utilisez le script `setjmsenv64`.

  JM 3.0 Pour Jakarta Messaging 3.0, utilisez l'un des scripts suivants pour définir les variables d'environnement:

  - Si vous utilisez une machine virtuelle Java 32 bits, utilisez le script `setjms30env`.
  - Si vous utilisez une machine virtuelle Java 64 bits, utilisez le script `setjms30env64`.

  Ces scripts se trouvent dans le répertoire *MQ_INSTALLATION_PATH*`/java/bin` , où *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

  Vous pouvez utiliser ces scripts de différentes manières. Vous pouvez utiliser le script comme base pour définir les variables d'environnement requises, comme indiqué dans le tableau, ou les ajouter à `.profile` à l'aide d'un éditeur de texte. Si vous disposez d'une configuration non standard, éditez le contenu du script si nécessaire. Vous pouvez également exécuter le script dans chaque session à partir de laquelle les scripts de démarrage JMS doivent être exécutés. Si vous choisissez cette option, vous devez exécuter le script dans chaque fenêtre shell que vous démarrez, pendant le processus de vérification JMS :

  - JMS 2.0 Pour JMS 2.0, entrez `. ./setjmsenv` ou `../setjmsenv64`.
  - JM 3.0 Pour Jakarta Messaging 3.0, entrez `. ./setjms30env` ou `. ./setjms30env64`.

  IBM i Sous IBM i, vous devez définir la variable d'environnement **QIBM_MULTI_THREADED** sur Y. Vous pouvez ensuite exécuter des applications à unités d'exécution multiples de la même manière que vous exécutez des applications à unités d'exécution uniques. Pour plus d'informations, voir Configuration d' IBM MQ avec Java et JMS.

Tâches associées

«Utilisation des modèles d'application IBM MQ classes for JMS», à la page 124
Les modèles d'application IBM MQ classes for JMS fournissent une présentation des fonctions communes de l'API JMS . Vous pouvez les utiliser pour vérifier la configuration de votre installation et de votre serveur de messagerie et pour vous aider à créer vos propres applications.

Référence associée

«Scripts fournis avec IBM MQ classes for JMS/Jakarta Messaging», à la page 127
Un certain nombre de scripts sont fournis pour vous aider à exécuter les tâches courantes qui doivent être exécutées lors de l'utilisation de IBM MQ classes for JMS et de IBM MQ classes for Jakarta Messaging.

Configuration des bibliothèques JNI (Java Native Interface)
Les applications IBM MQ classes for JMS qui se connectent à un gestionnaire de files d'attente à l'aide du transport de liaisons ou qui se connectent à un gestionnaire de files d'attente à l'aide du transport client et qui utilisent des programmes d'exit de canal écrits dans des langues autres que Javadoivent être exécutées dans un environnement qui permet d'accéder aux bibliothèques JNI (Java Native Interface).

Avant de commencer

Pour plus d'informations sur l'utilisation de l'environnement WebSphere Application Server , voir Configuration du fournisseur de messagerie IBM MQ avec des informations sur les bibliothèques natives .

Pourquoi et quand exécuter cette tâche

Pour configurer cet environnement, vous devez configurer le chemin d'accès à la bibliothèque de l'environnement de sorte que Java Virtual Machine (JVM) puisse charger la bibliothèque mqjbnd avant de démarrer l'application IBM MQ classes for JMS .

IBM MQ fournit deux bibliothèques JNI (Java Native Interface):

mqjbnd
Cette bibliothèque est utilisée par les applications qui se connectent à un gestionnaire de files d'attente à l'aide du transport de liaisons. Il fournit l'interface entre IBM MQ classes for JMS et le gestionnaire de files d'attente. La bibliothèque mqjbnd installée avec IBM MQ 9.4 peut être utilisée pour se connecter à n'importe quel gestionnaire de files d'attente IBM MQ 9.4 (ou version antérieure).

mqjexitstub02
La bibliothèque mqjexitstub02 est chargée par IBM MQ classes for JMS lorsqu'une application se connecte à un gestionnaire de files d'attente à l'aide du transport client et utilise un programme d'exit de canal écrit dans un langage autre que Java.

Sur certaines plateformes, IBM MQ installe les versions 32 bits et 64 bits de ces bibliothèques JNI. L'emplacement des bibliothèques pour chaque plateforme est indiqué dans le Tableau 1.

Tableau 9. Emplacement des bibliothèques IBM MQ classes for JMS pour chaque plateforme

Plateforme	**Répertoire contenant les bibliothèques IBM MQ classes for JMS**
AIX AIX Linux Linux (POWER, x86-64 et zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*\java\lib64 (bibliothèques 64 bits)
z/OS z/OS	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 31 bits et 64 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Remarque : z/OS Sous z/OS, vous pouvez utiliser une machine virtuelle Java Virtual Machine (JVM) 31 bits ou 64 bits. Il n'est pas nécessaire de spécifier les bibliothèques JNI à utiliser ; IBM MQ classes for JMS peut déterminer pour lui-même les bibliothèques JNI à charger.

Procédure

1. Configurez la propriété **java.library.path** de la machine virtuelle Java, qui peut être effectuée de deux manières:

   - En spécifiant l'argument JVM comme illustré dans l'exemple suivant:

     ```
     -Djava.library.path=path_to_library_directory
     ```

     
     Par exemple, pour une machine virtuelle Java 64 bits sur Linux pour une installation d'emplacement par défaut, spécifiez:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - En configurant l'environnement de l'interpréteur de commandes de sorte que la machine virtuelle Java définisse son propre `java.library.path`. Ce chemin varie en fonction de la plateforme et de l'emplacement dans lequel vous avez installé IBM MQ. Par exemple, pour une machine virtuelle Java 64 bits et un emplacement d'installation IBM MQ par défaut, vous pouvez utiliser les paramètres suivants:

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

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

     
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

   Voici un exemple de pile d'exceptions que vous voyez lorsque l'environnement n'a pas été configuré correctement:

   ```
   Causé par: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
   AMQ8598: Echec du chargement de la bibliothèque JNI native WebSphere MQ :'mqjbnd'.
       à l'adresse com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
       dans com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
       à l'adresse java.security.AccessController.doPrivileged(AccessController.java:400)
       à l'adresse com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
       à l'adresse com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
       dans com.ibm.mq.jmqi.local.LocalMQ. < init> (LocalMQ.java:1350)
       à com.ibm.mq.jmqi.local.LocalServer. < init> (LocalServer.java:230)
       at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
       à
   sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

       à
   sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
   a:58)
       sur java.lang.reflect.Constructor.newInstance(Constructor.java:542)
       à l'adresse com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
       dans com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
       à
   com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
   actory.java:8437)
       ... 7 autres
   Causé par: java.lang.UnsatisfiedLinkError: mqjbnd (introuvable dans java.library.path)
       à java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
       à java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
       sur java.lang.System.loadLibrary(System.java:534)
       dans com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
       ... 20 more
   ```

2. Une fois l'environnement 32 bits ou 64 bits configuré, démarrez l'application IBM MQ classes for JMS à l'aide de la commande suivante:

   ```
   java application-name
   ```

   où *application-name* est le nom de l'application IBM MQ classes for JMS à exécuter.

   Une exception contenant le code anomalie IBM MQ 2495 (MQRC_MODULE_NOT_FOUND) est émise par IBM MQ classes for JMS si:

- L'application IBM MQ classes for JMS est exécutée dans un environnement Java runtime environment32 bits et un environnement 64 bits a été configuré pour IBM MQ classes for JMS, car Java runtime environment 32 bits ne peut pas charger la bibliothèque native Java 64 bits.
- L'application IBM MQ classes for JMS est exécutée dans un Java runtime environment64 bits et un environnement 32 bits a été configuré pour IBM MQ classes for JMS, car Java runtime environment 64 bits ne peut pas charger la bibliothèque native Java 32 bits.

Le fichier de configuration IBM MQ classes for JMS/Jakarta Messaging
Les fichiers de configuration IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging spécifient les propriétés utilisées pour configurer IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging.

Remarque : Les propriétés définies dans le fichier de configuration peuvent également être définies en tant que propriétés système JVM. Si une propriété est définie à la fois dans le fichier de configuration et en tant que propriété système, la propriété système est prioritaire. Par conséquent, si nécessaire, vous pouvez remplacer n'importe quelle propriété dans le fichier de configuration en la spécifiant comme propriété système dans la commande **java** .

Le format d'un fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging est celui d'un fichier de propriétés Java standard. Un exemple de fichier de configuration appelé `jms.config` est fourni dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS . Ce fichier documente toutes les propriétés prises en charge et leurs valeurs par défaut.

Vous pouvez choisir le nom et l'emplacement d'un fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Lorsque vous démarrez votre application, utilisez une commande **java** au format suivant:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

Dans la commande, *config_file_url* est une URL qui spécifie le nom et l'emplacement du fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Les URL des types suivants sont prises en charge: http, file, ftp et jar.

Voici un exemple de commande **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config MyAppClass
```

Cette commande identifie le fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging en tant que fichier `D:\mydir\mjms.config` sur le système Windows local.

Lorsqu'une application démarre, IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging lit le contenu du fichier de configuration et stocke les propriétés spécifiées dans un magasin de propriétés interne. Si la commande **java** n'identifie pas de fichier de configuration ou si le fichier de configuration est introuvable, IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging utilise les valeurs par défaut pour toutes les propriétés.

Un fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging peut être utilisé avec tous les transports pris en charge entre une application et un gestionnaire de files d'attente ou un courtier.

Remplacement des propriétés spécifiées dans un fichier de configuration IBM MQ MQI client

Un fichier de configuration IBM MQ MQI client peut également spécifier les propriétés utilisées pour configurer IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging. Toutefois, les propriétés spécifiées dans un fichier de configuration IBM MQ MQI client s'appliquent uniquement lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client.

Si nécessaire, vous pouvez remplacer n'importe quel attribut dans un fichier de configuration IBM MQ MQI client en le spécifiant comme propriété dans un fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Pour remplacer un attribut dans un fichier de configuration IBM

MQ MQI client , utilisez une entrée au format suivant dans le fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Les variables de l'entrée ont les significations suivantes:

section
Nom de la section dans le fichier de configuration IBM MQ MQI client qui contient l'attribut

propName
Nom de l'attribut tel qu'il est spécifié dans le fichier de configuration IBM MQ MQI client

propValue
Valeur de la propriété qui remplace la valeur de l'attribut spécifié dans le fichier de configuration IBM MQ MQI client

Vous pouvez également remplacer un attribut dans un fichier de configuration IBM MQ MQI client en spécifiant la propriété en tant que propriété système dans la commande **java** . Utilisez le format précédent pour spécifier la propriété en tant que propriété système.

Seuls les attributs suivants d'un fichier de configuration IBM MQ MQI client sont pertinents pour IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging. Si vous spécifiez ou remplacez d'autres attributs, cela n'a aucun effet. En particulier, notez que les fichiers `ChannelDefinitionFile` et `ChannelDefinitionDirectory` de la section CHANNELS du fichier de configuration client ne sont pas utilisés. Pour plus d'informations sur l'utilisation de la table de définition de canal du client avec IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290 .

Tableau 10. Quelle section du fichier de configuration du client contient quel attribut

section	**Attribut**
Strophe CHANNELS du fichier de configuration client	Put1DefaultAlwaysSync
Strophe CHANNELS du fichier de configuration client	DefRecon
Strophe CHANNELS du fichier de configuration client	ReconDelay
Strophe CHANNELS du fichier de configuration client	PasswordProtection
ClientExitPath du fichier de configuration du client	ExitsDefaultPath
ClientExitPath du fichier de configuration du client	ExitsDefaultPath64
ClientExitPath du fichier de configuration du client	JavaExitsClasspath
Strophe JMQI du fichier de configuration du client	useMQCSPauthentication
sectionMessageBuffer du fichier de configuration du client	MaximumSize
sectionMessageBuffer du fichier de configuration du client	PurgeTime
sectionMessageBuffer du fichier de configuration du client	UpdatePercentage
Strophe TCP du fichier de configuration du client	ClntRcvBufSize
Strophe TCP du fichier de configuration du client	ClntSndBufSize

Tableau 10. Quelle section du fichier de configuration du client contient quel attribut (suite)	
section	**Attribut**
Strophe TCP du fichier de configuration du client	Connect_Timeout
Strophe TCP du fichier de configuration du client	KeepAlive

Pour plus de détails sur la configuration IBM MQ MQI client , voir le fichier de configuration IBM MQ MQI client , `mqclient.ini`

Utilisation de Java Standard Environment Trace pour configurer la trace JMS
Utilisez la section Java Standard Environment Trace Settings pour configurer la fonction de trace IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputNom*
traceOutputName est le répertoire et le nom de fichier dans lesquels la sortie de trace est envoyée.

Par défaut, les informations de trace sont écrites dans un fichier de trace dans le répertoire de travail en cours de l'application. Le nom du fichier de trace dépend de l'environnement dans lequel l'application s'exécute:

- JM 3.0 Depuis la IBM MQ 9.3.0, si l'application a chargé le fichier IBM MQ classes for Jakarta Messaging à partir du fichier JAR relocalisable `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) ou le fichier IBM MQ classes for JMS à partir du fichier JAR relocalisable `com.ibm.mq.allclient.jar` (JMS 2.0), la trace est écrite dans un fichier appelé `mqjavaclient_%PID%.cl%u.trc`.
- Si l'application a chargé le fichier IBM MQ classes for JMS à partir du fichier JAR relocalisable `com.ibm.mq.allclient.jar`, la trace est écrite dans un fichier appelé `mqjavaclient_%PID%.cl%u.trc`.
- Si l'application a chargé le fichier IBM MQ classes for JMS à partir du fichier JAR `com.ibm.mqjms.jar`, la trace est écrite dans un fichier appelé `mqjava_%PID%.cl%u.trc`.

où *%PID%* est l'identificateur de processus de l'application qui est tracée, et *%u* est un numéro unique permettant de différencier les fichiers entre les unités d'exécution qui exécutent la trace sous différents chargeurs de classe Java.

Si un ID de processus n'est pas disponible, un nombre aléatoire est généré et précédé de la lettre `f`. Pour inclure l'ID de processus dans un nom de fichier que vous spécifiez, utilisez la chaîne `%PID%`.

Si vous spécifiez un autre répertoire, il doit exister et vous devez disposer d'un droit d'accès en écriture pour ce répertoire. Si vous ne disposez pas de droits d'accès en écriture, la sortie de trace est écrite dans `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList est une liste de packages et de classes tracés, ou les valeurs spéciales `ALL` ou `NONE`.

Séparez les noms de package ou de classe par un point-virgule, `;`. *includeList* prend par défaut la valeur `ALL`et trace tous les packages et classes dans IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging.

Remarque : Vous pouvez inclure un package, mais exclure ensuite les sous-packages de ce package. Par exemple, si vous incluez le package `a.b` et le package d'exclusion `a.b.x`, la trace inclut tout ce qui se trouve dans `a.b.y` et `a.b.z`, mais pas dans `a.b.x` ou `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList est une liste de packages et de classes qui ne sont pas tracés, ou les valeurs spéciales `ALL` ou `NONE`.

Séparez les noms de package ou de classe par un point-virgule, `;`. *excludeList* prend la valeur par défaut `NONE`et n'exclut donc pas la trace de packages et de classes dans IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

Remarque : Vous pouvez exclure un package, puis inclure des sous-packages de ce package. Par exemple, si vous excluez le package a.b et incluez le package a.b.x, la trace inclut tout ce qui se trouve dans a.b.x et a.b.x.1, mais pas dans a.b.y ou a.b.z.

Tout package ou classe spécifié, au même niveau, comme inclus et exclu, est inclus.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayOctets*
maxArrayBytes est le nombre maximal d'octets tracés à partir de n'importe quel tableau d'octets.

Si *maxArrayBytes* est défini sur un entier positif, il limite le nombre d'octets dans un tableau d'octets qui sont écrits dans le fichier de trace. Il tronque le tableau d'octets après avoir écrit *maxArrayBytes* . La définition de *maxArrayBytes* réduit la taille du fichier de trace résultant et réduit l'effet de la fonction de trace sur les performances de l'application.

La valeur 0 pour cette propriété signifie qu'aucun contenu d'un tableau d'octets n'est envoyé au fichier de trace.

La valeur par défaut est -1, ce qui supprime toute limite sur le nombre d'octets d'un tableau d'octets envoyés au fichier de trace.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceOctets*
maxTraceBytes est le nombre maximal d'octets écrits dans un fichier de sortie de trace.

maxTraceBytes fonctionne avec *traceCycles*. Si le nombre d'octets de trace écrits est proche de la limite, le fichier est fermé et un nouveau fichier de sortie de trace est démarré.

La valeur 0 signifie qu'un fichier de sortie de trace a une longueur nulle. La valeur par défaut est -1, ce qui signifie que la quantité de données à écrire dans un fichier de sortie de trace est illimitée.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles est le nombre de fichiers de sortie de trace à parcourir.

Si le fichier de sortie de trace en cours atteint la limite spécifiée par *maxTraceBytes*, le fichier est fermé. Une sortie de trace supplémentaire est écrite dans le fichier de sortie de trace suivant dans l'ordre. Chaque fichier de sortie de trace se distingue par un suffixe numérique ajouté au nom de fichier. Le fichier de sortie de trace en cours ou le plus récent est mqjms.trc.0, le prochain fichier de sortie de trace le plus récent est mqjms.trc.1. Les anciens fichiers de trace suivent le même modèle de numérotation jusqu'à la limite.

La valeur par défaut de *traceCycles* est 1. Si *traceCycles* a la valeur 1, lorsque le fichier de sortie de trace en cours atteint sa taille maximale, le fichier est fermé et supprimé. Un nouveau fichier de sortie de trace portant le même nom est démarré. Par conséquent, il n'existe qu'un seul fichier de sortie de trace à la fois.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters contrôle si les paramètres de méthode et les valeurs de retour sont inclus dans la trace.

traceParameters prend par défaut la valeur TRUE. Si *traceParameters* est défini sur FALSE, seules les signatures de méthode sont tracées.

com.ibm.msg.client.commonservices.trace.startup = *démarrage*
Il existe une phase d'initialisation de IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging au cours de laquelle des ressources sont allouées. La fonction de trace principale est initialisée lors de la phase d'allocation de ressources.

Si *startup* est défini sur TRUE, la trace de démarrage est utilisée. Les informations de trace sont générées immédiatement et incluent la configuration de tous les composants, y compris la fonction de trace elle-même. Les informations de trace de démarrage peuvent être utilisées pour diagnostiquer les problèmes de configuration. Les informations de trace de démarrage sont toujours écrites dans System.err.

startup prend par défaut la valeur FALSE.

startup est vérifié avant la fin de l'initialisation. Pour cette raison, spécifiez uniquement la propriété sur la ligne de commande en tant que propriété système Java . Ne le spécifiez pas dans le fichier de configuration IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Définissez *compressedTrace* sur TRUE pour compresser la sortie de trace.

La valeur par défaut de *compressedTrace* est FALSE.

Si *compressedTrace* est défini sur TRUE, la sortie de trace est compressée. Le nom du fichier de sortie de trace par défaut porte l'extension `.trz`. Si la compression est définie sur FALSE, la valeur par défaut, le fichier a l'extension `.trc` pour indiquer qu'il est décompressé. Toutefois, si le nom de fichier de la sortie de trace a été spécifié dans *traceOutputName* , ce nom est utilisé à la place ; aucun suffixe n'est appliqué au fichier.

La sortie de trace compressée est plus petite que la sortie non compressée. Etant donné qu'il y a moins d'E-S, il peut être écrit plus rapidement que la trace non compressée. La fonction de trace compressée a moins d'effet sur les performances de IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging que la fonction de trace non compressée.

Si *maxTraceBytes* et *traceCycles* sont définis, plusieurs fichiers de trace compressés sont créés à la place de plusieurs fichiers à plat.

Si IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging se termine de manière non contrôlée, il se peut qu'un fichier de trace compressé ne soit pas valide. Pour cette raison, la compression de trace ne doit être utilisée que lorsque IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging se ferme de manière contrôlée. N'utilisez la compression de trace que si les problèmes examinés ne provoquent pas l'arrêt inattendu de la machine virtuelle Java elle-même. N'utilisez pas la compression de trace pour diagnostiquer les problèmes qui peuvent entraîner des arrêts de la machine virtuelle `System.Halt()` ou des arrêts anormaux et non contrôlés de la machine virtuelle Java.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel indique un niveau de filtrage pour la trace. Les niveaux de trace définis sont les suivants:

- `TRACE_NONE`: `0`
- `TRACE_EXCEPTION`: `1`
- `TRACE_WARNING`: `3`
- `TRACE_INFO`: `6`
- `TRACE_ENTRYEXIT`: `8`
- `TRACE_DATA`: `9`
- `TRACE_ALL`: `Integer.MAX_VALUE`

Chaque niveau de trace inclut tous les niveaux inférieurs. Par exemple, si le niveau de trace est défini sur `TRACE_INFO`, tout point de trace dont le niveau défini est `TRACE_EXCEPTION`, `TRACE_WARNING`ou `TRACE_INFO` est consigné dans la trace. Tous les autres points de trace sont exclus.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace contrôle si le service de traçage du client IBM MQ JMS est utilisé dans un environnement WebSphere Application Server .

Si *standaloneTrace* est défini sur TRUE, les propriétés de trace du client IBM MQ JMS sont utilisées pour déterminer la configuration de la trace.

Si *standaloneTrace* est défini sur FALSEet que le client IBM MQ JMS s'exécute dans un conteneur WebSphere Application Server , le service de trace WebSphere Application Server est utilisé. Les informations de trace générées dépendent des paramètres de trace du serveur d'applications.

La valeur par défaut de *standaloneTrace* est FALSE.

Section de journalisation
Utilisez la section Journalisation pour configurer la fonction de journal IBM MQ classes for JMS .

Les propriétés suivantes peuvent être incluses dans la section Journalisation:

com.ibm.msg.client.commonservices.log.outputName = chemin
Nom du fichier journal utilisé par la fonction de consignation IBM MQ classes for JMS . La valeur par défaut est `mqjms.log`, qui est écrite dans le répertoire de travail en cours de l'environnement d'exécution Java dans lequel s'exécute IBM MQ classes for JMS .

La propriété peut prendre l'une des valeurs suivantes:

- un nom de chemin unique
- une liste de noms de chemin séparés par des virgules (toutes les données sont consignées dans tous les fichiers)

Chaque nom de chemin peut être un nom de chemin absolu ou relatif ou:

"stderr" ou "System.err"
Représente le flux d'erreurs standard.

"stdout" ou "System.out"
Représente le flux de sortie standard.

com.ibm.msg.client.commonservices.log.maxBytes
Nombre maximal d'octets consignés à partir de n'importe quel appel pour consigner les données de message.

Entier positif
Les données sont écrites jusqu'à cette valeur d'octets par appel de journal.

0
Aucune donnée n'est écrite.

-1
Des données illimitées sont écrites (valeur par défaut).

com.ibm.msg.client.commonservices.log.limit
Nombre maximal d'octets écrits dans un fichier journal (la valeur par défaut est 262144).

Entier positif
Les données sont écrites jusqu'à cette valeur d'octets par fichier journal.

0
Aucune donnée n'est écrite.

-1
Des données illimitées sont écrites.

com.ibm.msg.client.commonservices.log.count
Nombre de fichiers journaux à parcourir. Lorsque chaque fichier atteint la trace `com.ibm.msg.client.commonservices.trace.limit` commence dans le fichier suivant, la valeur par défaut est 3.

Entier positif
Nombre de fichiers à parcourir.

0
Un seul fichier.

Section Java SE Specifics
Utilisez la strophe Java SE Specifics pour configurer les propriétés utilisées lorsque les IBM MQ classes for JMS sont utilisées dans un environnement Java Standard Edition .

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE|FALSE
Détermine si un fichier JavaCore est écrit immédiatement après la génération d'un fichier FDC par IBM MQ classes for JMS . S'il est défini sur TRUE, un fichier JavaCore est généré dans le répertoire de travail de l'environnement d'exécution Java dans lequel les IBM MQ classes for JMS s'exécutent.

true
Générez JavaCore, en fonction de la capacité de l'environnement d'exécution Java à le faire.

faux
Ne générez pas JavaCore ; il s'agit de la valeur par défaut.

Section Propriétés IBM MQ
Utilisez la strophe IBM MQ Properties pour définir les propriétés qui affectent la manière dont IBM MQ classes for JMS interagit avec IBM MQ.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Lorsqu'une application qui utilise IBM MQ classes for JMS se connecte à un gestionnaire de files d'attente IBM MQ à l'aide du mode de migration du fournisseur de messagerie IBM MQ , IBM MQ classes for JMS utilise une taille de mémoire tampon par défaut de 4 Ko lorsqu'il reçoit des messages. Si le message que l'application tente d'obtenir est supérieur à 4 Ko, IBM MQ classes for JMS redimensionne la mémoire tampon pour qu'elle soit suffisamment grande pour contenir le message. La taille de mémoire tampon la plus importante est ensuite utilisée lors de la réception des messages suivants.

Cette propriété contrôle quand la taille de la mémoire tampon est réduite à 4 Ko. Par défaut, lorsque dix messages consécutifs inférieurs à la taille de la mémoire tampon supérieure sont reçus, la taille de la mémoire tampon est réduite à 4 Ko. Pour réinitialiser la taille de la mémoire tampon à 4 Ko chaque fois qu'un message est reçu, définissez la propriété sur la valeur 0.

0

La mémoire tampon est toujours réinitialisée à la taille par défaut.

10

Il s'agit de la valeur par défaut. La mémoire tampon sera redimensionnée après le dixième message.

com.ibm.msg.client.wmq.receiveConversionCCSID
Lorsqu'une application qui utilise IBM MQ classes for JMS se connecte à un gestionnaire de files d'attente IBM MQ à l'aide du mode normal du fournisseur de messagerie IBM MQ , la propriété `receiveConversionCCSID` peut être définie pour remplacer la valeur de CCSID par défaut dans la structure MQMD utilisée pour recevoir les messages du gestionnaire de files d'attente. Par défaut, le MQMD contient une zone CCSID définie sur 1208, mais cette valeur peut être modifiée si, par exemple, le gestionnaire de files d'attente ne parvient pas à convertir des messages dans cette page de codes.

Les valeurs admises sont tout numéro de CCSID valide ou l'une des valeurs suivantes:

-1
Utilisez la valeur par défaut de la plateforme.

1208
Il s'agit de la valeur par défaut.

Section spécifique au mode client
Utilisez la section spécifique au mode client pour spécifier les propriétés qui sont utilisées lorsque IBM MQ classes for JMS se connecte à un gestionnaire de files d'attente qui utilise le transport CLIENT.

com.ibm.mq.polling.RemoteRequestEntry
Indique l'intervalle d'interrogation utilisé par IBM MQ classes for JMS pour rechercher les connexions rompues lorsqu'il attend une réponse d'un gestionnaire de files d'attente.

Entier positif
Nombre de millisecondes à attendre avant la vérification. La valeur par défaut est 10000 ou 10 secondes. La valeur minimale est 3000 et les valeurs inférieures sont traitées de la même manière que cette valeur minimale.

Propriétés utilisées pour configurer le comportement du client JMS
Utilisez ces propriétés pour configurer le comportement du client JMS .

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE
La spécification JMS 2.0 introduit des modifications dans le mode de fonctionnement de certains comportements. IBM MQ 8.0 inclut la propriété `com.ibm.mq.jms.SupportMQExtensions`, qui peut être définie sur *TRUE*, pour rétablir ces comportements modifiés dans les implémentations précédentes. L'annulation des comportements modifiés peut s'avérer nécessaire pour les applications JMS 2.0 , ainsi que pour certaines applications qui utilisent l'API JMS 1.1 mais s'exécutent sur le IBM MQ 8.0 IBM MQ classes for JMS.

exit utilisateur associé à une tâche
Les trois zones de fonctionnalité suivantes sont rétablies en définissant SupportMQExtensions sur TRUE:

Priorité de message
Une priorité peut être affectée aux messages, *0* - *9*. Avant JMS 2.0, les messages pouvaient également utiliser la valeur *-1*, indiquant que la priorité par défaut d'une file d'attente était utilisée. JMS 2.0 ne permet pas de définir une priorité de message de *-1* . L'activation de `SupportMQExtensions` permet d'utiliser la valeur *-1* .

ID de client
La spécification JMS 2.0 requiert que les ID client non nuls soient vérifiés pour leur unicité lorsqu'ils effectuent une connexion. L'activation de `SupportMQExtensions`signifie que cette exigence est ignorée et qu'un ID client peut être réutilisé.

`NoLocal`
La spécification JMS 2.0 requiert que lorsque cette constante est activée, un destinataire ne puisse pas recevoir de messages publiés par le même ID client. Avant JMS 2.0, cet attribut était défini sur un abonné pour l'empêcher de recevoir des messages publiés par sa propre connexion. L'activation de `SupportMQExtensions` rétablit ce comportement dans son implémentation précédente.

FALSE
Les changements de comportement sont conservés.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
Depuis IBM MQ 8.0.0 Fix Pack 2, une fois qu'une application a envoyé un message Bytes ou Stream, IBM MQ classes for JMS peut définir l'état du message qui vient d'être envoyé en lecture seule ou en écriture seule.

exit utilisateur associé à une tâche
Les objets sont définis en lecture seule après leur envoi. La définition de cette valeur gère la compatibilité avec la spécification JMS 2.0

FALSE
Les objets sont définis pour l'écriture uniquement après leur envoi. Il s'agit de la valeur par défaut.

Concepts associés
«Propriété SupportMQExtensions», à la page 336
La spécification JMS 2.0 a introduit des modifications dans le mode de fonctionnement de certains comportements. IBM MQ 8.0 et les versions ultérieures incluent la propriété **com.ibm.mq.jms.SupportMQExtensions**, qui peut être définie sur TRUE pour rétablir ces comportements modifiés dans les implémentations précédentes.

STEPLIB configuration for IBM MQ classes for JMS on z/OS
On z/OS, the STEPLIB used at run time must contain the IBM MQ SCSQAUTH and SCSQANLE libraries. Specify these libraries in the startup JCL or using the `.profile` file.

From z/OS UNIX System Services, you can add these using a line in your `.profile` as shown in the following code snippet, replacing `thlqual` with the high-level data set qualifier that you chose when installing IBM MQ:

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

In other environments, you typically need to edit the startup JCL to include SCSQAUTH and SCSQANLE on the STEPLIB concatenation:

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

IBM MQ classes for JMS et outils de gestion de logiciels
Les outils de gestion de logiciels tels que Apache Maven peuvent être utilisés avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging.

De nombreuses grandes organisations de développement utilisent ces outils pour gérer de manière centralisée les référentiels de bibliothèques tierces.

Les fichiers IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont composés d'un certain nombre de fichiers JAR. Lorsque vous développez des applications en langage Java à l'aide de cette API, une installation d'un serveur IBM MQ , d'un client IBM MQ ou d'un IBM MQ client SupportPac est requise sur la machine sur laquelle l'application est développée.

Si vous souhaitez utiliser un tel outil et ajouter les fichiers JAR qui constituent le IBM MQ classes for JMS à un référentiel géré de manière centralisée, les points suivants doivent être observés:

- Un référentiel ou un conteneur ne doit être mis à la disposition que des développeurs de votre organisation. Toute distribution en dehors de l'organisation n'est pas autorisée.
- Le référentiel doit contenir un ensemble complet et cohérent de fichiers JAR provenant d'une seule édition ou d'un seul groupe de correctifs IBM MQ .
- Vous êtes responsable de la mise à jour du référentiel avec toute maintenance fournie par le support IBM .

Les fichiers JAR suivants doivent être installés dans le référentiel:

- JMS 2.0 `com.ibm.mq.allclient.jar` et `jms.jar` sont requis si vous utilisez IBM MQ classes for JMS.
- JM 3.0 `com.ibm.mq.jakarta.client.jar` et `jakarta.jms-api.jar` sont requis si vous utilisez IBM MQ classes for Jakarta Messaging.
- `fscontext.jar` est obligatoire si vous utilisez IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging et que vous accédez à des objets gérés JMS qui sont stockés dans un contexte JNDI de système de fichiers.
- `providerutil.jar` est requis si vous utilisez IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging et que vous accédez à des objets gérés JMS qui sont stockés dans un contexte JNDI de système de fichiers.
- Le fournisseur de sécurité Bouncy Castle et les fichiers JAR de support CMS sont requis pour la prise en charge des environnements d'exécution Java nonIBM . Pour plus d'informations, voir Prise en charge des JRE non-IBM.

Exécution d'applications IBM MQ classes for JMS sous Java security manager

IBM MQ classes for JMS peut s'exécuter avec Java security manager activé. Pour exécuter des applications avec le Java security manager activé, vous devez configurer votre Java Virtual Machine (JVM) avec un fichier de configuration de règles approprié.

Le moyen le plus simple de créer un fichier de définition de règle approprié consiste à modifier le fichier de configuration de règle fourni avec votre environnement d'exécution Java (Java runtime environment). Sur la plupart des systèmes, ce fichier se trouve dans le répertoire `lib/security/java.policy` relatif

à votre répertoire JRE. Vous pouvez éditer le fichier de configuration des règles à l'aide de l'éditeur de votre choix ou à l'aide du programme d'outil de règles fourni avec votre environnement d'exécution Java.

Exemple de fichier de configuration de règles

Voici un exemple de fichier de configuration de règles qui permet à IBM MQ classes for JMS de s'exécuter correctement sous le gestionnaire de sécurité par défaut. Ce fichier doit être personnalisé pour spécifier les emplacements de certains fichiers et répertoires: *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé, *MQ_DATA_DIRECTORY* l'emplacement du répertoire de données MQ et *QM_NAME* le nom du gestionnaire de files d'attente pour lequel l'accès est configuré.

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/*" {
  //We need access to these properties, mainly for tracing
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  permission java.util.PropertyPermission "path.separator","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
  permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
  permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
  permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

  //Tracing - we need the ability to control java.util.logging
  permission java.util.logging.LoggingPermission "control";
  // And access to create the trace file and read the log file - assumed to be in the current
directory
  permission java.io.FilePermission "*","read,write";

  // We'd like to set up an mBean to control trace
  permission javax.management.MBeanServerPermission "createMBeanServer";
  permission javax.management.MBeanPermission "*","*";

  // We need to be able to read manifests etc from the jar files in the installation directory
  permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

  //For the client transport type.
  permission java.net.SocketPermission "*","connect,resolve";

  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";

  //For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

  //For applications that use User Exits
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
  permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
  permission java.lang.RuntimePermission "createClassLoader";

  //Required for the z/OS platform
  permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

  // Used by the internal ConnectionFactory implementation
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

  // Used for controlled class loading
  permission java.lang.RuntimePermission "setContextClassLoader";

  // Used to default the Application name in Client mode connections
  permission java.util.PropertyPermission "sun.java.command","read";

  // Used by the IBM JSSE classes
  permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

  //Required to determine if an IBM Java Runtime is running in FIPS mode,
  //and to modify the property values status as required.
  permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
  permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
  //Required if an IBM FIPS provider is to be used for SSL communication.
```

```
  permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

  // Required for non-IBM Java Runtimes that establish secure client
  // transport mode connections using mutual TLS authentication
  permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
  permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";
};
```

Dans l'exemple, l'instruction `grant` contient les droits requis par IBM MQ classes for JMS. Pour utiliser ces instructions d'octroi dans votre fichier de configuration de règles, vous devrez peut-être modifier les noms de chemin en fonction de l'emplacement où vous avez installé IBM MQ classes for JMS et où vous stockez vos applications.

Les modèles d'application fournis avec IBM MQ classes for JMS, ainsi que les scripts permettant de les exécuter, n'activent pas le gestionnaire de sécurité.

Important :

La fonction de trace IBM MQ classes for JMS requiert des droits d'accès supplémentaires car elle effectue des requêtes supplémentaires sur les propriétés système, ainsi que des opérations supplémentaires sur le système de fichiers.

Un modèle de fichier de règles de sécurité adapté à l'exécution sous un gestionnaire de sécurité avec la fonction de trace activée est fourni dans le répertoire `samples/wmqjava` de l'installation IBM MQ en tant que `example.security.policy`.

Configuration post-installation pour les applications IBM MQ classes for JMS

Cette rubrique vous indique les droits dont les applications IBM MQ classes for JMS ont besoin pour accéder aux ressources d'un gestionnaire de files d'attente. Il introduit également des modes de connexion et explique comment configurer un gestionnaire de files d'attente pour que les applications puissent se connecter en mode client.

N'oubliez pas de vérifier le fichier Readme de IBM MQ . Il peut contenir des informations qui remplacent celles de cette rubrique.

Objets utilisés par JMS qui nécessitent une autorisation pour les utilisateurs non privilégiés
Les utilisateurs non privilégiés doivent être autorisés à accéder aux files d'attente utilisées par JMS. Chaque application JMS a besoin d'une autorisation sur le gestionnaire de files d'attente avec lequel elle fonctionne.

Pour plus de détails sur le contrôle d'accès dans IBM MQ, voir Configuration de la sécurité.

Les applications IBM MQ classes for JMS ont besoin des droits `connect` et `inq` sur le gestionnaire de files d'attente. Vous pouvez définir les autorisations appropriées à l'aide de la commande de contrôle **setmqaut** , par exemple:

```
setmqaut -m QM1 -t qmgr -g jmsappsgroup +connect +inq
```

Pour le domaine point à point, les droits suivants sont requis:

- Les files d'attente utilisées par les objets MessageProducer doivent disposer des droits `put` .
- Les files d'attente utilisées par les objets MessageConsumer et QueueBrowser ont besoin des droits `get`, `inq`et `browse` .
- La méthode QueueSession.createTemporaryQueue () doit accéder à la file d'attente modèle spécifiée par la propriété TEMPMODEL de l'objet de fabrique QueueConnection. Par défaut, cette file d'attente modèle est SYSTEM.TEMP.MODEL.QUEUE.

Si l'une de ces files d'attente est une file d'attente alias, les files d'attente cible doivent disposer du droit d'interrogation. Si la file d'attente cible est une file d'attente de cluster, elle requiert également le droit de consultation.

Pour le domaine de publication / abonnement, les files d'attente suivantes sont utilisées si IBM MQ classes for JMS se connecte à un gestionnaire de files d'attente IBM MQ en mode de migration du fournisseur de messagerie IBM MQ :

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Pour plus d'informations sur le mode de migration du fournisseur de messagerie IBM MQ , voir Configuration de la propriété JMS **PROVIDERVERSION**

En outre, si IBM MQ classes for JMS se connecte à un gestionnaire de files d'attente dans ce mode, toute application qui publie des messages doit accéder à la file d'attente de flux spécifiée par la fabrique TopicConnectionou l'objet de rubrique. Par défaut, cette file d'attente est SYSTEM.BROKER.DEFAULT.STREAM.

Si vous utilisez ConnectionConsumer, IBM MQ Resource Adapter ou le fournisseur de messagerie WebSphere Application Server IBM MQ , une autorisation supplémentaire peut être nécessaire.

Les files d'attente qui doivent être lues par ConnectionConsumer doivent disposer des droits `get`, `inqet` `browse` . La file d'attente de rebut du système, ainsi que toute file d'attente de remise en file d'attente ou de rapport utilisée par ConnectionConsumer , doivent disposer des droits `put` et `passall` .

Lorsqu'une application utilise le mode normal du fournisseur de messagerie IBM MQ pour exécuter la messagerie de publication / abonnement, elle utilise la fonctionnalité de publication / abonnement intégrée fournie par le gestionnaire de files d'attente. Pour plus d'informations sur la sécurisation des rubriques et des files d'attente utilisées, voir Sécurité de publication / abonnement .

Modes de connexion pour IBM MQ classes for JMS
Une application IBM MQ classes for JMS peut se connecter à un gestionnaire de files d'attente en mode client ou en mode liaisons. En mode client, IBM MQ classes for JMS se connecte au gestionnaire de files d'attente via TCP/IP. En mode liaisons, IBM MQ classes for JMS se connecte directement au gestionnaire de files d'attente à l'aide de l'interface JNI (Java Native Interface).

Surz/OS , le mode liaisons peut être utilisé dans n'importe quel environnement, mais le mode client ne peut être utilisé que dans les environnements suivants :

- DansWebSphere Application Server ouWebSphere Liberty Profil se connectant à n'importe quel gestionnaire de files d'attente, sur n'importe quelle plateforme, y comprisz/OS .
- Dans les environnements batch lors de la connexion à unIBM MQ for z/OS gestionnaire de files d'attente, s'exécutant sur n'importe quelle partition logique.

Une application s'exécutant sur une autre plateforme peut se connecter à un gestionnaire de files d'attente en mode liaisons ou client.

Vous pouvez utiliser la version en cours ou toute version antérieure prise en charge d' IBM MQ classes for JMS avec un gestionnaire de files d'attente en cours, et vous pouvez utiliser une version en cours ou antérieure prise en charge du gestionnaire de files d'attente avec la version en cours d' IBM MQ classes for JMS. Si vous mélangez des versions différentes, la fonction est limitée au niveau de la version antérieure.

Les sections suivantes décrivent plus en détail chacun des modes de connexion.

Mode client

Pour se connecter à un gestionnaire de files d'attente en mode client, une application IBM MQ classes for JMS peut s'exécuter sur le même système que celui sur lequel le gestionnaire de files d'attente s'exécute

ou sur un autre système. Dans chaque cas, IBM MQ classes for JMS se connecte au gestionnaire de files d'attente via TCP/IP.

Mode liaison

Pour se connecter à un gestionnaire de files d'attente en mode liaisons, une application IBM MQ classes for JMS doit s'exécuter sur le même système que celui sur lequel le gestionnaire de files d'attente s'exécute.

IBM MQ classes for JMS se connecte directement au gestionnaire de files d'attente à l'aide de l'interface JNI (Java Native Interface). Pour utiliser le transport de liaisons, IBM MQ classes for JMS doit être exécuté dans un environnement ayant accès aux bibliothèques IBM MQ Java Native Interface ; voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 100 pour plus d'informations.

IBM MQ classes for JMS prend en charge les valeurs suivantes pour *ConnectOption*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_LIEN_PARTAGE
- MQCNO_LIEN_ISOLÉ_LIAISON
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Pour modifier les options de connexion utilisées par IBM MQ classes for JMS, modifiez la propriété de fabrique de connexions CONNOPT.

Pour plus d'informations sur les options de connexion, voir «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 755

Pour utiliser le transport de liaisons, l'environnement d'exécution Java utilisé doit prendre en charge l'ID de jeu de caractères codés (CCSID) du gestionnaire de files d'attente auquel IBM MQ classes for JMS se connecte.

Pour plus de détails sur la manière de déterminer les CCSID pris en charge par un environnement d'exécution Java , voir IBM MQ FDC avec l'ID sonde 21 généré lors de l'utilisation des classes IBM MQ V7 pour Java ou IBM MQ V7 pour JMS.

Configuration de votre gestionnaire de files d'attente pour que les applications IBM MQ classes for JMS puissent se connecter en mode client
Pour configurer votre gestionnaire de files d'attente afin que les applications IBM MQ classes for JMS puissent se connecter en mode client, vous devez créer une définition de canal de connexion serveur et démarrer un programme d'écoute.

Création d'une définition de canal de connexion serveur

Sur toutes les plateformes, vous pouvez utiliser la commande MQSC DEFINE CHANNEL pour créer une définition de canal de connexion serveur. Prenons cet exemple :

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i Sous IBM i, vous pouvez utiliser la commande CL CRTMQMCHL à la place, comme dans l'exemple suivant:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN)
TRPTYPE(*TCP)
MQMNAME(QMGRNAME)
```

Dans cette commande, *QMGRNAME* est le nom de votre gestionnaire de files d'attente.

Windows Linux Sous Linux et Windows, vous pouvez également créer une définition de canal de connexion serveur à l'aide de IBM MQ Explorer.

z/OS Sous z/OS , vous pouvez utiliser les panneaux d'opérations et de contrôle pour créer une définition de canal de connexion serveur.

Nom du canal (JAVA.CHANNEL dans les exemples précédents) doit être identique au nom de canal spécifié par la propriété CHANNEL de la fabrique de connexions utilisée par votre application pour se connecter au gestionnaire de files d'attente. La valeur par défaut de la propriété CHANNEL est SYSTEM.DEF.SVRCONN.

Démarrage d'un programme d'écoute

Vous devez démarrer un programme d'écoute pour votre gestionnaire de files d'attente si aucun n'est déjà démarré.

Multi Sous Multiplateformes, vous pouvez utiliser la commande MQSC START LISTENER pour démarrer un programme d'écoute après avoir créé un objet programme d'écoute à l'aide de la commande MQSC DEFINE LISTENER, comme illustré dans l'exemple suivant:

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

z/OS Sous z/OS, vous utilisez uniquement la commande START LISTENER, comme dans l'exemple suivant, mais notez que l'espace adresse de l'initiateur de canal doit être démarré avant de pouvoir démarrer un programme d'écoute:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i Sous IBM i, vous pouvez également utiliser la commande CL STRMQMLSR pour démarrer un programme d'écoute, comme dans l'exemple suivant:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

Dans cette commande, *QMGRNAME* est le nom de votre gestionnaire de files d'attente.

ALW Sous AIX, Linux, and Windows, vous pouvez également utiliser la commande de contrôle **runmqlsr** pour démarrer un programme d'écoute, comme dans l'exemple suivant:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

Dans cette commande, *QMgrName* est le nom de votre gestionnaire de files d'attente.

Windows Linux Sous Linux et Windows, vous pouvez également démarrer un programme d'écoute à l'aide de IBM MQ Explorer.

z/OS Sous z/OS, vous pouvez également utiliser les panneaux d'opérations et de contrôle pour démarrer un programme d'écoute.

Le numéro du port sur lequel le programme d'écoute est à l'écoute doit être identique au numéro de port spécifié par la propriété PORT de la fabrique de connexions que votre application utilise pour se connecter au gestionnaire de files d'attente. La valeur par défaut de la propriété PORT est 1414.

L'outil IVT point-à-point pour IBM MQ classes for JMS

Un programme de test de vérification de l'installation point à point (IVT) est fourni avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging. Le programme se connecte à un gestionnaire de files d'attente en mode liaison ou client et envoie un message à la file d'attente appelée

SYSTEM.DEFAULT.LOCAL.QUEUE, puis reçoit le message de la file d'attente. Le programme peut créer et configurer tous les objets dont il a besoin dynamiquement lors de l'exécution ou utiliser JNDI pour extraire des objets gérés d'un service d'annuaire.

Exécutez le test de vérification de l'installation sans utiliser JNDI au préalable car le test est autonome et ne nécessite pas l'utilisation d'un service d'annuaire. Pour obtenir une description des objets gérés, voir Configuration des objets JMS à l'aide de l'outil d'administration.

Test de vérification de l'installation point à point sans utiliser JNDI

Dans ce test, le programme IVT crée et configure tous les objets dont il a besoin dynamiquement lors de l'exécution et n'utilise pas JNDI.

Multi Sur les plateformes multiples, un script est fourni pour exécuter le programme IVT. Le script est appelé **IVTRun** sur les systèmes AIX and Linux et **IVTRun.bat** sur Windows. le script se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS . Le chemin d'accès aux classes doit contenir `com.ibm.mqjms.jar`.

Pour exécuter le test en mode liaisons, entrez la commande suivante:

```
IVTRun -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Pour exécuter le test en mode client, configurez d'abord le gestionnaire de files d'attente comme décrit dans «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098. Notez que le canal à utiliser est par défaut **SYSTEM.DEF.SVRCONN** et que la file d'attente à utiliser est **SYSTEM.DEFAULT.LOCAL.QUEUE**, puis entrez la commande suivante:

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

z/OS Aucun script équivalent n'est fourni sur les systèmes z/OS . A la place, vous exécutez le test IVT en mode liaisons en appelant directement la classe Java . Sous z/OS, vous avez le choix entre deux instances fonctionnellement identiques du programme IVT:

- `com.ibm.mq.jms.MQJMSIVT`, qui est disponible avec IBM MQ classes for JMS (JMS 2.0). Pour utiliser ce programme, le chemin d'accès aux classes doit contenir `com.ibm.mqjms.jar` ou `com.ibm.mq.allclient.jar`.
- `com.ibm.mq.jakarta.jms.MQJMSIVT`, qui est disponible avec IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0). Pour utiliser ce programme, le chemin d'accès aux classes doit contenir `com.ibm.mq.jakarta.client.jar`.

Pour exécuter le test en mode liaisons sous z/OS, entrez la commande suivante:

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

Les paramètres des commandes ont les significations suivantes:

-m *gestionnaire_files_attente*
Nom du gestionnaire de files d'attente auquel le programme IVT se connecte. Si vous exécutez le test en mode liaisons et omettez ce paramètre, le programme IVT se connecte au gestionnaire de files d'attente par défaut.

-host *nom_hôte*
Nom d'hôte ou adresse IP du système sur lequel le gestionnaire de files d'attente s'exécute.

-port *port*
Numéro du port sur lequel le programme d'écoute du gestionnaire de files d'attente est en mode écoute. La valeur par défaut est 1414.

-channel ***canal***
Nom du canal MQI utilisé par le programme IVT pour se connecter au gestionnaire de files d'attente. La valeur par défaut est SYSTEM.DEF.SVRCONN.

-v ***providerVersion***
Niveau d'édition du gestionnaire de files d'attente auquel le programme IVT s'attend à se connecter.

Ce paramètre est utilisé pour définir la propriété PROVIDERVERSION d'un objet de fabrique MQQueueConnectionet possède les mêmes valeurs valides que celles de la propriété PROVIDERVERSION. Pour plus d'informations sur ce paramètre, y compris ses valeurs valides, voir JMS: modifications apportées à la propriété PROVIDERVERSION et la description de la propriété PROVIDERVERSION dans Propriétés des objets IBM MQ classes for JMS.

La valeur par défaut est unspecified.

-ccsid ***ccsid***
Identificateur (CCSID) du jeu de caractères codés, ou page de codes, à utiliser par la connexion. La valeur par défaut est 819.

-t
La fonction de trace est activée. Par défaut, la fonction de trace est désactivée.

Un test réussi produit une sortie similaire à l'exemple de sortie suivant:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
WebSphere MQ classes for Java(tm) Message Service 7.0
Installation Verification Test

Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d627720202020202001edb14620005e03
JMSTimestamp:   1187170264000
JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 28
JMSXAppID: IBM MQ Client for Java
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_PutTime: 09310400
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Test de vérification de l'installation point à point à l'aide de JNDI

Dans ce test, le programme IVT utilise JNDI pour extraire des objets gérés d'un service d'annuaire.

Avant de pouvoir exécuter le test, vous devez configurer un service d'annuaire basé sur un serveur LDAP (Lightweight Directory Access Protocol) ou sur le système de fichiers local. Vous devez également configurer l'outil d'administration d' IBM MQ JMS afin qu'il puisse utiliser le service d'annuaire pour stocker des objets gérés. Pour plus d'informations sur ces prérequis, voir «Prérequis pour IBM MQ classes for JMS», à la page 91. Pour plus d'informations sur la configuration de l'outil d'administration d' IBM MQ JMS , voir Configuration de l'outil d'administration d' JMS.

Le programme IVT doit pouvoir utiliser JNDI pour extraire un objet de fabrique MQQueueConnectionet un objet MQQueue du service d'annuaire. Un script est fourni pour créer ces objets gérés pour vous. Le script est appelé IVTSetup sur les systèmes AIX and Linux et IVTSetup.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS . Pour exécuter le script, entrez la commande suivante:

```
IVTSetup
```

Le script appelle l'outil d'administration IBM MQ JMS pour créer les objets gérés.

L'objet de fabrique MQQueueConnectionest lié avec le nom ivtQCF et est créé avec les valeurs par défaut de toutes ses propriétés, ce qui signifie que le programme IVT s'exécute en mode liaisons et se connecte au gestionnaire de files d'attente par défaut. Si vous souhaitez que le programme IVT s'exécute en mode client ou que vous vous connectez à un gestionnaire de files d'attente autre que le gestionnaire de files d'attente par défaut, vous devez utiliser l'outil d'administration IBM MQ JMS ou IBM MQ Explorer pour modifier les propriétés appropriées de l'objet de fabrique MQQueueConnection. Pour plus d'informations sur l'utilisation de l'outil d'administration d' IBM MQ Explorer JMS , voir Configuration des objets JMS à l'aide de l'outil d'administration. Pour plus d'informations sur l'utilisation de IBM MQ Explorer, voir Introduction à IBM MQ Explorer ou l'aide fournie avec IBM MQ Explorer.

L'objet MQQueue est lié avec le nom ivtQ et est créé avec les valeurs par défaut pour toutes ses propriétés, à l'exception de la propriété QUEUE, qui a la valeur SYSTEM.DEFAULT.LOCAL.QUEUE.

Une fois que vous avez créé les objets gérés, vous pouvez exécuter le programme IVT. Pour exécuter le test à l'aide de JNDI, entrez la commande suivante:

```
IVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Les paramètres de la commande ont la signification suivante:

-url "*providerURL*"
Adresse URL du service d'annuaire. L'URL peut avoir l'un des formats suivants:

- `ldap://hostname/contextName` , pour un service d'annuaire basé sur un serveur LDAP
- `file:/directoryPath` , pour un service d'annuaire basé sur le système de fichiers local

Notez que vous devez placer l'URL entre guillemets (").

-icf *initCtxFait*
Nom de classe de la fabrique de contexte initial, qui doit être l'une des valeurs suivantes:

- `com.sun.jndi.ldap.LdapCtxFactory`, pour un service d'annuaire basé sur un serveur LDAP. Il s'agit de la valeur par défaut.
- `com.sun.jndi.fscontext.RefFSContextFactory`, pour un service d'annuaire basé sur le système de fichiers local.

-t
La fonction de trace est activée. Par défaut, la fonction de trace est désactivée.

Un test réussi produit une sortie similaire à celle d'un test réussi sans utiliser JNDI. La principale différence est que la sortie indique que le test utilise JNDI pour extraire un objet de fabrique MQQueueConnectionet un objet MQQueue.

Bien que cela ne soit pas strictement nécessaire, il est recommandé de mettre en ordre après le test en supprimant les objets gérés créés par le script IVTSetup. Un script est fourni à cette fin. Le script est appelé IVTTidy sur les systèmes AIX and Linux et IVTTidy.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS .

Identification des incidents pour le test de vérification de l'installation point à point

Le test de vérification de l'installation peut échouer pour les raisons suivantes:

- Si le programme IVT écrit un message indiquant qu'il ne trouve pas de classe, vérifiez que votre chemin d'accès aux classes est défini correctement, comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.
- Le test peut échouer avec le message suivant:

  ```
  Failed to connect to queue manager ' qmgr ' with connection mode ' connMode '
  and host name ' hostname '
  ```

  et un code anomalie associé de 2059. Les variables du message ont les significations suivantes:

  qmgr
  Nom du gestionnaire de files d'attente auquel le programme IVT tente de se connecter. Cette insertion de message est vide si le programme IVT tente de se connecter au gestionnaire de files d'attente par défaut en mode liaisons.

  connMode
  Le mode de connexion, qui est `Bindings` ou `Client`.

  nom_hôte
  Nom d'hôte ou adresse IP du système sur lequel le gestionnaire de files d'attente s'exécute.

  Ce message signifie que le gestionnaire de files d'attente auquel le programme IVT tente de se connecter n'est pas disponible. Vérifiez que le gestionnaire de files d'attente est en cours d'exécution et, si le programme IVT tente de se connecter au gestionnaire de files d'attente par défaut, assurez-vous qu'il est défini comme gestionnaire de files d'attente par défaut pour votre système.
- Le test peut échouer avec le message suivant:

  ```
  Failed to open MQ queue 'SYSTEM.DEFAULT.LOCAL.QUEUE'
  ```

  Ce message signifie que la file d'attente SYSTEM.DEFAULT.LOCAL.QUEUE n'existe pas sur le gestionnaire de files d'attente auquel le programme IVT est connecté. Sinon, si la file d'attente existe, le programme IVT ne peut pas l'ouvrir car il n'est pas activé pour l'insertion et l'extraction de messages. Vérifiez que la file d'attente existe et qu'elle est activée pour l'insertion et l'obtention de messages.
- Le test peut échouer avec le message suivant:

  ```
  Unable to bind to object
  ```

  Ce message signifie qu'il existe une connexion au serveur LDAP, mais que le serveur LDAP n'est pas correctement configuré. Soit le serveur LDAP n'est pas configuré pour stocker des objets Java , soit les droits sur les objets ou le suffixe ne sont pas corrects. Pour plus d'aide dans cette situation, consultez la documentation de votre serveur LDAP.
- Le test peut échouer avec le message suivant:

```
The security authentication was not valid that was supplied for
QueueManager ' qmgr ' with connection mode 'Client' and host name ' hostname '
```

Ce message signifie que le gestionnaire de files d'attente n'est pas correctement configuré pour accepter une connexion client à partir de votre système. Voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098 pour des détails.

IVT de publication / abonnement pour IBM MQ classes for JMS

Un programme de test de vérification de l'installation de publication / abonnement (IVT) est fourni avec IBM MQ classes for JMS. Le programme se connecte à un gestionnaire de files d'attente en liaisons ou en mode client, s'abonne à une rubrique, publie un message sur la rubrique, puis reçoit le message qu'il vient de publier. Le programme peut créer et configurer tous les objets dont il a besoin dynamiquement lors de l'exécution ou utiliser JNDI pour extraire des objets gérés d'un service d'annuaire.

Exécutez le test de vérification de l'installation sans utiliser JNDI au préalable car le test est autonome et ne nécessite pas l'utilisation d'un service d'annuaire. Pour obtenir une description des objets gérés, voir Configuration des objets JMS à l'aide de l'outil d'administration.

Test de vérification de l'installation de publication / abonnement sans utiliser JNDI

Dans ce test, le programme IVT crée et configure tous les objets dont il a besoin dynamiquement lors de l'exécution et n'utilise pas JNDI.

Un script est fourni pour exécuter le programme IVT. Le script est appelé PSIVTRun sur les systèmes AIX and Linux et PSIVTRun.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS .

Pour exécuter le test en mode liaisons, entrez la commande suivante:

```
PSIVTRun -nojndi [-m qmgr ] [-bqm brokerQmgr ] [-v providerVersion ] [-t]
```

Pour exécuter le test en mode client, configurez d'abord le gestionnaire de files d'attente comme décrit dans «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098 en notant que le canal à utiliser est par défaut SYSTEM.DEF.SVRCONN, puis entrez la commande suivante:

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

Les paramètres des commandes ont les significations suivantes:

-m *gestionnaire_files_attente*
Nom du gestionnaire de files d'attente auquel le programme IVT se connecte. Si vous exécutez le test en mode liaisons et omettez ce paramètre, le programme IVT se connecte au gestionnaire de files d'attente par défaut.

-host *nom_hôte*
Nom d'hôte ou adresse IP du système sur lequel le gestionnaire de files d'attente s'exécute.

-port *port*
Numéro du port sur lequel le programme d'écoute du gestionnaire de files d'attente est en mode écoute. La valeur par défaut est 1414.

-channel *canal*
Nom du canal MQI utilisé par le programme IVT pour se connecter au gestionnaire de files d'attente. La valeur par défaut est `SYSTEM.DEF.SVRCONN`.

-bqm *brokerQmgr*
Nom du gestionnaire de files d'attente sur lequel le courtier s'exécute. La valeur par défaut est le nom du gestionnaire de files d'attente auquel le programme IVT se connecte.

Ce paramètre n'est pas pertinent pour le numéro de version de gestionnaire de files d'attente v 7 ou supérieur.

-v *providerVersion*
Niveau d'édition du gestionnaire de files d'attente auquel le programme IVT s'attend à se connecter.

Ce paramètre est utilisé pour définir la propriété PROVIDERVERSION d'un objet de fabrique MQTopicConnectionet possède les mêmes valeurs valides que celles de la propriété PROVIDERVERSION. Pour plus d'informations sur ce paramètre, y compris ses valeurs valides, voir la description de la propriété PROVIDERVERSION dans Propriétés des objets IBM MQ classes for JMS.

La valeur par défaut est `unspecified`.

-ccsid *ccsid*
Identificateur (CCSID) du jeu de caractères codés, ou page de codes, à utiliser par la connexion. La valeur par défaut est 819.

-t
La fonction de trace est activée. Par défaut, la fonction de trace est désactivée.

Un test réussi produit une sortie similaire à l'exemple de sortie suivant:

```
5724-H72, 5655-R36, 5724-L26, 5655-L82 (c) Copyright IBM Corp. 2008, 2024. All
Rights Reserved.
IBM MQ classes for Java Message Service 7.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMSMessage class: jms_text
JMSType:     null
JMSDeliveryMode: 2
JMSExpiration:  0
JMSPriority:   4
JMSMessageID:   ID:414d5120514d5f6d62772020202020201edb14620006706
JMSTimestamp:   1187182520203
JMSCorrelationID: ID:414d5120514d5f6d62772020202020201edb14620006704
JMSDestination:  topic://MQJMS/PSIVT/Information
JMSReplyTo:    null
JMSRedelivered:  false
JMSXUserID: mwhite
JMS_IBM_Encoding: 273
JMS_IBM_PutApplType: 26
JMSXAppID: QM_mbw
JMSXDeliveryCount: 1
JMS_IBM_PutDate: 20070815
JMS_IBM_ConnectionID: 414D5143514D5F6D62772020202020201EDB14620006601
JMS_IBM_PutTime: 12552020
JMS_IBM_Format: MQSTR
JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

Test de vérification de l'installation de publication / abonnement à l'aide de JNDI

Dans ce test, le programme IVT utilise JNDI pour extraire des objets gérés d'un service d'annuaire.

Avant de pouvoir exécuter le test, vous devez configurer un service d'annuaire basé sur un serveur LDAP (Lightweight Directory Access Protocol) ou sur le système de fichiers local. Vous devez également configurer l'outil d'administration d' IBM MQ JMS afin qu'il puisse utiliser le service d'annuaire pour stocker des objets gérés. Pour plus d'informations sur ces prérequis, voir «Prérequis pour IBM MQ classes for JMS», à la page 91. Pour plus d'informations sur la configuration de l'outil d'administration d' IBM MQ JMS , voir Configuration de l'outil d'administration d' JMS.

Le programme IVT doit pouvoir utiliser JNDI pour extraire un objet de fabrique MQTopicConnectionet un objet MQTopic du service d'annuaire. Un script est fourni pour créer ces objets gérés pour vous. Le script est appelé IVTSetup sur les systèmes AIX and Linux et IVTSetup.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS . Pour exécuter le script, entrez la commande suivante:

```
IVTSetup
```

Le script appelle l'outil d'administration IBM MQ JMS pour créer les objets gérés.

L'objet MQTopicConnectionFactory est lié avec le nom ivtTCF et est créé avec les valeurs par défaut pour toutes ses propriétés, ce qui signifie que le programme IVT s'exécute en mode liaisons, se connecte au gestionnaire de files d'attente par défaut et utilise la fonction de publication / abonnement intégrée. Si vous souhaitez que le programme IVT s'exécute en mode client, qu'il se connecte à un gestionnaire de files d'attente autre que le gestionnaire de files d'attente par défaut ou qu'il utilise IBM Integration Bus à la place de la fonction de publication / abonnement intégrée, vous devez utiliser l'outil d'administration IBM MQ JMS ou l'explorateur IBM MQ pour modifier les propriétés appropriées de l'objet de fabrique MQTopicConnection. Pour plus d'informations sur l'utilisation de l'outil d'administration d' IBM MQ JMS , voir Configuration des objets JMS à l'aide de l'outil d'administration. Pour plus d'informations sur l'utilisation de IBM MQ Explorer, voir l'aide fournie avec IBM MQ Explorer.

L'objet MQTopic est lié avec le nom ivtT et est créé avec les valeurs par défaut de toutes ses propriétés, à l'exception de la propriété TOPIC, qui a la valeur MQJMS/PSIVT/Information.

Une fois que vous avez créé les objets gérés, vous pouvez exécuter le programme IVT. Pour exécuter le test à l'aide de JNDI, entrez la commande suivante:

```
PSIVTRun -url "providerURL" [-icf initCtxFact ] [-t]
```

Les paramètres de la commande ont la signification suivante:

-url "*providerURL*"
Adresse URL du service d'annuaire. L'URL peut avoir l'un des formats suivants:

- `ldap://hostname/contextName` , pour un service d'annuaire basé sur un serveur LDAP
- `file:/directoryPath` , pour un service d'annuaire basé sur le système de fichiers local

Notez que vous devez placer l'URL entre guillemets (").

-icf *initCtxFait*
Nom de classe de la fabrique de contexte initial, qui doit être l'une des valeurs suivantes:

- `com.sun.jndi.ldap.LdapCtxFactory`, pour un service d'annuaire basé sur un serveur LDAP. Il s'agit de la valeur par défaut.
- `com.sun.jndi.fscontext.RefFSContextFactory`, pour un service d'annuaire basé sur le système de fichiers local.

-t
La fonction de trace est activée. Par défaut, la fonction de trace est désactivée.

Un test réussi produit une sortie similaire à celle d'un test réussi sans utiliser JNDI. La principale différence est que la sortie indique que le test utilise JNDI pour extraire un objet de fabrique MQTopicConnectionet un objet MQTopic.

Bien que cela ne soit pas strictement nécessaire, il est recommandé de mettre en ordre après le test en supprimant les objets gérés créés par le script IVTSetup. Un script est fourni à cette fin. Le script est appelé IVTTidy sur les systèmes AIX and Linux et IVTTidy.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS .

Identification des problèmes pour le test de vérification de l'installation de publication / abonnement

Le test de vérification de l'installation peut échouer pour les raisons suivantes:

- Si le programme IVT écrit un message indiquant qu'il ne trouve pas de classe, vérifiez que votre chemin d'accès aux classes est défini correctement, comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.
- Le test peut échouer avec le message suivant:

  ```
  Failed to connect to queue manager ' qmgr ' with
  connection mode ' connMode ' and host name ' hostname '
  ```

  et un code anomalie associé de 2059. Les variables du message ont les significations suivantes:

  qmgr
  Nom du gestionnaire de files d'attente auquel le programme IVT tente de se connecter. Cette insertion de message est vide si le programme IVT tente de se connecter au gestionnaire de files d'attente par défaut en mode liaisons.

  connMode
  Le mode de connexion, qui est `Bindings` ou `Client`.

  nom_hôte
  Nom d'hôte ou adresse IP du système sur lequel le gestionnaire de files d'attente s'exécute.

  Ce message signifie que le gestionnaire de files d'attente auquel le programme IVT tente de se connecter n'est pas disponible. Vérifiez que le gestionnaire de files d'attente est en cours d'exécution et, si le programme IVT tente de se connecter au gestionnaire de files d'attente par défaut, assurez-vous qu'il est défini comme gestionnaire de files d'attente par défaut pour votre système.
- Le test peut échouer avec le message suivant:

  ```
  Unable to bind to object
  ```

  Ce message signifie qu'il existe une connexion au serveur LDAP, mais que le serveur LDAP n'est pas correctement configuré. Soit le serveur LDAP n'est pas configuré pour stocker des objets Java , soit les droits sur les objets ou le suffixe ne sont pas corrects. Pour plus d'aide dans cette situation, consultez la documentation de votre serveur LDAP.
- Le test peut échouer avec le message suivant:

  ```
  The security authentication was not valid that was supplied for
  QueueManager 'qmgr' with connection mode 'Client' and host name 'hostname'
  ```

  Ce message signifie que le gestionnaire de files d'attente n'est pas configuré correctement pour accepter une connexion client à partir de votre système. Pour plus d'informations, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.

JMS 2.0 *Utilisation des modèles d'application IBM MQ classes for JMS*

Les modèles d'application IBM MQ classes for JMS fournissent une présentation des fonctions communes de l'API JMS . Vous pouvez les utiliser pour vérifier la configuration de votre installation et de votre serveur de messagerie et pour vous aider à créer vos propres applications.

Pourquoi et quand exécuter cette tâche

Si vous avez besoin d'aide pour créer vos propres applications, vous pouvez utiliser les modèles d'application comme base de départ. La version source et la version compilée sont fournies pour chaque application. Utilisez le code source exemple pour identifier les étapes clé permettant de créer chaque objet requis pour votre application (ConnectionFactory, Connection, Session, Destination, mais aussi un expéditeur et/ou un consommateur) et de définir les propriétés spécifiques dont vous avez besoin pour spécifier la manière dont l'application doit fonctionner. Pour plus d'informations, voir «Ecriture d'applications IBM MQ classes for JMS/Jakarta Messaging», à la page 144. Les exemples peuvent être sujets à des modifications dans les éditions futures d' IBM MQ.

Pour JMS 2.0, Tableau 11, à la page 124 indique où les modèles d'application IBM MQ classes for JMS sont installés sur chaque plateforme.

Remarque :

JM 3.0 Pour IBM MQ classes for Jakarta Messaging, de nouveaux exemples sont en cours de préparation.

Tableau 11. Répertoires d'installation des exemples d'application IBM MQ classes for JMS

Plateforme	**Répertoire**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/jms/samples`
Windows Windows	*MQ_INSTALLATION_PATH*`\tools\jms\samples`
IBM i IBM i	`/qibm/proddata/mqm/java/samples/jms/samples`
z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/samples/jms`

Dans ce répertoire, il existe des sous-répertoires qui contiennent un ou plusieurs exemples d'application, comme illustré dans la Tableau 12, à la page 124.

Tableau 12. IBM MQ classes for JMS modèles d'application

Nom du modèle	**Description**
JmsBrowser.java	Une application de navigateur de files d'attente JMS qui examine tous les messages disponibles dans la file d'attente nommée, sans les supprimer, dans l'ordre dans lequel ils seraient reçus par une application client.
JmsConsumer.java	Une application de navigateur de files d'attente JMS qui examine tous les messages disponibles dans la file d'attente nommée, sans les supprimer, dans l'ordre dans lequel ils seraient reçus par une application client, en recherchant l'instance de fabrique de connexions et l'instance de destination dans un contexte initial (cet exemple prend en charge uniquement le contexte du système de fichiers).
JmsJndiConsumer.java	Une application de destinataire (récepteur ou abonné) JMS qui reçoit un message de la destination nommée (file d'attente ou rubrique) en recherchant l'instance de fabrique de connexions et l'instance de destination dans un contexte initial (cet exemple prend en charge uniquement le contexte du système de fichiers).

Tableau 12. IBM MQ classes for JMS modèles d'application (suite)	
Nom du modèle	**Description**
JmsJndiProducer.java	Application de fournisseur JMS (émetteur ou diffuseur de publications) qui envoie un message simple à la destination nommée (file d'attente ou rubrique) en recherchant l'instance de fabrique de connexions et l'instance de destination dans un contexte initial (cet exemple prend en charge uniquement le contexte du système de fichiers).
JmsProducer.java	Application de fournisseur JMS (expéditeur ou diffuseur de publications) qui envoie un message simple à la destination nommée (file d'attente ou rubrique).
/interactive/	
SampleConsumerJava.java	Réception de message (s) à partir d'un sujet / d'une file d'attente.
SampleProducerJava.java	Envoyer le (s) message (s) à une rubrique / file d'attente.
/interactive/helper/	
BaseOptions.java	Classe abstraite pouvant être étendue pour fournir la fonctionnalité d'option (s) utilisateur.
IsValidType.java	Classe abstraite pour les classes du vérificateur de validité.
JmsApp.java	Classe abstraite qui peut être étendue pour fournir la fonctionnalité de consommateur / producteur.
Keys.java	Ensemble de clés qui définissent les options des exemples d'application.
Literals.java	Ensemble de littéraux constants.
MyContext.java	Contexte dans lequel les options sont présentées.
Options.java	Fournit des fonctionnalités pour la ou les options utilisateur.
OptionsPresenter.java	Contexte dans lequel les options en cours sont présentées.
/simple/	
SimpleAsyncPutPTP.java	Application simple pour la messagerie point-à-point ; le message est envoyé de manière asynchrone (également appelée messagerie *fire-and-forget*). Aucun message n'est reçu.
SimpleDurableSub.java	Application simple qui illustre la fonction d'abonnement durable.
SimpleJNDILookup.java	Application minimale et simple qui illustre la recherche d'objets JMS à l'aide du contexte initial. Aucune connexion au gestionnaire de files d'attente n'est établie et aucun message n'est envoyé ou reçu.
SimpleMQMDRead.java	Application simple qui montre comment une application JMS peut utiliser des zones MQ Message Descriptor (MQMD) en tant que propriétés de message JMS . Aucun message n'est envoyé ; il est supposé que la file d'attente en cours d'utilisation est remplie avec certains messages.
SimpleMQMDWrite.java	Application simple qui montre comment une application JMS peut écrire des zones MQ Message Descriptor (MQMD). Aucun message n'est reçu.

Tableau 12. IBM MQ classes for JMS modèles d'application (suite)	
Nom du modèle	**Description**
SimplePTP.java	Une application minimale et simple pour la messagerie point-à-point.
SimplePubSub.java	Application minimale et simple pour la messagerie de publication / abonnement.
SimpleReadAheadPTP.java	Application simple pour la messagerie point-à-point ; les messages sont diffusés en continu à partir du gestionnaire de files d'attente (également appelé fonction de lecture anticipée). Aucun message n'est envoyé ; il est supposé que la file d'attente en cours d'utilisation est remplie avec certains messages.
SimpleRequestor.java	Application simple qui utilise un demandeur pour envoyer un message de demande, puis attendre et recevoir la réponse. Remarque: Il est supposé qu'une autre application traitera le message de demande et enverra le message de réponse.
SimpleResponder.java	Application simple qui écoute sur une destination un message, puis envoie une réponse à la destination replyTo du message. L'application est écrite pour fonctionner avec l'exemple SimpleRequestor .
SimpleRetainedPub.java	Application simple qui illustre une publication conservée. Aucun message n'est reçu.
SimpleWMQJMSPTP.java	Une application minimale et simple pour la messagerie point-à-point.
SimpleWMQJMSPubSub.java	Application minimale et simple pour la messagerie de publication / abonnement.

IBM MQ classes for JMS fournit un script appelé `runjms` qui peut être utilisé pour exécuter les modèles d'application. Ce script configure l'environnement IBM MQ pour vous permettre d'exécuter les modèles d'application IBM MQ classes for JMS .

Tableau 13, à la page 126 indique l'emplacement du script sur chaque plateforme:

Tableau 13. Emplacement du script `runjms`	
Plateforme	**Répertoire**
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/java/bin/runjms`
Windows Windows	*MQ_INSTALLATION_PATH*`\java\bin\runjms.bat`

IBM i IBM i	`/qibm/proddata/mqm/java/bin/runjms`
	ou
	`/qibm/proddata/mqm/java/bin/runjms64`

z/OS z/OS	*MQ_INSTALLATION_PATH*`/java/bin/runjms`

Pour utiliser le script `runjms` afin d'appeler un exemple d'application, procédez comme suit:

Procédure

1. Ouvrez une invite de commande et accédez au répertoire contenant le modèle d'application que vous souhaitez exécuter.
2. Entrez la commande suivante :

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Le modèle d'application affiche la liste des paramètres dont il a besoin.
3. Entrez la commande suivante pour exécuter l'exemple avec ces paramètres:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Exemple

Linux Par exemple, pour exécuter l'exemple `JmsBrowser` sous Linux, entrez les commandes suivantes:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Concepts associés

«Ce qui est installé pour IBM MQ classes for JMS», à la page 92
Un certain nombre de fichiers et de répertoires sont créés lorsque vous installez IBM MQ classes for JMS. Sous Windows, une partie de la configuration est effectuée lors de l'installation en définissant automatiquement des variables d'environnement. Sur d'autres plateformes et dans certains environnements Windows , vous devez définir des variables d'environnement avant de pouvoir exécuter des applications IBM MQ classes for JMS .

Scripts fournis avec IBM MQ classes for JMS/Jakarta Messaging

Un certain nombre de scripts sont fournis pour vous aider à exécuter les tâches courantes qui doivent être exécutées lors de l'utilisation de IBM MQ classes for JMS et de IBM MQ classes for Jakarta Messaging.

Tableau 14, à la page 127 répertorie tous les scripts et leurs utilisations. Les scripts se trouvent dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

Tableau 14. Scripts fournis avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging

Service public	**Utilisation**
Nettoyage [1]	Ce script est géré à des fins de compatibilité avec les éditions précédentes, mais il n'effectue aucune fonction. Le nettoyage manuel des informations d'abonnement n'est plus nécessaire.
DefaultConfiguration	Exécute l'application de configuration par défaut sur des plateformes autres que Windows.
formatLog [1]	Ce script est géré à des fins de compatibilité avec les éditions précédentes, mais il n'effectue aucune fonction. La sortie du journal est désormais générée sous forme de texte lisible.
IVTRun [1] IVTSetup [1] IVTTidy [1]	Utilisé dans le test de vérification de l'installation point à point, comme décrit dans «L'outil IVT point-à-point pour IBM MQ classes for JMS», à la page 115.
JM 3.0 JMS30Admin [1]	Exécute l'outil d'administration IBM MQ Jakarta Messaging , comme décrit dans Démarrage de l'outil d'administration.

Tableau 14. Scripts fournis avec IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging (suite)	
Service public	**Utilisation**
JM 3.0 JMS30Admin.config	Fichier de configuration de l'outil d'administration de IBM MQ Jakarta Messaging , comme décrit dans Configuration de l'outil d'administration de JMS.
JMS 2.0 JMSAdmin [1]	Exécute l'outil d'administration IBM MQ JMS , comme décrit dans Démarrage de l'outil d'administration.
JMS 2.0 JMSAdmin.config	Fichier de configuration de l'outil d'administration de IBM MQ JMS , comme décrit dans Configuration de l'outil d'administration de JMS.
PSIVTRun [1]	Exécute le programme de test de vérification de l'installation de publication / abonnement, comme décrit dans «IVT de publication / abonnement pour IBM MQ classes for JMS», à la page 120.
PSReportDump.class	Cette classe est gérée à des fins de compatibilité avec les éditions précédentes, mais n'effectue aucune fonction.
JM 3.0 setjms30env «2», à la page 128	Linux AIX Pour Jakarta Messaging 3.0, définit les variables d'environnement pour l'exécution d'une application IBM MQ classes for JMS dans une machine virtuelle Java 32 bits sur les systèmes AIX and Linux , comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.
JMS 2.0 setjmsenv «2», à la page 128	Linux AIX Pour JMS 2.0, définit les variables d'environnement pour l'exécution d'une application IBM MQ classes for JMS dans une machine virtuelle Java 32 bits sur les systèmes AIX and Linux , comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.
JM 3.0 setjms30env64 «2», à la page 128	Linux AIX Pour Jakarta Messaging 3.0, définit les variables d'environnement pour l'exécution d'une application IBM MQ classes for JMS dans une machine virtuelle Java 64 bits sur les systèmes AIX and Linux , comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.
JMS 2.0 setjmsenv64 «2», à la page 128	Linux AIX Pour JMS 2.0, définit les variables d'environnement pour l'exécution d'une application IBM MQ classes for JMS dans une machine virtuelle Java 64 bits sur les systèmes AIX and Linux , comme décrit dans «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.

Remarque :

1. Sous Windows, le nom de fichier porte l'extension `.bat`.
2. Ces scripts sont disponibles sous AIX and Linux uniquement. Sous Windows, après avoir installé IBM MQ, exécutez la commande **setmqenv**. Pour plus d'informations, voir «Définition des variables d'environnement pour IBM MQ classes for JMS/Jakarta Messaging», à la page 96.

Prise en charge d'OSGi avec IBM MQ classes for JMS

OSGi fournit une infrastructure qui prend en charge le déploiement d'applications en tant que bundles. Les bundles OSGi sont fournis avec IBM MQ classes for JMS.

IBM MQ classes for JMS inclut les bundles OSGi suivants.

com.ibm.msg.client.osgi.jms*version_number*.jar
Couche de code commune dans IBM MQ classes for JMS. Pour plus d'informations sur l'architecture en couches des classes IBM MQ pour JMS, voir IBM MQ classes for JMS architecture.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
Fichiers d'archive Java (JAR) prérequis pour la couche commune.

com.ibm.msg.client.osgi.commonservices.j2se_*version_number*.jar
Services communs pour les applications Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_*version_number*.jar
Messages pour la couche commune.

com.ibm.msg.client.osgi.wmq_*version_number*.jar
Le fournisseur de messagerie IBM MQ dans IBM MQ classes for JMS. Pour plus d'informations sur l'architecture en couches de IBM MQ classes for JMS, voir IBM MQ classes for JMS architecture.

com.ibm.msg.client.osgi.wmq.prereq_*version_number*.jar
Fichiers JAR prérequis pour le fournisseur de messagerie IBM MQ .

com.ibm.msg.client.osgi.wmq.nls_*version_number*.jar
Messages du fournisseur de messagerie IBM MQ .

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 Pour Jakarta Messaging 3.0, ce fichier JAR permet aux applications d'utiliser à la fois IBM MQ classes for JMS et IBM MQ classes for Java, et inclut également le code permettant de gérer les messages PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 Pour Jakarta Messaging 3.0, ce fichier JAR fournit les prérequis pour com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Pour JMS 2.0, ce fichier JAR permet aux applications d'utiliser à la fois IBM MQ classes for JMS et IBM MQ classes for Java, et inclut également le code permettant de gérer les messages PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Pour JMS 2.0, ce fichier JAR fournit les prérequis pour com.ibm.mq.osgi.allclient_*version_number*.jar.

où *version_number* est le numéro de version de IBM MQ qui est installé.

Les bundles sont installés dans le sous-répertoire java/lib/OSGi de votre installation IBM MQ ou dans le dossier java\lib\OSGi sous Windows.

Depuis IBM MQ 8.0, utilisez les bundles com.ibm.mq.osgi.allclient_8.0.0.0.jaret com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar pour toute nouvelle application. L'utilisation de ces bundles supprime la restriction de ne pas pouvoir exécuter à la fois IBM MQ classes for JMS et IBM MQ classes for Java dans la même infrastructure OSGi, mais toutes les autres restrictions s'appliquent.

Le bundle com.ibm.mq.osgi.java*version_number*.jar, qui est également installé dans le sous-répertoire java/lib/OSGi de votre installation IBM MQ ou dans le dossier java\lib\OSGi sous Windows, fait partie de IBM MQ classes for Java. Ce bundle ne doit pas être chargé dans un environnement d'exécution OSGi dans lequel IBM MQ classes for JMS est chargé.

Les bundles OSGi pour IBM MQ classes for JMS ont été écrits dans la spécification OSGi Release 4. Ils ne fonctionnent pas dans un environnement OSGi Release 3.

Vous devez définir correctement le chemin du système ou le chemin de la bibliothèque pour que l'environnement d'exécution OSGi puisse trouver les fichiers DLL ou les bibliothèques partagées requis.

Si vous utilisez les bundles OSGi pour IBM MQ classes for JMS, les rubriques temporaires ne fonctionnent pas. En outre, les classes d'exit de canal écrites en Java ne sont pas prises en charge en raison d'un problème inhérent au chargement des classes dans un environnement de chargeur de classes multiples

tel que OSGi. Un bundle d'utilisateurs peut connaître les bundles IBM MQ classes for JMS , mais les bundles IBM MQ classes for JMS ne connaissent aucun bundle d'utilisateurs. Par conséquent, le chargeur de classe utilisé dans un bundle IBM MQ classes for JMS ne peut pas charger une classe d'exit de canal qui se trouve dans un bundle utilisateur.

Pour plus d'informations sur OSGi, voir le site Web OSGi Alliance .

z/OS MQ Adv. VUE JMS/Jakarta Messaging client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for JMS/Jakarta Messaging application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.
- Le gestionnaire de files d'attente connecté est en cours d'exécution avec l'autorisation d'utilisation d' IBM MQ Advanced for z/OS Value Unit Edition et par conséquent, le paramètre **ADVCAP** est défini sur ENABLED.

Pour plus d'informations sur IBM MQ Advanced for z/OS Value Unit Edition , voir IBM MQ product Identifier and export information.

Voir DISPLAY QMGR pour plus d'informations sur **ADVCAP** et START QMGR pour plus d'informations sur **QMGRPROD**.

Note that batch is the only environment supported; there is no support for JMS/Jakarta Messaging for CICS or JMS/Jakarta Messaging for IMS.

An IBM MQ classes for JMS/Jakarta Messaging application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS.

If an IBM MQ classes for JMS/Jakarta Messaging application on z/OS attempts to connect using client mode, and is not allowed to do so, exception message JMSFMQ0005 is issued.

Advanced Message Security (AMS) support

IBM MQ classes for JMS/Jakarta Messaging client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Pour utiliser AMS de cette manière, les applications client doivent utiliser un type de magasin de clés `jceracfks` dans `keystore.conf`, où:

- Le préfixe de nom de propriété est `jceracfks` et ce préfixe de nom est insensible à la casse.
- Le magasin de clés est un fichier de clés RACF .
- Les mots de passe ne sont pas obligatoires et seront ignorés. En effet, les fichiers de clés RACF n'utilisent pas de mots de passe.
- Si vous spécifiez le fournisseur, celui-ci doit être `IBMJCE`.

Lorsque vous utilisez `jceracfks` avec AMS, le magasin de clés doit être au format suivant: `safkeyring://`*`user`*`/`*`keyring`*, où:

- `safkeyring` est un littéral et ce nom est insensible à la casse
- *`user`* est l'ID utilisateur RACF qui possède le fichier de clés
- *`keyring`* est le nom du fichier de clés RACF et le nom du fichier de clés est sensible à la casse

L'exemple suivant utilise le fichier de clés AMS standard pour l'utilisateur `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

“Java client connectivity to batch applications running on z/OS” on page 381
Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

Obtention du IBM MQ classes for JMS et du IBM MQ classes for Jakarta Messaging séparément

Les bibliothèques et les utilitaires nécessaires à l'exécution d'applications à l'aide de IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging sont disponibles dans un fichier JAR auto-extractible que vous téléchargez depuis Fix Central. Vous pouvez effectuer cette opération si vous souhaitez obtenir uniquement ces fichiers, par exemple pour le déploiement dans un outil de gestion de logiciels ou pour une utilisation avec des applications client autonomes.

Avant de commencer

Avant de commencer cette tâche, assurez-vous qu'un environnement d'exécution Java (JRE) Java runtime environment est installé sur votre machine et que ce dernier a été ajouté au chemin du système.

Le programme d'installation de Java utilisé dans ce processus d'installation ne nécessite pas d'exécution en tant que superutilisateur ou utilisateur spécifique. La seule condition requise est que l'utilisateur sous lequel il est exécuté dispose d'un accès en écriture au répertoire dans lequel vous souhaitez placer les fichiers.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Pourquoi et quand exécuter cette tâche

Un fichier JAR auto-extractible est utilisé pour réduire la taille du téléchargement et le temps nécessaire à l'exécution de l'extraction. Le contenu exact de ce fichier JAR et les sous-répertoires dans lesquels il extrait les fichiers dépendent de la version de IBM MQ.

Lorsque vous exécutez le fichier JAR auto-extractible, il affiche le contrat de licence IBM MQ , qui doit être accepté. Il vous permet également de modifier le répertoire parent de l'extraction.

Pour IBM MQ 9.3 et les versions ultérieures, le fichier JAR auto-extractible extrait les fichiers dans la structure de répertoire suivante:

wmq/JavaEE
Fichiers EAR et RAR de l'adaptateur de ressources IBM MQ .

JMS 2.0 Les fichiers suivants doivent être utilisés avec les objets JMS 2.0 et JMS 1.1 :

- `wmq.jmsra.ivt.ear`
- `wmq.jmsra.rar`

JM 3.0 Il existe également un ensemble équivalent à utiliser avec les objets Jakarta Messaging 3.0 :

- `wmq.jakarta.jmsra.ivt.ear`: Contient les fichiers de test de vérification de l'installation.
- `wmq.jakarta.jmsra.rar`: Contient les fichiers d'adaptateur de ressources.

wmq/JavaSE

wmq/JavaSE/bin
Les outils **JMSAdmin** et **JMS30Admin** . Permet de définir des entités JNDI qui représentent des objets JMS ou Jakarta Messaging .

JMS 2.0 Les fichiers suivants doivent être utilisés avec les objets JMS 2.0 et JMS 1.1 :

- `JMSAdmin.bat`
- `JMSAdmin`
- `JMSAdmin.config`

JM 3.0 Il existe également un ensemble équivalent à utiliser avec les objets Jakarta Messaging 3.0 :

- `JMS30Admin.bat`: Fichier utilisé pour démarrer l'outil sous Windows.
- `JMS30Admin`: Script utilisé pour démarrer l'outil sur les plateformes Linux et UNIX .
- `JMS30Admin.config`: Exemple de fichier de configuration pour l'outil.

Remarque :

- Avant l'ajout de l'outil **JMSAdmin** au fichier JAR auto-extractible, les fichiers de ce répertoire se trouvaient dans le répertoire parent `wmq/JavaSE`.
- Un client installé à l'aide du fichier JAR auto-extractible peut utiliser l'outil **JMSAdmin** ou **JMS30Admin** pour créer des objets gérés de messagerie Java dans un contexte de système de fichiers (fichier `.bindings`). Le client peut également rechercher et utiliser ces objets administrés.
- JMS 2.0 L'outil **JMSAdmin** à utiliser avec les objets JMS 2.0 et JMS 1.1 a été ajouté au fichier JAR auto-extractible dans IBM MQ 9.2.0 Fix Pack 2 et IBM MQ 9.2.2.
- JM 3.0 L'outil **JMS30Admin** à utiliser avec les objets Jakarta Messaging 3.0 a été ajouté au fichier JAR auto-extractible dans IBM MQ 9.3.0.

`wmq/JavaSE/lib`

Advanced Message Security utilise les packages open source Bouncy Castle suivants pour prendre en charge la syntaxe de message cryptographique (CMS). Voir Support for non-IBM JREs with AMS.

V 9.4.0 Depuis IBM MQ 9.4.0:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

Les fichiers suivants contiennent chacun les classes correspondant à leur niveau JMS ou Jakarta Messaging spécifique:

- JMS 2.0 `com.ibm.mq.allclient.jar` (JMS 2.0 et JMS 1.1)
- JM 3.0 `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0)

Autres fichiers JAR prérequis:

- `fscontext.jar`: Obligatoire si votre application effectue des recherches JNDI à l'aide d'un contexte de système de fichiers.
- JM 3.0 `jakarta.jms-api.jar`: Contient l'interface Jakarta Messaging 3.0 et les définitions d'exception.
- JMS 2.0 `jms.jar`: Contient l'interface JMS 2.0 et les définitions d'exception.
- `org.json.jar`: Contient des classes qui permettent à IBM MQ classes for JMS d'interpréter les fichiers CCDT au format JSON.
- `providerutil.jar`: Obligatoire si votre application effectue des recherches JNDI à l'aide d'un contexte de système de fichiers.

Remarque : Stabilized `com.ibm.mq.allclient.jar` et `com.ibm.mq.jakarta.client.jar` contiennent tous deux une copie de IBM MQ classes for Java. Toutefois, dans IBM MQ 9.0, ces classes sont déclarées comme étant stabilisées fonctionnellement au niveau fourni dans IBM MQ 8.0. Voir Dépréciations, stabilisations et retraits dans IBM MQ 9.0.

wmq/OSGi
Les bundles de client OSGi IBM MQ :

- JM 3.0 `com.ibm.mq.jakarta.osgi.allclient_V.R.M.F.jar`
- JM 3.0 `com.ibm.mq.jakarta.osgi.allclientprereqs_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclient_V.R.M.F.jar`
- JMS 2.0 `com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar`

où *V.R.M.F* est le numéro de version, d'édition, de modification et de groupe de correctifs.

Procédure

1. Téléchargez le fichier JAR du client IBM MQ Java / JMS à partir de Fix Central.
   a) Cliquez sur ce lien: IBM MQ Java / JMS client.
   b) Recherchez le client correspondant à votre version de IBM MQ dans la liste affichée des correctifs disponibles.
   Exemple :

   ```
   release level: 9.3.0.0-IBM-MQ-Install-Java-All
   Long Term Support: 9.3.0.0 IBM MQ JMS and Java 'All Client'
   ```

   Cliquez ensuite sur le nom du fichier client et suivez le processus de téléchargement.
2. Lancez l'extraction à partir du répertoire dans lequel vous avez téléchargé le fichier.
   Pour démarrer l'extraction, entrez une commande au format suivant:

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

   où *V.R.M.F* est le numéro de version du produit, par exemple `9.3.0.0`, et *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` est le nom du fichier qui a été téléchargé depuis Fix Central.
   Par exemple, pour extraire le client JMS pour l'édition IBM MQ 9.4.0 , utilisez la commande suivante:

   ```
   java -jar 9.4.0.0-IBM-MQ-Install-Java-All.jar
   ```

   Remarque : Pour effectuer cette installation, vous devez disposer d'un environnement d'exécution Java (JRE) installé sur votre machine et ajouté au chemin système.

   Lorsque vous entrez la commande, les informations suivantes s'affichent:

   ```
   Avant de pouvoir utiliser, extraire ou installer IBM MQ V9.4, vous devez accepter
   les termes de 1. IBM International License Agreement for Evaluation of
   Programmes 2. IBM -Conditions Internationales d'Utilisation de Logiciels IBM et autres
   license information. Veuillez lire attentivement les contrats de licence suivants.

   The license agreement is separately viewable using the
   --viewLicenseAgreement option.

   Press Enter to display the license terms now, or 'x' to skip.
   ```
3. Lisez et acceptez les dispositions du contrat de licence:
   a) Pour afficher la licence, appuyez sur Entrée.
   Vous pouvez également appuyer sur x pour ignorer l'affichage de la licence.
   Une fois la licence affichée ou immédiatement si vous appuyez sur x, le message suivant s'affiche:

```
Des informations supplémentaires sur les licences peuvent être visualisées séparément à
l'aide du
-- Option d'informationviewLicense.

Appuyez sur Entrée pour afficher des informations supplémentaires sur la licence ou
sur'x'pour les ignorer.
```

b) Pour afficher les dispositions de licence supplémentaires, appuyez sur Entrée.

Vous pouvez également appuyer sur x pour ignorer l'affichage des dispositions de licence supplémentaires.

Une fois les dispositions du contrat de licence supplémentaires affichées, ou immédiatement si vous appuyez sur x, le message suivant s'affiche:

```
By choosing the "I Agree" option below, you agree to the terms of the
license agreement and non-IBM terms, if applicable. Si vous ne
agree, select "I do not Agree".

Select [1] I Agree, or [2] I do not Agree:
```

c) Pour accepter le contrat de licence et continuer à sélectionner le répertoire d'installation, sélectionnez 1.

Vous pouvez également sélectionner 2 pour mettre fin immédiatement à l'installation.

Si vous sélectionnez 1, un message similaire au message suivant s'affiche:

```
Enter directory for product files or leave blank to accept the default value.
Le répertoire cible par défaut est H: \downloads

Target directory for product files?
```

4. Indiquez le répertoire parent pour l'extraction.

L'emplacement par défaut est le répertoire de travail.

- Si vous souhaitez extraire les fichiers du produit à l'emplacement par défaut, appuyez sur Entrée sans indiquer de valeur.
- Si vous souhaitez extraire les fichiers du produit à un autre emplacement, indiquez le nom du répertoire dans lequel vous souhaitez extraire les fichiers, puis appuyez sur Entrée pour lancer l'extraction.

Le nom de répertoire que vous spécifiez ne doit pas déjà exister. Dans le cas contraire, lorsque vous démarrez l'extraction, une erreur est signalée et aucun fichier n'est installé.

S'il n'existe pas déjà, le répertoire indiqué est créé et les fichiers programme sont extraits dans ce répertoire. Lors de l'installation, un nouveau répertoire nommé `wmq` est créé dans le répertoire parent que vous avez spécifié.

Trois sous-répertoires, `JavaEE`, `JavaSE`et `OSGi`, sont créés dans le répertoire `wmq` avec le contenu suivant:

JavaEE

JM 3.0 `wmq.jakarta.jmsra.ivt.ear`

JM 3.0 `wmq.jakarta.jmsra.rar`

JMS 2.0 `wmq.jmsra.ivt.ear`

JMS 2.0 `wmq.jmsra.rar`

JavaSE

Ce répertoire contient les sous-répertoires et fichiers suivants:

JavaSE/lib

V 9.4.0 `bcpkix-jdk18on.jar`

V 9.4.0 `bcprov-jdk18on.jar`

V 9.4.0 `bcutil-jdk18on.jar`

JMS 2.0 `com.ibm.mq.allclient.jar`

JM 3.0 com.ibm.mq.jakarta.client.jar

fscontext.jar

jms.jar

org.json.jar

providerutil.jar

JavaSE/bin

JMSAdmin.bat

JMSAdmin

JMSAdmin.config

OSGi

JM 3.0 com.ibm.mq.jakarta.osgi.allclient_*V.R.M.F*.jar

JM 3.0 com.ibm.mq.jakarta.osgi.allclientprereqs_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclient_*V.R.M.F*.jar

JMS 2.0 com.ibm.mq.osgi.allclientprereqs_*V.R.M.F*.jar

Une fois l'extraction terminée, un message de confirmation s'affiche, comme illustré dans l'exemple suivant:

```
Extraction des fichiers dans H: \downloads\wmq
Successfully extracted all product files.
```

Liste autorisée dans IBM MQ classes for JMS/Jakarta Messaging

Le mécanisme de sérialisation et de désérialisation des objets Java a été identifié comme un risque potentiel pour la sécurité. Les listes autorisées dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging offrent une protection contre certains risques de sérialisation.

Pourquoi et quand exécuter cette tâche

Le mécanisme de sérialisation et de désérialisation des objets Java a été identifié comme un risque de sécurité potentiel car la désérialisation instancie des objets Java arbitraires, où il est possible que des données envoyées de manière malveillante causent divers problèmes. Une application notable de la sérialisation est dans Jakarta Messaging 3.0 et Java Message Service 2.0 ObjectMessages qui utilisent la sérialisation pour encapsuler et transférer des objets arbitraires.

La mise en liste autorisée de la sérialisation est une atténuation potentielle contre certains des risques que pose la sérialisation. En spécifiant explicitement les classes qui peuvent être encapsulées dans, et extraites de, ObjectMessages, la mise sur liste autorisée offre une protection contre certains risques de sérialisation.

Concepts associés

«Exécution d'applications IBM MQ classes for JMS sous Java security manager», à la page 110
IBM MQ classes for JMS peut s'exécuter avec Java security manager activé. Pour exécuter des applications avec le Java security manager activé, vous devez configurer votre Java Virtual Machine (JVM) avec un fichier de configuration de règles approprié.

Liste autorisée des concepts

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messagingprennent en charge la liste autorisée des classes dans l'implémentation de l'interface JMS ObjectMessage . Cela permet d'atténuer certains risques de sécurité liés au mécanisme de sérialisation et de désérialisation des objets Java .

Liste autorisée dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging

Important :

Partout où c'était possible, le terme *allowlist* a remplacé le terme *whitelist*. Cela inclut certains noms de propriété système Java mentionnés dans cette rubrique. Il n'est pas nécessaire de changer la configuration existante. Les noms de propriété système précédents fonctionnent également.

IBM MQ classes for JMS (JMS 2.0) et IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) prennent en charge la liste autorisée des classes dans l'implémentation de l'interface JMS ObjectMessage .

- JMS 2.0 Pour IBM MQ classes for JMS, les noms de propriété appropriés sont **`com.ibm.mq.jms.allowlist.*`**.
- JM 3.0 Pour IBM MQ classes for Jakarta Messaging, les noms de propriété appropriés sont **`com.ibm.mq.jakarta.jms.allowlist.*`**

Une liste autorisée définit les classes Java qui peuvent être sérialisées avec ObjectMessage.setObject() et désérialisées avec ObjectMessage.getObject().

- JMS 2.0 Les tentatives de sérialisation ou de désérialisation d'une instance d'une classe non incluse dans la liste autorisée avec ObjectMessage entraînent l'émission d'une exception javax.jms.MessageFormatException avec une exception java.io.InvalidClassException comme cause.
- JM 3.0 Les tentatives de sérialisation ou de désérialisation d'une instance d'une classe non incluse dans la liste autorisée avec ObjectMessage provoquent l'émission d'une exception jakarta.jms.MessageFormatException avec une exception java.io.InvalidClassException comme cause.

Génération de la liste autorisée

Important : IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging ne peuvent pas être distribués avec une liste autorisée. Le choix des classes à transférer à l'aide de ObjectMessages est un choix de conception d'application et IBM MQ ne peut pas le préempter.

Pour cette raison, le mécanisme de liste autorisée permet deux modes de fonctionnement:

DISCOVERY
: Dans ce mode, le mécanisme génère une liste des noms de classe qualifiés complets, en signalant toutes les classes qui ont été observées comme étant sérialisées ou désérialisées dans ObjectMessages.

Application
: Dans ce mode, le mécanisme impose la mise en liste autorisée, en rejetant les tentatives de sérialisation ou de désérialisation des classes qui ne figurent pas dans la liste autorisée.

Si vous souhaitez utiliser ce mécanisme, vous devez d'abord exécuter en mode DISCOVERY pour regrouper la liste des classes actuellement sérialisées et désérialisées, examiner la liste et l'utiliser comme base de votre liste autorisée. Il peut même être approprié d'utiliser la liste inchangée, mais la liste doit d'abord être revue avant que vous ne décidiez de le faire.

Contrôle du mécanisme de liste autorisée

Trois propriétés système sont disponibles pour contrôler le mécanisme de liste autorisée:

com.ibm.mq.jms.allowlist (JMS 2.0) et com.ibm.mq.jakarta.jms.allowlist (Jakarta Messaging 3.0)
: Cette propriété peut être spécifiée de l'une des manières suivantes:

  - Nom de chemin du fichier qui contient la liste autorisée, au format d'URI de fichier (c'est-à-dire, commençant par `file:`). En mode DISCOVERY, ce fichier est écrit par le mécanisme de liste autorisée. Le fichier ne doit pas exister. Si le fichier existe, le mécanisme émet une exception au lieu de l'écraser. En mode ENFORCEMENT, ce fichier est lu par le mécanisme de liste autorisée.

- Noms de classe qualifiés complets séparés par des virgules qui constituent la liste autorisée.

  Si cette propriété n'est pas définie, le mécanisme de liste autorisée est inactif.

  Si vous utilisez un Java security manager, vous devez vous assurer que les fichiers JAR IBM MQ classes for JMS disposent d'un accès en lecture et en écriture à ce fichier.

com.ibm.mq.jms.allowlist.discover (JMS 2.0) et com.ibm.mq.jakarta.jms.allowlist.discover (Jakarta Messaging 3.0)

- Si cette propriété n'est pas définie ou est définie sur false, le mécanisme de liste autorisée s'exécute en mode ENFORCEMENT.
- Si cette propriété est définie sur true et que la liste autorisée a été spécifiée en tant qu'URI de fichier, le mécanisme de liste autorisée s'exécute en mode DISCOVERY.
- Si cette propriété est définie sur true et que la liste autorisée a été spécifiée en tant que liste de noms de classe, le mécanisme de liste autorisée émet une exception appropriée.
- Si cette propriété est définie sur true et que la liste autorisée n'a pas été spécifiée à l'aide de la propriété com.ibm.mq.jms.allowlist ou com.ibm.mq.jakarta.jms.allowlist, le mécanisme de liste autorisée est inactif.
- Si cette propriété est définie sur true et que le fichier de liste autorisée existe déjà, le mécanisme de liste autorisée émet une exception java.io.InvalidClassException et les entrées ne sont pas ajoutées au fichier.

com.ibm.mq.jms.allowlist.mode (JMS 2.0) et com.ibm.mq.jakarta.jms.allowlist.mode (Jakarta Messaging 3.0)

Cette propriété de chaîne peut être spécifiée de l'une des trois manières suivantes:

- Si cette propriété est définie sur SERIALIZE, le mode ENFORCEMENT effectue une validation de liste autorisée uniquement sur la méthode ObjectMessage.setObject().
- Si cette propriété est définie sur DESERIALIZE, le mode ENFORCEMENT effectue la validation de la liste autorisée uniquement sur la méthode ObjectMessage.getObject().
- Si cette propriété n'est pas définie ou qu'elle est définie sur une autre valeur, le mode ENFORCEMENT effectue une validation de liste autorisée sur les méthodes ObjectMessage.getObject() et ObjectMessage.setObject().

Format du fichier de liste autorisée

Voici les principales caractéristiques du format du fichier de liste autorisée:

- Le fichier de liste autorisée est dans le codage de fichier de plateforme par défaut avec des fins de ligne appropriées à la plateforme.

  Remarque : Si un fichier de liste autorisée est utilisé, ce fichier est toujours écrit et lu à l'aide du codage de fichier par défaut de la machine virtuelle Java.

  Cela est correct si le fichier de liste autorisée est généré de l'une des manières suivantes:

  – z/OS Généré par une application autonome s'exécutant sous z/OS et utilisé par d'autres applications autonomes s'exécutant également sous z/OS.
  – Généré par une application s'exécutant dans WebSphere Application Server sur n'importe quelle plateforme et utilisé par une autre instance de WebSphere Application Server.
  – Multi Généré par une application autonome s'exécutant sous IBM MQ for Multiplatformset utilisé par d'autres applications autonomes s'exécutant sous IBM MQ for Multiplatforms, ou par des applications s'exécutant dans WebSphere Application Server sur n'importe quelle plateforme.

  Toutefois, comme WebSphere Application Server utilise ASCII et qu'une machine virtuelle Java autonome utilise EBCDIC, des problèmes de codage de fichier se produisent si le fichier de liste autorisée est généré de l'une des manières suivantes:

  - Généré sur z/OS, puis utilisé par des applications autonomes s'exécutant sur une plateforme autre que z/OS ou par WebSphere Application Server.
  - Généré par WebSphere Application Server ou une application autonome s'exécutant sur une plateforme autre que z/OS, puis utilisé par une application autonome sur z/OS.
- Chaque ligne non vide contient un nom de classe qualifié complet. Les lignes vides sont ignorées.
- Les commentaires peuvent être inclus-tout ce qui suit un caractère'#', jusqu'à la fin de la ligne, est ignoré.
- Il existe un mécanisme de caractères génériques très basique:
  - '*'peut être le **dernier** élément d'un nom de classe.
  - '*'correspond à un élément **unique** d'un nom de classe, c'est-à-dire la classe, mais pas de partie du package.

  Ainsi, `com.ibm.mq.*` correspond à `com.ibm.mq.MQMessage` mais pas à `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  Les caractères génériques ne fonctionnent pas pour les classes du package par défaut qui est destiné aux classes sans nom de package explicite, de sorte qu'un nom de classe "*" est rejeté.
- Les fichiers de liste autorisée mal formatés, par exemple les fichiers qui contiennent une entrée telle que `com.ibm.mq.*.Message`, où le caractère générique n'est pas le dernier élément, génèrent une exception java.lang.IllegalArgumentException .
- Un fichier de liste autorisée vide a pour effet de désactiver totalement l'utilisation de ObjectMessage.

Format de la liste autorisée sous forme de liste séparée par des virgules

Le même mécanisme de caractères génériques est disponible pour une liste autorisée sous la forme d'une liste séparée par des virgules.

- Le caractère'*'peut être développé par le système d'exploitation s'il est spécifié sur une ligne de commande ou dans un script de shell ou un fichier de traitement par lots. Il peut donc nécessiter un traitement spécial.
- Le caractère de commentaire'#'n'est applicable que lorsqu'un fichier est spécifié. Si la liste autorisée est spécifiée sous la forme d'une liste de noms de classe séparés par des virgules, en supposant que le système d'exploitation ou l'interpréteur de commandes ne la traite pas, car il s'agit du caractère de commentaire par défaut dans de nombreux interpréteurs de commandes AIX and Linux , elle est traitée comme un caractère normal.

Quand l'inscription autorisée a-t-elle lieu?

La liste autorisée est lancée lorsque l'application exécute pour la première fois une méthode ObjectMessage setMessage() ou getMessage().

Les propriétés système sont évaluées, le fichier de liste autorisée est ouvert et en mode ENFORCEMENT, la liste des classes autorisées est chargée lorsque le mécanisme est initialisé. A ce stade, une entrée est écrite dans le fichier journal IBM MQ JMS de l'application.

Lorsque le mécanisme est initialisé, ses paramètres peuvent ne pas être modifiés. Comme le moment de l'initialisation n'est pas facile à prévoir, car il dépend du comportement de l'application. Les paramètres de propriété système et le contenu du fichier de liste autorisée doivent donc être considérés comme fixes à partir du moment où l'application est démarrée. Ne modifiez pas les propriétés ou le contenu du fichier de liste autorisée lorsque l'application est en cours d'exécution, car les résultats ne sont pas garantis.

Points à prendre en considération

La meilleure approche pour atténuer les risques inhérents au mécanisme de sérialisation Java consiste à explorer d'autres approches du transfert de données, telles que l'utilisation de JSON à la place de ObjectMessage. L'utilisation des mécanismes Advanced Message Security (AMS) peut renforcer la sécurité en s'assurant que les messages proviennent de sources dignes de confiance.

Si vous utilisez le mécanisme Java security manager avec votre application, vous devez accorder les droits suivants:

- FilePermission sur tout fichier de liste autorisée que vous utilisez, avec droit de lecture pour le mode ENFORCEMENT, droit d'écriture pour le mode DISCOVER.
- JMS 2.0 PropertyPermission (lecture) sur les propriétés **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover**et **com.ibm.mq.jms.allowlist.mode** .
- JM 3.0 PropertyPermission (lecture) sur les propriétés **com.ibm.mq.jakarta.jms.allowlist**, **com.ibm.mq.jakarta.jms.allowlist.discover**et **com.ibm.mq.jakarta.jms.allowlist.mode** .

Informations complémentaires

Pour plus d'informations sur les listes autorisées, voir «Configuration et utilisation d'une liste autorisée JMS ou Jakarta Messaging», à la page 139 et «Liste autorisée dans WebSphere Application Server», à la page 141 .

Concepts associés

«Exécution d'applications IBM MQ classes for JMS sous Java security manager», à la page 110
IBM MQ classes for JMS peut s'exécuter avec Java security manager activé. Pour exécuter des applications avec le Java security manager activé, vous devez configurer votre Java Virtual Machine (JVM) avec un fichier de configuration de règles approprié.

Configuration et utilisation d'une liste autorisée JMS ou Jakarta Messaging

Ces informations vous indiquent comment une liste autorisée fonctionne et comment vous en configurez une à l'aide de la fonctionnalité contenue dans IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging pour générer un fichier de liste autorisée, contenant une liste des types de ObjectMessages qu'une application peut traiter.

Avant de commencer

Important :

Partout où c'était possible, le terme *allowlist* a remplacé le terme *whitelist*. Cela inclut certains noms de propriété système Java mentionnés dans cette rubrique. Il n'est pas nécessaire de changer la configuration existante. Les noms de propriété système précédents fonctionnent également.

Avant de commencer cette tâche, assurez-vous d'avoir lu et compris «Liste autorisée des concepts», à la page 135

Pourquoi et quand exécuter cette tâche

Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

Une fois que vous avez activé la fonctionnalité de mise en liste autorisée, les IBM MQ classes for JMS l'utilisent de la manière suivante:

- Lorsqu'une application souhaite envoyer un ObjectMessage, elle peut le créer de l'une des deux manières suivantes:
  - Méthode Session.createObjectMessage(Serializable), transmettant l'objet qui doit être contenu dans le message.
  - Méthode Session.createObjectMessage(), pour créer un ObjectMessagevide, puis appeler ObjectMessage.setObject(Serializable) pour stocker l'objet à envoyer dans ObjectMessage.

  Lorsque les méthodes Session.createObjectMessage(Serializable) ou ObjectMessage.setObject(Serializable) sont appelées, les classes de JMS vérifient si l'objet transmis est d'un type mentionné dans la liste autorisée.

S'il s'agit d'un type mentionné, l'objet est sérialisé et stocké dans ObjectMessage. Toutefois, si l'objet est d'un type qui ne figure pas dans la liste autorisée, le IBM MQ classes for JMS émet une exception JMSException contenant le message:

```
JMSCC0052: Une exception s'est produite lors de la sérialisation de l'objet:
'java.io.InvalidClassException: < classe d'objet> ; la classe ne peut pas être sérialisée
ou désérialisé car il n'a pas été inclus dans la liste autorisée'< listeautorisée >'.
```

retour à l'application.

Important : Si l'exception est émise à partir de la méthode Session.createObjectMessage(Serializable), l'objet ObjectMessage n'est pas créé. De même, si l'exception JMSException est émise à partir de la méthode ObjectMessage.setObject(Serializable), l'objet n'est pas ajouté à ObjectMessage.

- Si une application reçoit un ObjectMessage, elle appelle la méthode ObjectMessage.getObject() pour obtenir l'objet qu'elle contient. Lorsque cette méthode est appelée, IBM MQ classes for JMS vérifie le type d'objet contenu dans ObjectMessage, pour voir si cet objet est d'un type spécifié dans la liste autorisée.

  Si tel est le cas, l'objet est désérialisé et renvoyé à l'application. Toutefois, si l'objet est d'un type qui ne figure pas dans la liste autorisée, le IBM MQ classes for JMS émet une exception JMSException contenant le message:

  ```
  JMSCC0053: Une exception s'est produite lors de la désérialisation d'un message:
  'java.io.InvalidClassException: < classe d'objet> ; la classe ne peut pas être
  sérialisé ou désérialisé car il n'a pas été inclus dans le
  allowlist'< listeautorisée >'.'.
  ```

  retour à l'application.

Par exemple, supposons que votre application contienne le code suivant pour envoyer un ObjectMessage contenant un objet de type java.net.URI:

```
java.net.URL testURL = new java.net.URL("https://www.ibm.com/");
ObjectMessage msg = session.createObjectMessage(testURL);
sender.send(msg);
```

Comme la liste autorisée n'est pas activée, l'application est en mesure d'insérer correctement le message dans la destination requise.

Si vous créez un fichier appelé `C:\allowlist.txt` contenant une entrée unique, `java.net.URL`, et que vous redémarrez l'application avec la propriété système Java définie:

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

la fonctionnalité de liste autorisée est activée. L'application est toujours en mesure de créer et d'envoyer le message ObjectMessage contenant un objet de type java.net.URI , car ce type est spécifié dans la liste autorisée.

Toutefois, si vous modifiez le fichier `allowlist.txt` de sorte qu'il contienne l'entrée unique `java.util.Calendar`, car la fonctionnalité de liste autorisée est toujours activée, lorsque l'application appelle:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS vérifiez la liste autorisée et constatez qu'elle ne contient pas d'entrée pour java.net.URI.

Par conséquent, une exception JMSException contenant le message JMSCC0052 est émise.

De même, supposons que vous disposiez d'une autre application qui reçoit ObjectMessages à l'aide de ce code:

```
ObjectMessage message = (ObjectMessage)receiver.receive(30000);
if (message != null) {
    Object messageBody = objectMessage.getObject();
    if (messageBody instanceof java.net.URI) {
      :    :    :    :    :    :    :
```

Si la mise en liste autorisée n'est pas activée, l'application peut recevoir des ObjectMessages qui contiennent un objet de n'importe quel type. L'application vérifie ensuite si l'objet est de type java.net.URL avant d'effectuer le traitement approprié.

Si vous démarrez maintenant l'application avec la propriété système Java :

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

définie, la fonctionnalité de liste autorisée est activée. Lorsque l'application appelle:

```
Object messageBody = objectMessage.getObject();
```

la méthode ObjectMessage.getObject() ne renvoie que les objets de type java.net.URL.

Si l'objet contenu dans ObjectMessage n'est pas de ce type, la méthode ObjectMessage.getObject() émet une exception JMSException contenant le message JMSCC0053 . L'application doit ensuite décider de ce qu'elle doit faire avec le message ; par exemple, le message peut être déplacé dans la file d'attente de rebut de ce gestionnaire de files d'attente.

L'application n'est renvoyée normalement que si l'objet dans ObjectMessage est de type java.net.URL.

Procédure

1. Exécutez l'application qui traite ObjectMessages, avec les propriétés système Java suivantes spécifiées:

   ```
   -Dcom.ibm.mq.jms.allowlist.discover=true
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Lorsque l'application s'exécute, le IBM MQ classes for JMS crée un fichier qui contient les types d'objets traités par l'application.
2. Une fois que l'application a traité un échantillon représentatif d' ObjectMessages sur une période donnée, arrêtez-le.

   Le fichier de liste autorisée contient désormais une liste de tous les types d'objets contenus dans les ObjectMessages que l'application a traités lors de son exécution.

   Si vous avez exécuté l'application pendant un temps suffisant, cette liste inclut tous les types possibles d'objets contenus dans ObjectMessages que l'application est susceptible de traiter.
3. Redémarrez l'application avec l'ensemble de propriétés système suivant:

   ```
   -Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
   ```

   Cela active la mise sur liste autorisée, et si IBM MQ classes for JMS détecte un ObjectMessage d'un type qui ne figure pas dans la liste autorisée, une exception JMSException contenant le message JMSCC0052 ou JMSCC0053 est émise.

Liste autorisée dans WebSphere Application Server

Comment utiliser la IBM MQ classes for JMS liste autorisée dans WebSphere Application Server.

Important :

Partout où c'était possible, le terme *allowlist* a remplacé le terme *whitelist*. Cela inclut certains noms de propriété système Java mentionnés dans cette rubrique. Il n'est pas nécessaire de changer la configuration existante. Les noms de propriété système précédents fonctionnent également.

Vous devez vous assurer que votre installation WebSphere Application Server inclut une version de l'adaptateur de ressources IBM MQ qui prend en charge la mise sur liste autorisée.

Pour plus d'informations sur l'utilisation des deux produits, voir «Utilisation conjointe de IBM MQ et de WebSphere Application Server», à la page 511 .

IBM MQ 9.0.0 Fix Pack 1 et les versions suivantes incluent la fonctionnalité appropriée.

Une fois le serveur d'applications mis à jour, vous pouvez utiliser les propriétés système Java :

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

décrite dans «Configuration et utilisation d'une liste autorisée JMS ou Jakarta Messaging», à la page 139.

Remarque : Vous devez définir les propriétés système Java en tant qu'arguments JVM génériques, sur le Java Virtual Machine utilisé pour exécuter le serveur d'applications et le serveur d'applications redémarré pour que les modifications soient prises en compte.

Pour plus d'informations, voir la section *Arguments JVM génériques* dans Paramètres de la machine virtuelle Java .

Pour définir les propriétés, accédez à la fenêtre Java Virtual Machine dans *Définitions de processus* et entrez l'argument approprié.

Le paramètre suivant:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

fait en sorte que le serveur d'applications utilise la liste autorisée *votreIdutilisateur_MyObject*. Seuls les objets du type sont traités par le serveur d'applications.

Les paramètres suivants:

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

configurer le serveur d'applications pour qu'il utilise le mode *Discover* et enregistrer les détails des ObjectMessages JMS que le serveur d'applications traite dans le fichier `C:\allowlist.txt`

Le paramètre suivant:

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

entraîne le serveur d'applications à charger le fichier `C:/allowlist.txt`et à utiliser les informations de ce fichier pour déterminer la liste autorisée.

Concepts associés

«Exécution d'applications IBM MQ classes for JMS sous Java security manager», à la page 110
IBM MQ classes for JMS peut s'exécuter avec Java security manager activé. Pour exécuter des applications avec le Java security manager activé, vous devez configurer votre Java Virtual Machine (JVM) avec un fichier de configuration de règles approprié.

Conversions de chaînes de caractères dans IBM MQ classes for JMS

Les IBM MQ classes for JMS utilisent CharsetEncoders et CharsetDecoders directement pour la conversion de chaînes de caractères. Le comportement par défaut de la conversion de chaîne de caractères peut être configuré avec deux propriétés système. Le traitement des messages contenant des caractères non mappables peut être configuré via des propriétés de message pour définir l'action UnmappableCharacteret les octets de remplacement.

Avant IBM MQ 8.0, les conversions de chaîne dans IBM MQ classes for JMS étaient effectuées en appelant les méthodes `java.nio.charset.Charset.decode(ByteBuffer)` et `Charset.encode(CharBuffer)` .

L'utilisation de l'une ou l'autre de ces méthodes entraîne le remplacement par défaut (REPLACE) de données syntaxiquement incorrectes ou non traduisibles. Ce comportement peut masquer les erreurs dans les applications et entraîner des caractères inattendus, par exemple ?, dans les données traduites.

Depuis la IBM MQ 8.0, pour détecter ces problèmes plus tôt et plus efficacement, les IBM MQ classes for JMS utilisent directement CharsetEncoders et CharsetDecoders et configurent explicitement le traitement des données syntaxiquement incorrectes et non traduisibles. Le comportement par défaut consiste à REPORT de tels problèmes en émettant un `MQException`approprié.

Configuration

La conversion de UTF-16 (la représentation de caractères utilisée dans Java) en un jeu de caractères natif, tel que UTF-8, est appelée *codage*, tandis que la conversion dans la direction opposée est appelée *décodage*.

Le décodage utilise le comportement par défaut pour `CharsetDecoders`, qui signale les erreurs en émettant une exception.

Un paramètre est utilisé pour spécifier un paramètre `java.nio.charset.CodingErrorAction` afin de contrôler le traitement des erreurs lors du codage et du décodage. Un autre paramètre est utilisé pour contrôler le ou les octets de remplacement lors du codage. La chaîne de remplacement Java par défaut sera utilisée dans les opérations de décodage.

UnmappableCharacterParamètres des octets d'action et de remplacement dans IBM MQ classes for JMS

Depuis la IBM MQ 8.0, les deux propriétés suivantes sont disponibles pour définir l'action UnmappableCharacteret les octets de remplacement. Les définitions de constante appropriées se trouvent dans `com.ibm.msg.client.wmq.WMQConstants`.

ACTION JMS_IBM_UNMAPPABLE_ACTION
Définit ou obtient le `CodingErrorAction` à appliquer lorsqu'un caractère ne peut pas être mappé dans une opération de codage ou de décodage.

Vous devez définir ce paramètre en tant que `CodingErrorAction.{REPLACE|REPORT|IGNORE}.toString()` comme suit:

```
public static final String JMS_IBM_UNMAPPABLE_ACTION = "JMS_IBM_Unmappable_Action";
```

JMS_IBM_UNMAPPABLE_REPLACEMENT
Définit ou extrait les octets de remplacement à appliquer lorsqu'un caractère ne peut pas être mappé dans une opération de codage.

La chaîne de remplacement Java par défaut est utilisée dans les opérations de décodage.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Les propriétés `JMS_IBM_UNMAPPABLE_ACTION` et `JMS_IBM_UNMAPPABLE_REPLACEMENT` peuvent être définies sur des destinations ou des messages. Une valeur définie sur un message remplace la valeur définie sur la destination à laquelle le message est envoyé.

Notez que `JMS_IBM_UNMAPPABLE_REPLACEMENT` doit être défini sous la forme d'un octet unique.

Propriétés système pour la définition des valeurs par défaut du système

Depuis la IBM MQ 8.0, les deux propriétés système Java suivantes sont disponibles pour configurer le comportement par défaut de la conversion de chaîne de caractères.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Indique l'action à effectuer pour les données non traduisibles lors du codage et du décodage. La valeur peut être `REPORT`, `REPLACE`ou `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Définit ou obtient les octets de remplacement à appliquer lorsqu'un caractère ne peut pas être mappé dans une opération de codage. La chaîne de remplacement Java par défaut est utilisée dans les opérations de décodage.

Pour éviter toute confusion entre les représentations de caractères Java et d'octets natifs, vous devez spécifier `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` en tant que nombre décimal représentant l'octet de remplacement dans le jeu de caractères natifs.

Par exemple, la valeur décimale de ?, en tant qu'octet natif, est 63 si le jeu de caractères natif est basé sur ASCII, tel que ISO-8859-1, et 111 si le jeu de caractères natif est EBCDIC.

Remarque : Notez que si les zones **unmappableAction** ou **unMappableReplacement** sont définies pour un objet MQMD ou MQMessage, les valeurs de ces zones sont prioritaires sur les propriétés système Java . Cela permet de remplacer les valeurs spécifiées par les propriétés système Java pour chaque message si nécessaire.

Concepts associés

«Conversions de chaînes de caractères dans IBM MQ classes for Java», à la page 361
Les IBM MQ classes for Java utilisent CharsetEncoders et CharsetDecoders directement pour la conversion de chaînes de caractères. Le comportement par défaut de la conversion de chaîne de caractères peut être configuré avec deux propriétés système. Le traitement des messages contenant des caractères non mappables peut être configuré via `com.ibm.mq.MQMD`.

Ecriture d'applications IBM MQ classes for JMS/Jakarta Messaging

Après une brève introduction au modèle JMS , cette section fournit des conseils détaillés sur la façon d'écrire des applications IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging .

Pourquoi et quand exécuter cette tâche

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Concepts associés

IBM MQ classes for Jakarta Messaging: présentation

Le modèle JMS et Jakarta Messaging

Le modèle JMS et Jakarta Messaging définit un ensemble d'interfaces que les applications Java peuvent utiliser pour effectuer des opérations de messagerie. IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont des fournisseurs de messagerie. Ils définissent comment les objets JMS et Jakarta Messaging sont liés aux concepts IBM MQ . Les spécifications JMS et Jakarta Messaging s'attendent à ce que certains objets JMS et Jakarta Messaging soient des objets gérés.

JMS 2.0 IBM MQ 8.0 a ajouté la prise en charge de la version JMS 2.0 de la norme JMS , qui a introduit une API simplifiée, tout en conservant l'API classique, à partir de JMS 1.1.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 continue de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API JMS 2.0 et de l'API Jakarta Messaging 3.0 dans la même application n'est pas prise en charge.

Remarque : Pour Jakarta Messaging 3.0, le contrôle de la spécification JMS passe de Oracle à Java Community Process. Toutefois, Oracle conserve le contrôle du nom "javax", qui est utilisé dans d'autres technologies Java qui n'ont pas été déplacées vers le processus de communauté Java . Ainsi, alors que Jakarta Messaging 3.0 est fonctionnellement équivalent à JMS 2.0 , il existe des différences de dénomination:

- Le nom officiel de la version 3.0 est Jakarta Messaging et non Java Message Service.
- Les noms de pack et de constante sont précédés de `jakarta` et non de `javax`. Par exemple, dans JMS 2.0 , la connexion initiale à un fournisseur de messagerie est un objet `javax.jms.Connection` et dans Jakarta Messaging 3.0 , il s'agit d'un objet `jakarta.jms.Connection` .

JMS 2.0 Les packages javax.jms définissent les interfaces JMS et un fournisseur JMS implémente ces interfaces pour un produit de messagerie spécifique. IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ.

JMS 3.0 Les packages jakarta.jms définissent les interfaces Jakarta Messaging et un fournisseur Jakarta Messaging implémente ces interfaces pour un produit de messagerie spécifique. IBM MQ classes for Jakarta Messaging est un fournisseur Jakarta Messaging qui implémente les interfaces Jakarta Messaging pour IBM MQ.

Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

API simplifiée

JMS 2.0 a introduit l'API simplifiée, tout en conservant les interfaces spécifiques au domaine et indépendantes du domaine d' JMS 1.1. L'API simplifiée réduit le nombre d'objets nécessaires à l'envoi et à la réception de messages et se compose des interfaces ci-dessous :

ConnectionFactory
ConnectionFactory est un objet géré qui est utilisé par un client JMS pour créer une connexion. Cette interface est également utilisée dans l'API classique.

JMSContexte
Cet objet combine les objets Connection et Session de l'API classicCd. Les objets JMSContext peuvent être créés à partir d'autres objets JMSContext, auquel cas la connexion sous-jacente est dupliquée.

JMSExpéditeur
Un objet JMSProducer est créé par un objet JMSContext et il utilisé pour envoyer des messages à une file d'attente ou à une rubrique. L'objet JMSProducer entraîne la création des objets nécessaires à l'envoi du message.

JMSConsommateur
Un objet JMSConsumer est créé par un objet JMSContext et il est utilisé pour recevoir des messages provenant d'une rubrique ou d'une file d'attente.

L'API simplifiée a un certain nombre d'effets :

- L'objet JMSContext démarre toujours automatiquement la connexion sous-jacente.
- Les objets JMSProducers et JMSConsumers peuvent maintenant traiter directement le corps des messages, sans devoir détenir l'objet message complet, par le biais de la méthode getBody du message.
- Les propriétés du message peuvent être définies sur l'objet JMSProducer à l'aide du chaînage de méthodes avant d'envoyer un 'corps', c'est-à-dire le contenu des messages. L'objet JMSProducer traite la création de tous les objets nécessaires à l'envoi du message. A l'aide de JMS 2.0, les propriétés sont définies et le message suivant est envoyé :

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

JMS 2.0 a également introduit des abonnements partagés dans lesquels les messages peuvent être partagés entre plusieurs consommateurs. Tous les abonnements JMS 1.1 sont traités en tant qu'abonnements non partagés.

API classique

Les interfaces JMS principales qui constituent l'API classique sont répertoriées ci-dessous :

Destination
Une destination représente l'endroit où une application envoie des messages, l'endroit d'où elle en reçoit, ou les deux.

ConnectionFactory
Un objet ConnectionFactory encapsule un ensemble de propriétés de configuration pour une connexion. Une application utilise une fabrique de connexions pour créer une connexion.

Connexion
Un objet Connection encapsule la connexion active d'une application sur un serveur de messagerie. Une application utilise une connexion pour créer des sessions.

Session
Une session est un contexte à unité d'exécution unique pour l'envoi et la réception de messages. Une application utilise ensuite une session pour créer des messages, des expéditeurs de message et des consommateurs de message. Une session est transactionnelle ou non transactionnelle.

Message
Un objet Message encapsule un message envoyé ou reçu par une application.

MessageProducer
Une application utilise un expéditeur de message pour envoyer des messages vers une destination.

MessageConsumer
Une application utilise un consommateur de message pour la réception des messages envoyés vers une destination.

La Figure 9, à la page 146 présente ces objets et leurs relations.

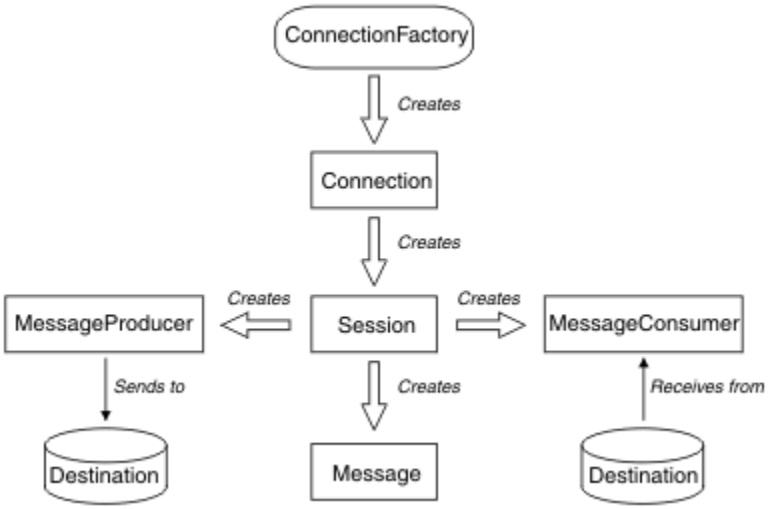


Figure 9. Objets JMS et leurs relations

Un objet Destination, ConnectionFactory ou Connection peut être utilisé simultanément par différentes unités d'exécution d'une application comportant plusieurs unités d'exécution, mais un objet Session, MessageProducer ou MessageConsumer ne peut pas être utilisé par des unités d'exécution différentes. La manière la plus simple de s'assurer qu'un objet Session, MessageProducer ou MessageConsumer n'est pas utilisé simultanément consiste à créer un objet Session distinct pour chaque unité d'exécution.

Domaines de messagerie

JMS prend en charge deux styles de messagerie:

- Messagerie point-à-point
- Messagerie de type publication/abonnement

Ces deux styles de messagerie sont également appelés *domaines de messagerie* et ils peuvent être combinés dans une application. Dans le domaine point-à-point, une destination est une file d'attente et, dans le domaine de publication/abonnement, une destination est une rubrique.

Avec les versions de JMS antérieures à JMS 1.1, la programmation pour le domaine point-à-point utilise un ensemble d'interfaces et de méthodes et la programmation pour le domaine de publication/

abonnement utilise un autre ensemble. Les deux ensembles sont similaires, mais distincts. A partir de JMS 1.1, vous pouvez utiliser un ensemble commun d'interfaces et de méthodes qui prennent en charge les deux domaines de messagerie. Les interfaces communes fournissent une vue indépendante du domaine de chaque domaine de messagerie. Le Tableau 15, à la page 147 affiche la liste des interfaces indépendantes du domaine JMS et leurs interfaces propres au domaine correspondantes.

Tableau 15. Les interfaces indépendantes du domaine et spécifiques du domaine JMS

Interfaces indépendantes du domaine	**Interfaces propres au domaine pour le domaine point-à-point**	**Interfaces propres au domaine pour le domaine de publication/ abonnement**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connexion	QueueConnection	TopicConnection
Destination	File d'attente	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 prend en charge à la fois les interfaces spécifiques au domaine JMS 1.1 et l'API simplifiée de JMS 2.0. IBM MQ classes for JMS 2.0 peut donc être utilisé pour gérer des applications existantes, y compris pour développer de nouvelles fonctions dans des applications existantes.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 prend en charge les versions Jakarta Messaging des mêmes interfaces et est recommandé pour le développement de nouvelles applications.

Dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, les objets JMS sont liés aux concepts IBM MQ de la manière suivante:

- Un objet Connection possède des propriétés issues des propriétés de la fabrique de connexions qui a été utilisée pour créer la connexion. Ces propriétés contrôlent comment une application se connecte à un gestionnaire de files d'attente. Des exemples de ces propriétés sont le nom du gestionnaire de files d'attente et, pour une application qui se connecte au gestionnaire de files d'attente en mode client, le nom d'hôte ou l'adresse IP du système sur lequel s'exécute le gestionnaire de files d'attente.
- Un objet Session encapsule le descripteur de connexion IBM MQ qui définit la portée transactionnelle de la session.
- Un objet MessageProducer et un objet MessageConsumer encapsule chacun un descripteur d'objet IBM MQ.

Lorsque vous utilisez IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, toutes les règles normales de IBM MQ s'appliquent. Sachez, en particulier, qu'une application peut envoyer un message à une file d'attente éloignée, mais elle peut uniquement recevoir un message d'une file d'attente qui appartient au gestionnaire de files d'attente auquel l'application est connectée.

La spécification JMS s'attend à ce que les objets ConnectionFactory et Destination soient des objets gérés. Un administrateur crée et gère les objets gérés dans un référentiel central et une application JMS extrait ces objets à l'aide de l'interface JNDI (Java Naming and Directory Interface).

Dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, l'implémentation de l'interface Destination est une superclasse abstraite de Queue et Topic, et donc une instance de Destination est soit un objet Queue, soit un objet Topic. Les interfaces indépendantes du domaine traitent une file d'attente ou une rubrique en tant que destination. Le domaine de messagerie pour un objet MessageProducer ou MessageConsumer varie selon que la destination est une file d'attente ou une rubrique.

Par conséquent, dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging , les objets des types suivants peuvent être des objets gérés:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- File d'attente
- Topic
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Concepts associés

Interfaces de langage IBM MQ Java

«Création et configuration de fabriques de connexions et de destinations», à la page 211

Une application IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging peut créer des fabriques de connexions et des destinations en les extrayant en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), à l'aide des extensions IBM JMS ou à l'aide des extensions IBM MQ JMS . Une application peut également utiliser les extensions IBM JMS ou IBM MQ JMS pour définir les propriétés des fabriques de connexions et des destinations.

Messages JMS

Les messages JMS sont composés d'un en-tête, de propriétés et d'un corps. JMS définit cinq types de corps de message.

Les messages JMS sont composés des éléments suivants:

En-tête
Tous les messages prennent en charge le même ensemble de zones d'en-tête. Les zones d'en-tête contiennent des valeurs utilisées par les clients et les fournisseurs pour identifier et acheminer les messages.

Propriétés
Chaque message contient une fonction intégrée permettant de prendre en charge les valeurs de propriété définies par l'application. Les propriétés fournissent un mécanisme efficace pour filtrer les messages définis par l'application.

Corps
JMS définit cinq types de corps de message qui couvrent la majorité des styles de messagerie actuellement utilisés:

Flux
Flux de valeurs primitives Java . Il est rempli et lu séquentiellement.

Mappe
Un ensemble de paires nom-valeur, où les noms sont des chaînes et les valeurs sont des types primitifs Java . Les entrées sont accessibles de manière séquentielle ou aléatoire par nom. L'ordre des entrées n'est pas défini.

Texte
Message contenant un java.lang.String.

Objet
Message contenant un objet Java sérialisable

Octets
Flux d'octets non interprétés. Ce type de message permet de coder littéralement un corps pour qu'il corresponde à un format de message existant.

La zone d'en-tête JMSCorrelationID permet de lier un message à un autre. Il lie généralement un message de réponse à son message de demande. JMSCorrelationID peut contenir un ID message spécifique au fournisseur, une chaîne spécifique à l'application ou une valeur byte [ ] native du fournisseur.

Sélecteurs de message dans JMS
Les messages peuvent contenir des valeurs de propriété définies par l'application. Une application peut utiliser des sélecteurs de message pour avoir des messages de filtre de fournisseur JMS .

Un message contient une fonction intégrée permettant de prendre en charge les valeurs de propriété définies par l'application. En effet, cela fournit un mécanisme permettant d'ajouter des zones d'en-tête spécifiques à une application à un message. Les propriétés permettent à une application, à l'aide de sélecteurs de message, de faire en sorte qu'un fournisseur JMS sélectionne ou filtre les messages pour son compte, en utilisant des critères spécifiques à l'application. Les propriétés définies par l'application doivent respecter les règles suivantes:

- Les noms de propriété doivent respecter les règles d'un identificateur de sélecteur de message.
- Les valeurs de propriété peuvent être Boolean, byte, short, int, long, float, double et String.
- Les préfixes de nom JMSX et JMS_ sont réservés.

Les valeurs de propriété sont définies avant l'envoi d'un message. Lorsqu'un client reçoit un message, les propriétés du message sont en lecture seule. Si un client tente de définir des propriétés à ce stade, une exception MessageNotWriteableException est émise. Si clearProperties est appelée, les propriétés peuvent désormais être lues et écrites.

Une valeur de propriété peut dupliquer une valeur dans un corps de message. JMS ne définit pas de règle pour ce qui peut être transformé en propriété. Toutefois, les développeurs d'applications doivent savoir que les fournisseurs JMS traitent probablement les données d'un corps de message de manière plus efficace que les données des propriétés de message. Pour de meilleures performances, les applications doivent utiliser les propriétés de message uniquement lorsqu'elles doivent personnaliser un en-tête de message. La raison principale de cette opération est la prise en charge de la sélection de messages personnalisés.

Un sélecteur de message JMS permet à un client de spécifier les messages qui l'intéressent à l'aide de l'en-tête de message. Seuls les messages dont les en-têtes correspondent au sélecteur sont distribués.

Les sélecteurs de message ne peuvent pas faire référence à des valeurs de corps de message.

Un sélecteur de message correspond à un message lorsque le sélecteur a la valeur true lorsque les valeurs de zone d'en-tête de message et de propriété sont remplacées par leurs identificateurs correspondants dans le sélecteur.

Un sélecteur de message est une chaîne dont la syntaxe est basée sur un sous-ensemble de la syntaxe d'expression conditionnelle SQL92 . L'ordre dans lequel un sélecteur de message est évalué est de gauche à droite dans un niveau de priorité. Vous pouvez utiliser des parenthèses pour modifier cet ordre. Les littéraux de sélecteur et les noms d'opérateur prédéfinis sont écrits ici en majuscules ; toutefois, ils ne sont pas sensibles à la casse.

Contenu d'un sélecteur de message

Un sélecteur de message peut contenir:

- Littéraux
  - Un littéral chaîne est placé entre guillemets. Un guillemet double représente un guillemet. Par exemple,'literal'et'literal''s'. Comme les littéraux chaîne Java , ils utilisent le codage de caractères Unicode.
  - Un littéral numérique exact est une valeur numérique sans séparateur décimal, telle que 57, -957 et +62. Les nombres de la plage Java long sont pris en charge.
  - Un littéral numérique approximatif est une valeur numérique dans la notation scientifique, telle que 7E3 ou -57.9E2, ou une valeur numérique avec une décimale, telle que 7., -95.7ou +6.2. Les nombres de la plage Java double sont pris en charge.
  - Les littéraux booléens TRUE et FALSE.
- Identificateurs :

  - Un identificateur est une séquence de longueur illimitée de Java lettres et de Java chiffres, dont le premier doit être une lettre Java . Une lettre est un caractère pour lequel la méthode Character.isJavaLetter renvoie la valeur true. Cela inclut _ et $. Une lettre ou un chiffre est un caractère pour lequel la méthode Character.isJavaLetterOrDigit renvoie la valeur true.
  - Les identificateurs ne peuvent pas être les noms NULL, TRUE ou FALSE.
  - Les identificateurs ne peuvent pas être NOT, AND, OR, BETWEEN, LIKE, IN ou IS.
  - Les identificateurs sont des références de zone d'en-tête ou des références de propriété.
  - Les identificateurs sont sensibles à la casse.
  - Les références de zone d'en-tête de message sont limitées à:

    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID
    - JMSType

    Les valeurs JMSMessageID, JMSTimestamp, JMSCorrelationIDet JMSType peuvent être null et, si tel est le cas, sont traitées comme une valeur NULL.
  - Tout nom commençant par JMSX est un nom de propriété défini par JMS.
  - Tout nom commençant par JMS_ est un nom de propriété spécifique au fournisseur.
  - Tout nom qui ne commence pas par JMS est un nom de propriété propre à l'application. S'il existe une référence à une propriété qui n'existe pas dans un message, sa valeur est NULL. S'il existe, sa valeur est la valeur de propriété correspondante.
- Les espaces sont identiques à ceux définis pour Java: espace, tabulation horizontale, saut de page et caractère de fin de ligne.
- Expressions :
  - Un sélecteur est une expression conditionnelle. Un sélecteur qui a la valeur true correspond ; un sélecteur qui a la valeur false ou unknown ne correspond pas.
  - Les expressions arithmétiques sont composées d'elles-mêmes, d'opérations arithmétiques, d'identificateurs (avec une valeur qui est traitée comme un littéral numérique) et de littéraux numériques.
  - Les expressions conditionnelles sont composées d'elles-mêmes, d'opérations de comparaison et d'opérations logiques.
- La méthode standard bracketing (), qui permet de définir l'ordre dans lequel les expressions sont évaluées, est prise en charge.
- Opérateurs logiques dans l'ordre de priorité: NOT, AND, OR.
- Opérateurs de comparaison: =, >, > =, <, < =, < > (non égal à).
  - Seules les valeurs de même type peuvent être comparées. Une exception est qu'il est valide pour comparer les valeurs numériques exactes et les valeurs numériques approximatives. (La conversion de type requise est définie par les règles de la promotion numérique Java .) S'il y a une tentative de comparaison de différents types, le sélecteur est toujours faux.
  - La comparaison de chaînes et de booléens est limitée à = et < >. Deux chaînes sont égales uniquement si elles contiennent la même séquence de caractères.
- Opérateurs arithmétiques dans l'ordre de priorité:
  - +,-unaire.
  - *, /, multiplication et division.
  - +,-, addition et soustraction.

  - Les opérations arithmétiques sur une valeur NULL ne sont pas prises en charge. S'ils sont tentés, le sélecteur complet est toujours faux.
  - Les opérations arithmétiques doivent utiliser la promotion numérique Java .
- Opérateur de comparaison arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 :
  - L'âge ENTRE 15 et 19 est équivalent à l'âge > = 15 ET à l'âge < = 19.
  - L'âge NON ENTRE 15 et 19 est équivalent à l'âge < 15 OU à l'âge > 19.
  - Si l'une des expressions d'une opération BETWEEN est NULL, la valeur de l'opération est false. Si l'une des expressions d'une opération NOT BETWEEN est NULL, la valeur de l'opération est true.
- identifier [NOT] IN (string-literal1, string-literal2, ...) opérateur de comparaison dans lequel l'identificateur a une valeur de chaîne ou NULL.
  - Pays IN ('UK','US','France') est vrai pour'UK'et faux pour'Pérou'. Elle est équivalente à l'expression (Country = 'UK') OR (Country = 'US') OR (Country = 'France').
  - Le pays NON IN ('UK','US','France') est faux pour'UK'et vrai pour'Pérou'. Elle est équivalente à l'expression NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')).
  - Si l'identificateur d'une opération IN ou NOT IN est NULL, la valeur de l'opération est inconnue.
- identificateur [NOT] LIKE pattern-value [ESCAPE-character] opérateur de comparaison, où l'identificateur a une valeur de chaîne. pattern-value est un littéral chaîne, où _ représente un caractère unique et% représente une séquence de caractères (y compris la séquence vide). Tous les autres personnages se tiennent pour eux-mêmes. Le caractère d'échappement facultatif est un littéral chaîne de caractères unique, avec un caractère utilisé pour échapper la signification spéciale des caractères _ et% dans la valeur de modèle.
  - Le numéro de téléphone LIKE'12%3'est vrai pour 123 et 12993 et faux pour 1234.
  - word LIKE'l_se'est vrai pour "lose" et faux pour "loose".
  - souligné comme'_ %' escape'\'est vrai pour "_foo" et faux pour "bar".
  - phone NOT LIKE'12%3'est false pour 123 et 12993 et true pour 1234.
  - Si l'identificateur d'une opération LIKE ou NOT LIKE est NULL, la valeur de l'opération est inconnue.
- L'opérateur de comparaison IS NULL de l'identificateur teste une valeur de zone d'en-tête NULL ou une valeur de propriété manquante.
  - prop_name IS NULL.
- L'opérateur de comparaison d'identificateur IS NOT NULL teste l'existence d'une valeur de zone d'en-tête non nulle ou d'une valeur de propriété.
  - prop_name IS NOT NULL.

Exemple de sélecteur de message

Le sélecteur de message suivant sélectionne les messages dont le type de message est voiture, la couleur bleue et le poids est supérieur à 2500 lbs:

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

Valeurs de propriété NULL

Comme indiqué dans la liste précédente, les valeurs de propriété peuvent être NULL. L'évaluation des expressions de sélecteur qui contiennent des valeurs NULL est définie par la sémantique SQL 92 NULL. La liste suivante donne une brève description de ces sémantiques:

- SQL traite une valeur NULL comme inconnue.
- La comparaison ou l'arithmétique avec une valeur inconnue donne toujours une valeur inconnue.
- L'opérateur IS NULL convertit une valeur inconnue en valeur TRUE.

- L'opérateur IS NOT NULL convertit une valeur inconnue en valeur FALSE.

Comportement spécial de JMSMessageID et JMSCorrelationID

Les classes IBM MQ pour JMS contiennent des optimisations lors de la sélection de messages dans une file d'attente en fonction de JMSMessageID ou JMSCorrelationID.

Si une application spécifie un sélecteur de la forme:

JMSMessageID= 'ID:*id_message*'

où *id_message* est une chaîne contenant un identificateur de message IBM MQ standard, puis les classes IBM MQ pour JMS utilisent **MatchOption** MQMO_MATCH_MSG_ID pour obtenir le message avec l'identificateur de message spécifié.

Par exemple, pour obtenir un message avec l'identificateur de message 414D51207061756C745639314C545320C57C1A5F25ECE602 à partir d'une file d'attente, une application doit utiliser le sélecteur de message suivant:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

De même, si l'application spécifie un sélecteur dont le format est le suivant:

JMSCorrelationID = 'ID:*id_corrélation*'

où *correlation_id* est une chaîne contenant un identificateur de corrélation IBM MQ standard, les classes IBM MQ for JMS utilisent **MatchOption** MQMO_MATCH_CORREL_ID pour extraire de la file d'attente le message avec l'identificateur de corrélation spécifié.

Dans l'exemple suivant, un sélecteur de message est utilisé pour obtenir un message dont l'identificateur de corrélation est 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Si un sélecteur de message contient une valeur de zéro pour *message_id* ou *correlation_id*, il correspond à n'importe quel message de la file d'attente. Par exemple, si une application utilise le sélecteur:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

Tout message de la file d'attente est considéré comme une correspondance et renvoyé à l'application.

Pour plus d'informations sur MQMO_MATCH_MSG_ID et MQMO_MATCH_CORREL_ID **MatchOptions**, voir MatchOptions (MQLONG).

Restrictions

Bien que SQL prenne en charge la comparaison décimale fixe et l'arithmétique, les sélecteurs de message JMS ne le font pas. C'est pourquoi les littéraux numériques exacts sont limités à ceux sans décimale. C'est également la raison pour laquelle il existe des valeurs numériques avec une décimale comme représentation alternative pour une valeur numérique approximative.

Les commentaires SQL ne sont pas pris en charge.

Mappage de messages JMS en messages IBM MQ
Les messages IBM MQ sont composés d'un descripteur de message, d'un en-tête MQRFH2 facultatif et d'un corps. Le contenu d'un message JMS est partiellement mappé et partiellement copié dans un message IBM MQ .

Cette rubrique décrit comment la structure de message JMS décrite dans la première partie de cette section est mappée à un message IBM MQ . Il est intéressant pour les programmeurs qui souhaitent transmettre des messages entre les applications JMS et IBM MQ traditionnelles. Il est également intéressant pour les personnes qui souhaitent manipuler les messages transmis entre deux applications JMS , par exemple, dans une implémentation IBM Integration Bus .

Cette section ne s'applique pas si une application utilise une connexion en temps réel à un courtier. Lorsqu'une application utilise une connexion en temps réel, toutes les communications sont effectuées directement sur TCP/IP; aucune file d'attente ou message IBM MQ n'est impliqué.

Les messages IBM MQ sont composés de trois composants:

- Le descripteur de message IBM MQ (MQMD)
- Un en-tête IBM MQ MQRFH2
- Corps du message.

MQRFH2 est facultatif et son inclusion dans un message sortant est régie par l'indicateur TARGCLIENT dans la classe de destination JMS . Vous pouvez définir cet indicateur à l'aide de l'outil d'administration d' IBM MQ JMS . Etant donné que MQRFH2 contient des informations spécifiques à JMS, incluez-le toujours dans le message lorsque l'expéditeur sait que la destination de réception est une application JMS . Normalement, omettez MQRFH2 lors de l'envoi d'un message directement à une application nonJMS . En effet, une telle application n'attend pas de MQRFH2 dans son message IBM MQ .

Si un message entrant n'a pas d'en-tête MQRFH2 , l'objet Queue ou Topic dérivé de la zone d'en-tête JMSReplyTo du message, par défaut, a cet indicateur défini de sorte qu'un message de réponse envoyé à la file d'attente ou à la rubrique n'ait pas non plus d'en-tête MQRFH2 . Vous pouvez désactiver ce comportement d'inclusion d'un en-tête MQRFH2 dans un message de réponse uniquement si le message d'origine comporte un en-tête MQRFH2 , en définissant la propriété TARGCLIENTMATCHING de la fabrique de connexions sur NO.

La Figure 10, à la page 153 montre comment la structure d'un message JMS est transformée en message IBM MQ et à nouveau:

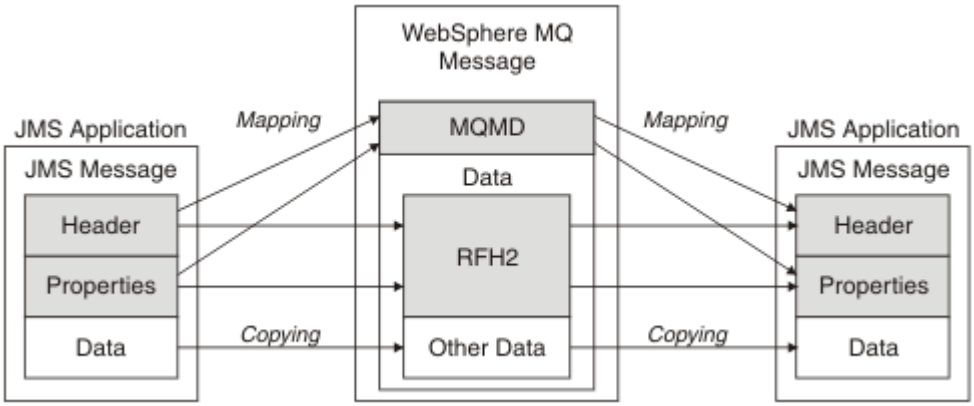


Figure 10. Comment les messages sont transformés entre JMS et IBM MQ à l'aide de l'en-tête MQRFH2

Les structures sont transformées de deux manières:

Mappage
Lorsque le MQMD inclut une zone équivalente à la zone JMS , la zone JMS est mappée à la zone MQMD. Des zones MQMD supplémentaires sont exposées en tant que propriétés JMS , car une application JMS peut avoir besoin d'obtenir ou de définir ces zones lors de la communication avec une application nonJMS .

Copie en cours
Lorsqu'il n'existe pas d'équivalent MQMD, une zone ou une propriété d'en-tête JMS est transmise, éventuellement transformée, en tant que zone dans MQRFH2.

L'en-tête MQRFH2 et JMS
Cette collection de rubriques décrit l'en-tête MQRFH version 2, qui contient des données spécifiques à JMSassociées au contenu du message. L'en-tête MQRFH version 2 est extensible et peut également contenir des informations supplémentaires qui ne sont pas directement associées à JMS. Toutefois, cette section ne couvre que son utilisation par JMS.Pour une description complète, voir MQRFH2 -Règles et en-tête de formatage 2.

Il y a deux parties de l'en-tête, une partie fixe et une partie variable.

Partie fixe
La partie fixe est modélisée sur le modèle d'en-tête *standard* IBM MQ et comprend les zones suivantes:

StrucId (MQCHAR4)
Identificateur de structure.

Doit être MQRFH_STRUC_ID (valeur: "RFH ") (valeur initiale).

MQRFH_STRUC_ID_ARRAY (valeur: "R", "F", "H", " ") est également défini.

Version (MQLONG)
Numéro de version de la structure.

Doit être MQRFH_VERSION_2 (valeur: 2) (valeur initiale).

StrucLength (MQLONG)
Longueur totale de MQRFH2, y compris les zones de données NameValue.

La valeur définie dans StrucLength doit être un multiple de 4 (pour cela, les données des zones de données NameValuepeuvent être remplies avec des espaces).

Codage (MQLONG)
Codage des données.

Codage des données numériques de la partie du message suivant MQRFH2 (en-tête suivant ou données de message suivant cet en-tête).

CodedCharSetId (MQLONG)
Identificateur de jeu de caractères codés.

Représentation des données de type caractère dans la partie du message qui suit l'en-tête MQRFH2 (en-tête suivant ou données de message qui suivent cet en-tête).

Format (MQCHAR8)
Nom de format.

Nom de format de la partie du message qui suit MQRFH2.

Indicateurs (MQLONG)
Indicateurs.

MQRFH_NO_FLAGS = 0. Aucun indicateur défini.

CCSID NameValue(MQLONG)
ID de jeu de caractères codés (CCSID) pour les chaînes de caractères de données NameValuecontenues dans cet en-tête. Les données NameValuepeuvent être codées dans un jeu de caractères différent des autres chaînes de caractères contenues dans l'en-tête (StrucID et Format).

Si le CCSID NameValueest un CCSID Unicode à 2 octets (1200, 13488 ou 17584), l'ordre des octets d'Unicode est le même que celui des octets des zones numériques dans MQRFH2. (Par exemple, Version, StrucLengthet NameValueCCSID lui-même.)

Tableau 16. Valeurs possibles pour la zone NameValueCCSID

CCSID	**Explication**
1 200	UTF-16, version Unicode la plus récente prise en charge
13488	UTF-16, sous-ensemble de la version Unicode 2.0
17584	UTF-16, Unicode version 3.0 sous-ensemble (inclut le symbole Euro)
1208	UTF-8, version Unicode la plus récente prise en charge

Partie variable

La partie variable suit la partie fixe. La partie variable contient un nombre variable de dossiers MQRFH2 . Chaque dossier contient un nombre variable d'éléments ou de propriétés. Propriétés associées au groupe de dossiers. Les en-têtes MQRFH2 créés par JMS peuvent contenir l'un des dossiers suivants:

Le dossier `mcd`

`mcd` contient des ropriétés qui décrivent le format du message. Par exemple, la propriété du domaine de service de message `Msd` identifie un message JMS comme étant `JMSTextMessage`, `JMSBytesMessage`, `JMSStreamMessage`, `JMSMapMessage`, `JMSObjectMessage` ou NULL.

Le dossier `mcd` est toujours présent dans un message JMS contenant un `MQRFH2`.

Il est toujours présent dans un message contenant un `MQRFH2` envoyé par IBM Integration Bus. Il décrit le domaine, le format, le type et l'ensemble de messages d'un message.

Tableau 17. Nom de la propriété `mcd`, synonyme, type de données et dossier

Synonyme de propriété	Nom de la propriété	Type de données	Dossier
	`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>`

N'ajoutez pas vos propres propriétés dans le dossier `mcd`.

Le dossier `jms`

`jms` contient des zones d'en-tête JMS et des propriétés JMSX qui ne peuvent pas être entièrement exprimées dans `MQMD`. Le dossier `jms` est toujours présent dans un `MQRFH2JMS`.

Le dossier `usr`

`usr` contient les propriétés JMS définies par l'application associées au message. Le dossier `usr` n'est présent que si une application a défini une propriété définie par l'application.

Le dossier `mqext`

`mqext` contient les types de propriété suivants :

- Les propriétés qui ne sont utilisées que par WebSphere Application Server.
- Les propriétés liées à la distribution retardée des messages.

Le dossier est présent si l'application a défini au moins l'une des propriétés définies par IBM ou utilisé un retard de distribution.

Tableau 18. Nom de la propriété mqext, synonyme, type de données et dossier			
Synonyme de propriété	**Nom de la propriété**	**Type de données**	**Dossier**
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>
JMSDeliveryTime	mqext.Dlt	i8	<mqext><Dlt>*DeliveryTime*</Dlt></mqext>
JMSDeliveryDelay	mqext.Dly	i8	<mqext><Dly>*DeliveryTime*</Dly></mqext>

N'ajoutez pas vos propres propriétés dans le dossier mqext.

Le dossier mqps

mqps contient des propriétés qui sont utilisées uniquement par la publication/l'abonnement IBM MQ. Le dossier est présent uniquement si l'application a défini au moins l'une des propriétés de publication/abonnement intégré.

Tableau 19. Nom de la propriété mqps, synonyme, type de données et dossier			
Synonyme de propriété	**Nom de la propriété**	**Type de données**	**Dossier**
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>

N'ajoutez pas vos propres propriétés dans le dossier mqps.

Le Tableau 20, à la page 157 affiche la liste complète des noms de propriété.

Tableau 20. Dossiers et propriétés MQRFH2 utilisés par JMS

JMS nom de zone	Java type	Nom de dossier MQRFH2	Nom de la propriété	Type / valeurs
JMSDestination	Destination	jms	Outils de maintenance en mode dédié	chaîne
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	PRI	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	CID	chaîne
JMSReplyTo	Destination	jms	RTO	chaîne
JMSTimestamp	long	jms	tms	i8
JMSType	String	distribution dans plusieurs pays	Type, Ensemble, Fmt	chaîne
JMSXGroupID	String	jms	gid	chaîne
JMSXGroupSeq	int	jms	Séq	i4
xxx (défini par l'utilisateur)	Tous	USR	xxx	toutes
		distribution dans plusieurs pays	MSD	jms_none texte_JMS octets_JMS jms_map jms_stream objet_JMS

NameValueLongueur (MQLONG)
Longueur en octets de la chaîne de données NameValuequi suit immédiatement cette zone de longueur (elle n'inclut pas sa propre longueur).

Données NameValue(MQCHARn)
Chaîne de caractères unique, dont la longueur en octets est donnée par la zone de longueur NameValueprécédente. Il contient un dossier contenant une séquence de propriétés. Chaque propriété est un triplet nom / type/valeur, contenu dans un élément XML dont le nom est le nom de dossier, comme suit:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

La balise `</foldername>` de fermeture peut être suivie d'espaces en tant que caractères de remplissage. Chaque triplet est codé à l'aide d'une syntaxe de type XML:

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

L'élément `dt='datatype'` est facultatif et est omis pour de nombreuses propriétés, car le type de données est prédéfini. S'il est inclus, un ou plusieurs espaces doivent être inclus avant la balise `dt=` .

name
est le nom de la propriété ; voir Tableau 20, à la page 157.

datatype
doit correspondre, après le pliage, à l'un des types de données répertoriés dans le Tableau 21, à la page 158.

value
est une représentation de chaîne de la valeur à transmettre, à l'aide des définitions dans Tableau 21, à la page 158.

Une valeur nulle est codée à l'aide de la syntaxe suivante:

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

N'utilisez pas `xsi:nil='false'`.

Tableau 21. Types de données de propriété

Type de données	**Définition**
chaîne	Toute séquence de caractères à l'exception de < et &
boolean	Le caractère `0` ou `1` (`0` = false, `1` = true)
bin.hex	Chiffres hexadécimaux représentant les octets
i1	Nombre, exprimé à l'aide de chiffres `0..9`, avec un signe facultatif (aucune fraction ou exposant). Doit être compris entre -128 et 127 inclus
i2	Nombre, exprimé à l'aide de chiffres `0..9`, avec un signe facultatif (aucune fraction ou exposant). Doit être compris entre -32768 et 32767 inclus
i4	Nombre, exprimé à l'aide de chiffres `0..9`, avec un signe facultatif (aucune fraction ou exposant). Doit être compris entre -2147483648 et 2147483647 inclus
i8	Nombre, exprimé à l'aide de chiffres `0..9`, avec un signe facultatif (aucune fraction ou exposant). Doit être compris entre -9223372036854775808 et 9223372036854775807 inclus
int	Nombre, exprimé à l'aide de chiffres `0..9`, avec un signe facultatif (aucune fraction ou exposant). Doit être compris dans la même plage que i8. Peut être utilisé à la place de l'un des types i * si l'expéditeur ne souhaite pas associer une précision particulière à la propriété
r4	Nombre à virgule flottante, magnitude < = 3.40282347E+38, > = 1.175E-37 exprimée à l'aide de chiffres `0..9`, signe facultatif, chiffres fractionnaires facultatifs, exposant facultatif
r8	Nombre à virgule flottante, magnitude < = 1.7976931348623E+308, > = 2.225E-307 exprimée à l'aide de chiffres `0..9`, signe facultatif, chiffres fractionnaires facultatifs, exposant facultatif

Une valeur de chaîne peut contenir des espaces. Vous devez utiliser les séquences d'échappement suivantes dans une valeur de chaîne:

- `&` pour le caractère &
- `<` pour le caractère <

Vous pouvez utiliser les séquences d'échappement suivantes, mais elles ne sont pas obligatoires:

- `>` pour le caractère >
- `&apos;` pour le caractère '
- `&quot;` pour le caractère "

Zones et propriétés JMS avec les zones MQMD correspondantes
Ces tableaux présentent les zones MQMD équivalentes aux zones d'en-tête JMS , aux propriétés JMS et aux propriétés spécifiques du fournisseur JMS .

Tableau 22, à la page 159 répertorie les zones d'en-tête JMS et Tableau 23, à la page 159 répertorie les propriétés JMS qui sont mappées directement aux zones MQMD. Le Tableau 24, à la page 159 répertorie les propriétés spécifiques au fournisseur et les zones MQMD auxquelles elles sont mappées.

Tableau 22. Mappage des zones d'en-tête JMS aux zones MQMD

JMS Zone d'en-tête	**Java type**	**Zone MQMD**	**Type sté**
JMSDeliveryMode	int	Persistance	MQLONG
JMSExpiration	long	Expiration	MQLONG
JMSPriority	int	Priorité	MQLONG
JMSMessageID	String	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	String	CorrelId	MQBYTE24

Tableau 23. Mappage des propriétés JMS aux zones MQMD

JMS Propriété	**Java type**	**Zone MQMD**	**Type sté**
JMSXUserID	String	UserIdentifier	MQCHAR12
JMSXAppID	String	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	String	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Tableau 24. Mappage des propriétés spécifiques au fournisseur JMS vers les zones MQMD

Propriété spécifique au fournisseur JMS	**Java type**	**Zone MQMD**	**Type sté**
JMS_IBM_Report_Exception	int	Rapport	MQLONG
JMS_IBM_Report_Expiration	int	Rapport	MQLONG
JMS_IBM_Report_COA	int	Rapport	MQLONG
JMS_IBM_Report_COD	int	Rapport	MQLONG
JMS_IBM_Report_PAN	int	Rapport	MQLONG
JMS_IBM_Report_NAN	int	Rapport	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Rapport	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Rapport	MQLONG
JMS_IBM_Report_Discard_Msg	int	Rapport	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Commentaires	MQLONG

Tableau 24. Mappage des propriétés spécifiques au fournisseur JMS vers les zones MQMD (suite)			
Propriété spécifique au fournisseur JMS	**Java type**	**Zone MQMD**	**Type sté**
JMS_IBM_Format	String	Format «1», à la page 160	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codage	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId «2», à la page 160	MQLONG
JMS_IBM_PutDate	String	PutDate	MQCHAR8
JMS_IBM_PutTime	String	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

Remarque :

1. JMS_IBM_Format représente le format du corps du message. Cela peut être défini par l'application définissant la propriété JMS_IBM_Format du message (notez qu'il existe une limite de 8 caractères) ou peut être défini par défaut sur le format IBM MQ du corps du message correspondant au type de message JMS . JMS_IBM_Format est mappé à la zone Format MQMD uniquement si le message ne contient aucune section RFH ou RFH2 . Dans un message standard, il est mappé au champ Format de RFH2 précédant immédiatement le corps du message.
2. La valeur de la propriété JMS_IBM_Character_Set est une valeur de chaîne qui contient l'équivalent de jeu de caractères Java pour la valeur numérique CodedCharacterSetId . La zone MQMD CodedCharacterSetId est une valeur numérique qui contient l'équivalent de la chaîne de jeu de caractères Java spécifiée par la propriété JMS_IBM_Character_Set.

Mappage de zones JMS vers des zones IBM MQ (messages sortants)
Ces tableaux montrent comment les zones d'en-tête et de propriété JMS sont mappées dans les zones MQMD et MQRFH2 au moment de l'envoi () ou de la publication ().

Tableau 25, à la page 160 montre comment les zones d'en-tête JMS sont mappées dans des zones MQMD/RFH2 au moment de l'envoi () ou de la publication (). La Tableau 26, à la page 161 montre comment les propriétés JMS sont mappées dans les zones MQMD/RFH2 au moment de l'envoi () ou de la publication (). Tableau 27, à la page 162 montre comment les propriétés spécifiques au fournisseur JMS sont mappées aux zones MQMD au moment de l'envoi () ou de la publication (),

Pour les zones marquées comme Set by Message Object, la valeur transmise est la valeur contenue dans le message JMS immédiatement avant l'opération send () ou publish (). La valeur du message JMS reste inchangée par l'opération.

Pour les zones marquées comme définies par la méthode d'envoi, une valeur est affectée lors de l'envoi () ou de la publication () (toute valeur contenue dans le message JMS est ignorée). La valeur du message JMS est mise à jour pour afficher la valeur utilisée.

Les champs marqués comme Recevoir uniquement ne sont pas transmis et restent inchangés dans le message par send () ou publish ().

Tableau 25. Mappage de zones de message sortant			
Nom de zone d'en-tête JMS	**Zone MQMD utilisée pour la transmission**	**En-tête**	**Définie par**
JMSDestination		MQRFH2	Méthode d'envoi
JMSDeliveryMode	Persistance	MQRFH2	Méthode d'envoi

Tableau 25. Mappage de zones de message sortant (suite)			
Nom de zone d'en-tête JMS	**Zone MQMD utilisée pour la transmission**	**En-tête**	**Définie par**
JMSExpiration	Expiration	MQRFH2	Méthode d'envoi
JMSPriority	Priorité	MQRFH2	Méthode d'envoi
JMSMessageID	MsgID		Méthode d'envoi
JMSTimestamp	PutDate/PutTime		Méthode d'envoi
JMSCorrelationID	CorrelId	MQRFH2	objet Message
JMSReplyTo	ReplyToQ/ ReplyToGestionnaire de files d'attente	MQRFH2	objet Message
JMSType		MQRFH2	objet Message
JMSRedelivered			Réception uniquement

Remarque :

1. La zone MQMD CodedCharacterSetId est une valeur numérique qui contient l'équivalent de la chaîne de jeu de caractères Java spécifiée par la propriété JMS_IBM_Character_Set.

Tableau 26. Mappage de la propriété JMS du message sortant			
Nom de la propriété JMS	**Zone MQMD utilisée pour la transmission**	**En-tête**	**Définie par**
JMSXUserID	UserIdentifier		Méthode d'envoi
JMSXAppID	PutApplName		Méthode d'envoi
JMSXDeliveryCount			Réception uniquement
JMSXGroupID	GroupId	MQRFH2	objet Message
JMSXGroupSeq	MsgSeqNumber	MQRFH2	objet Message

Remarque :

Ces propriétés sont définies en lecture seule par la spécification JMS et sont définies (dans certains cas, en option) par le fournisseur JMS.

Dans IBM MQ classes for JMS , deux de ces propriétés peuvent être remplacées par l'application. Pour ce faire, assurez-vous que la destination a été configurée correctement en définissant les propriétés suivantes:

1. Définissez la propriété `WMQConstants.WMQ_MQMD_MESSAGE_CONTEXT` sur `WMQConstants.WMQ_MDCTX_SET_ALL_CONTEXT`.
2. Définissez la propriété `WMQConstants.WMQ_MQMD_WRITE_ENABLED` sur `true`.

Les propriétés suivantes peuvent être remplacées par l'application:

JMSXAppID
Cette propriété peut être remplacée en définissant la propriété `WMQConstants.JMS_IBM_MQMD_PUTAPPLNAME` sur le message-la valeur doit être une chaîne Java .

JMSXGroupID
Cette propriété peut être remplacée en définissant la propriété `WMQConstants.JMS_IBM_MQMD_GROUPID` sur le message-la valeur doit être un tableau d'octets.

Tableau 27. Mappage de propriété spécifique au fournisseur JMS de message sortant			
Nom de propriété spécifique au fournisseur JMS	**Zone MQMD utilisée pour la transmission**	**En-tête**	**Définie par**
JMS_IBM_Report_Exception	Rapport		objet Message
JMS_IBM_Report_Expiration	Rapport		objet Message
JMS_IBM_Report_COA/COD	Rapport		objet Message
JMS_IBM_Rapport_NAN/PAN	Rapport		objet Message
JMS_IBM_Report_Pass_Msg_ID	Rapport		objet Message
JMS_IBM_Report_Pass_Correl_ID	Rapport		objet Message
JMS_IBM_Report_Discard_Msg	Rapport		objet Message
JMS_IBM_MsgType	MsgType		objet Message
JMS_IBM_Feedback	Commentaires		objet Message
JMS_IBM_Format	Format		objet Message
JMS_IBM_PutApplType	PutApplType		Méthode d'envoi
JMS_IBM_Encoding	Codage		objet Message
JMS_IBM_Character_Set	CodedCharacterSetId		objet Message
JMS_IBM_PutDate	PutDate		Méthode d'envoi
JMS_IBM_PutTime	PutTime		Méthode d'envoi
JMS_IBM_Last_Msg_In_Group	MsgFlags		objet Message

Mappage des zones d'en-tête JMS à send () ou publish ()
Ces remarques concernent le mappage des zones JMS à send () ou publish ().

JMSDestination vers MQRFH2
Elle est stockée sous la forme d'une chaîne qui sérialise les caractéristiques principales de l'objet de destination de sorte qu'un objet JMS de réception puisse reconstituer un objet de destination équivalent. La zone MQRFH2 est codée en tant qu'URI (pour plus de détails sur la notation de l'URI, voir «uniform resource identifier (URI)», à la page 229).

JMSReplyTo vers MQMD.ReplyToQ, ReplyToQMgr, MQRFH2
Le nom de la file d'attente est copié dans MQMD.ReplyToQ et le nom du gestionnaire de files d'attente sont copiés dans les zones du gestionnaire de files d'attente ReplyTo. Les informations d'extension de destination (autres détails utiles conservés dans l'objet de destination) sont copiées dans la zone MQRFH2 . La zone MQRFH2 est codée en tant qu'URI (voir «uniform resource identifier (URI)», à la page 229 pour plus de détails sur la notation de l'URI).

JMSDeliveryMode vers MQMD.Persistence
La valeur JMSDeliveryMode est définie par la méthode send () ou publish () ou MessageProducer, sauf si l'objet de destination la remplace. La valeur JMSDeliveryMode est mappée à MQMD.Persistence de persistance comme suit:

- La valeur JMS PERSISTENT est équivalente à MQPER_PERSISTENT
- La valeur JMS NON_PERSISTENT est équivalente à MQPER_NOT_PERSISTENT

Si la propriété de persistance MQQueue n'est pas définie sur WMQConstants.WMQ_PER_QDEF, la valeur du mode de distribution est également codée dans MQRFH2.

JMSExpiration vers / depuis MQMD.Expiry, MQRFH2
JMSExpiration stocke le délai d'expiration (somme de l'heure en cours et de la durée de vie), tandis que MQMD stocke la durée de vie. De plus, JMSExpiration est en millisecondes, mais MQMD.Expiry est en dixièmes de seconde.

- Si la méthode send () définit une durée de vie illimitée, MQMD.Expiry d'expiration est défini sur MQEI_UNLIMITED et aucun élément JMSExpiration n'est codé dans MQRFH2.
- Si la méthode send () définit une durée de vie inférieure à 214748364.7 secondes (environ 7 ans), la durée de vie est stockée dans MQMD.Expiryet le délai d'expiration (en millisecondes) sont codés en tant que valeur i8 dans MQRFH2.
- Si la méthode send () définit une durée de vie supérieure à 214748364.7 secondes, MQMD.Expiry est définie sur MQEI_UNLIMITED. Le délai d'expiration réel en millisecondes est codé en tant que valeur i8 dans MQRFH2.

JMSPriority pour MQMD.Priority
Mapper directement la valeur JMSPriority (0-9) à la valeur de priorité MQMD (0-9). Si JMSPriority est défini sur une valeur autre que la valeur par défaut, le niveau de priorité est également codé dans MQRFH2.

JMSMessageID de MQMD.MessageID
Tous les messages envoyés depuis JMS ont des identificateurs de message uniques affectés par IBM MQ. La valeur affectée est renvoyée dans MQMD.MessageId après l'appel MQPUT et est retransmise à l'application dans la zone JMSMessageID . IBM MQ messageId est une valeur binaire de 24 octets, tandis que JMSMessageID est une chaîne. L'ID JMSMessageID est composé de la valeur binaire messageId convertie en une séquence de 48 caractères hexadécimaux, préfixée avec l'ID caractères:. JMS fournit une suggestion qui peut être définie pour désactiver la production des identificateurs de message. Cette suggestion est ignorée et un identificateur unique est affecté dans tous les cas. Toute valeur définie dans la zone JMSMessageID avant une fonction send () est remplacée.

Si vous avez besoin de la possibilité de spécifier le MQMD MQMD.MessageID, vous pouvez effectuer cette opération avec l'une des extensions IBM MQ JMS décrites dans «Lecture et écriture du descripteur de message à partir d'une application IBM MQ classes for JMS», à la page 251.

JMSTimestamp à MQRFH2
Lors d'un envoi, la zone JMSTimestamp est définie en fonction de l'horloge de la machine virtuelle Java. Cette valeur est définie dans MQRFH2. Toute valeur définie dans la zone JMSTimestamp avant qu'une fonction send () ne soit écrasée. Voir aussi les propriétés JMS_IBM_PutDate et JMS_IBM_PutTime .

JMSType vers MQRFH2
Cette chaîne est définie dans la zone MQRFH2 mcd.Type . S'il est au format URI, il peut également affecter les zones mcd.Set et mcd.Fmt .

JMSCorrelationID à MQMD.CorrelId, MQRFH2
L'ID JMSCorrelationID peut contenir l'un des éléments suivants:

Un ID de message spécifique au fournisseur
Il s'agit d'un identificateur de message provenant d'un message précédemment envoyé ou reçu. Il doit donc s'agir d'une chaîne de 48 chiffres hexadécimaux minuscules préfixés avec le préfixe `ID:` . Le préfixe est supprimé, les caractères restants sont convertis en caractères binaires, puis ils sont définis dans MQMD.CorrelId .

Valeur de l'octet natif du fournisseur [ ]
La valeur est copiée dans MQMD.CorrelId zone-complétée avec des valeurs nulles ou tronquée à 24 octets si nécessaire. Aucune valeur CorrelId n'est codée dans MQRFH2.

Une chaîne spécifique à l'application
La valeur est copiée dans MQRFH2. Les 24 premiers octets de la chaîne, au format UTF8 , sont écrits dans MQMD.CorrelID.

Mappage des zones de propriété JMS
Ces remarques font référence au mappage des zones de propriété JMS dans les messages IBM MQ .

JMSXUserID de MQMD UserIdentifier
JMSXUserID est défini en cas de retour d'un appel d'envoi.

JMSXAppID de MQMD PutApplNom
JSMXAppID est défini en cas de retour de l'appel d'envoi.

JMSXGroupID vers MQRFH2 (point à point)
Pour les messages point à point, JMSXGroupID est copié dans la zone MQMD GroupID . Si JMSXGroupID commence par le préfixe ID:, il est converti en binaire. Sinon, il est codé en tant que chaîne UTF8 . La valeur est complétée ou tronquée si nécessaire à une longueur de 24 octets. L'indicateur MQMF_MSG_IN_GROUP est défini.

JMSXGroupID sur MQRFH2 (publication / abonnement)
Pour les messages de publication / abonnement, JMSXGroupID est copié dans MQRFH2 sous forme de chaîne.

JMSXGroupSeq MQMD MsgSeqNuméro (point à point)
Pour les messages point à point, JMSXGroupSeq est copié dans la zone MQMD MsgSeqNumber. L'indicateur MQMF_MSG_IN_GROUP est défini.

JMSXGroupSeq MQMD MsgSeqNuméro (publication / abonnement)
Pour les messages de publication / abonnement, JMSXGroupSeq est copié dans MQRFH2 en tant que i4.

Mappage des zones spécifiques au fournisseur JMS
Les remarques suivantes font référence au mappage des zones spécifiques au fournisseur JMS dans les messages IBM MQ .

Rapport JMS_IBM_Report_*XXX* vers MQMD
Une application JMS peut définir les options de rapport MQMD à l'aide des propriétés JMS_IBM_Report_*XXX* suivantes. Le MQMD unique est mappé à plusieurs propriétés JMS_IBM_Report_*XXX* .

Les constantes JMS_IBM_Report_*XXX* se trouvent dans `com.ibm.msg.client.jakarta.wmq.WMQConstants` ou `com.ibm.msg.client.wmq.WMQConstants`.

JMS_IBM_Report_Exception

MQRO_EXCEPTION ou
MQRO_EXCEPTION_WITH_DATA ou
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration

MQRO_EXPIRATION ou
MQRO_EXPIRATION_WITH_DATA ou
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA

MQRO_COA ou
MQRO_COA_WITH_DATA ou
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD

MQRO_COD ou
MQRO_COD_WITH_DATA ou
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

Les valeurs MQRO se trouvent dans `com.ibm.mq.constants.CMQC`.

JMS_IBM_MsgType to MQMD MsgType
La valeur est mappée directement sur MQMD MsgType. Si l'application n'a pas défini une valeur explicite de JMS_IBM_MsgType, une valeur par défaut est utilisée. Cette valeur par défaut est déterminée comme suit:

- Si JMSReplyTo est défini sur une destination de file d'attente IBM MQ , MSGType est défini sur la valeur MQMT_REQUEST
- Si JMSReplyTo n'est pas défini ou est défini sur une destination autre qu'une destination de file d'attente IBM MQ , MsgType est défini sur la valeur MQMT_DATAGRAM

Commentaires en retour JMS_IBM_Feedback vers MQMD
La valeur est directement mappée à MQMD Feedback.

JMS_IBM_Format vers le format MQMD
La valeur est directement mappée au format MQMD.

Codage JMS_IBM_Encoding en MQMD
Si cette propriété est définie, elle remplace le codage numérique de la file d'attente de destination ou de la rubrique.

JMS_IBM_Character_Set vers MQMD CodedCharacterSetId
Si cette propriété est définie, elle remplace la propriété de jeu de caractères codés de la file d'attente de destination ou de la rubrique.

JMS_IBM_PutDate de MQMD PutDate
La valeur de cette propriété est définie, lors de l'envoi, directement à partir de la zone PutDate dans le MQMD. Toute valeur définie dans la propriété JMS_IBM_PutDate avant un envoi est remplacée. Cette zone est une chaîne de huit caractères, au format de date IBM MQ AAAAMMJJ. Cette propriété peut être utilisée avec la propriété JMS_IBM_PutTime pour déterminer l'heure à laquelle le message a été inséré en fonction du gestionnaire de files d'attente.

JMS_IBM_PutTime de MQMD PutTime
La valeur de cette propriété est définie, lors de l'envoi, directement à partir de la zone PutTime dans le MQMD. Toute valeur définie dans la propriété JMS_IBM_PutTime avant un envoi est remplacée. Cette zone est une chaîne de huit caractères, au format d'heure IBM MQ HHMMSSTH. Cette propriété peut être utilisée avec la propriété JMS_IBM_PutDate pour déterminer l'heure à laquelle le message a été inséré en fonction du gestionnaire de files d'attente.

JMS_IBM_Last_Msg_In_Group vers MQMD MsgFlags
Pour la messagerie point-à-point, cette valeur booléenne est mappée à l'indicateur MQMF_LAST_MSG_IN_GROUP dans la zone MQMD MsgFlags . Il est normalement utilisé avec les propriétés JMSXGroupID et JMSXGroupSeq pour indiquer à une application IBM MQ existante que ce message est le dernier d'un groupe. Cette propriété est ignorée pour la messagerie de publication / abonnement.

Mappage de zones IBM MQ à des zones JMS (messages entrants)
Ces tableaux montrent comment les zones d'en-tête et de propriété JMS sont mappées dans les zones MQMD et MQRFH2 à l'heure get () ou receive ().

La Tableau 28, à la page 166 montre comment les zones d'en-tête JMS sont mappées aux zones MQMD/MQRFH2 au moment de l'obtention () ou de la réception (). La Tableau 29, à la page 166 montre comment les zones de propriété JMS sont mappées sur des zones MQMD/MQRFH2 au moment de l'obtention () ou de la réception (). Le Tableau 30, à la page 167 montre comment les propriétés spécifiques au fournisseur JMS sont mappées.

Tableau 28. Mappage de zones d'en-tête JMS de message entrant		
Nom de zone d'en-tête JMS	**Zone MQMD extraite de**	**Zone MQRFH2 extraite de**
JMSDestination		jms.Dst ou mqps.Top «1», à la page 166
JMSDeliveryMode	Persistance«2», à la page 166	jms.Dlv «2», à la page 166
JMSExpiration		jms.Exp
JMSPriority	Priorité	
JMSMessageID	MsgID	
JMSTimestamp	PutDate «2», à la page 166 PutTime «2», à la page 166	jms.Tms «2», à la page 166
JMSCorrelationID	CorrelId «2», à la page 166	jms.Cid «2», à la page 166
JMSReplyTo	ReplyToQ «2», à la page 166 ReplyToGestionnaire de files d'attente «2», à la page 166	jms.Rto «2», à la page 166
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Remarque :

1. Si jms.Dst et mqps.Top sont définis, la valeur de jms.Dst est utilisée.
2. Pour les propriétés qui peuvent avoir des valeurs extraites de MQRFH2 ou de MQMD, si les deux sont disponibles, le paramètre dans MQRFH2 est utilisé.
3. La valeur de la propriété JMS_IBM_Character_Set est une valeur de chaîne qui contient l'équivalent de jeu de caractères Java pour la valeur numérique CodedCharacterSetId .

Tableau 29. Mappage des propriétés de message entrant		
Nom de la propriété JMS	**Zone MQMD extraite de**	**Zone MQRFH2 extraite de**
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId «1», à la page 166	jms.Gid «1», à la page 166
JMSXGroupSeq	MsgSeqNuméro «1», à la page 166	jms.Seq «1», à la page 166

Remarque :

1. Pour les propriétés qui peuvent avoir des valeurs extraites de MQRFH2 ou de MQMD, si les deux sont disponibles, le paramètre dans MQRFH2 est utilisé. Les propriétés sont définies à partir des valeurs MQMD uniquement si les indicateurs de message MQMF_MSG_IN_GROUP ou MQMF_LAST_MSG_IN_GROUP sont définis.

Tableau 30. Mappage de propriété JMS spécifique au fournisseur de messages entrants		
Nom de la propriété JMS	**Zone MQMD extraite de**	**Zone MQRFH2 extraite de**
JMS_IBM_Report_Exception	Rapport	
JMS_IBM_Report_Expiration	Rapport	
JMS_IBM_Report_COA	Rapport	
JMS_IBM_Report_COD	Rapport	
JMS_IBM_Report_PAN	Rapport	
JMS_IBM_Report_NAN	Rapport	
JMS_IBM_Report_Pass_Msg_ID	Rapport	
JMS_IBM_Report_Pass_Correl_ID	Rapport	
JMS_IBM_Report_Discard_Msg	Rapport	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Commentaires	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding «1», à la page 167	Codage	
JMS_IBM_Character_Set «1», à la page 167	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Défini uniquement si le message entrant est un message Bytes.

Echange de messages entre une application JMS et une application IBM MQ traditionnelle
Cette rubrique décrit ce qui se passe lorsqu'une application JMS échange des messages avec une application IBM MQ traditionnelle qui ne peut pas traiter l'en-tête MQRFH2 .

La Figure 11, à la page 168 montre le mappage.

L'administrateur indique que l'application JMS communique avec une application IBM MQ traditionnelle en définissant la propriété TARGCLIENT de la destination sur *MQ*. Cela indique qu'aucun en-tête MQRFH2 ne doit être généré. Si tel n'est pas le cas, l'application de réception doit pouvoir traiter l'en-tête MQRFH2 .

Le mappage de JMS vers MQMD destiné à une application IBM MQ traditionnelle est identique au mappage de JMS vers MQMD destiné à une application JMS . Si IBM MQ classes for JMS reçoit un message IBM MQ avec la zone MQMD *Format* définie sur autre chose que MQFMT_RFH2, des données sont reçues d'une application nonJMS . Si le format est MQFMT_STRING, le message est reçu sous la forme d'un message texte JMS . Sinon, il est reçu sous la forme d'un message d'octets JMS . Etant donné qu'il n'existe pas de MQRFH2, seules les propriétés JMS transmises dans le MQMD peuvent être restaurées.

Si IBM MQ classes for JMS reçoit un message qui ne comporte pas d'en-tête MQRFH2 , la propriété TARGCLIENT de l'objet Queue ou Topic dérivé de la zone d'en-tête JMSReplyTo du message est définie sur MQ par défaut. Cela signifie qu'un message de réponse envoyé à la file d'attente ou à la rubrique ne comporte pas non plus d'en-tête MQRFH2 . Vous pouvez désactiver ce comportement d'inclusion d'un

en-tête MQRFH2 dans un message de réponse uniquement si le message d'origine comporte un en-tête MQRFH2 , en définissant la propriété TARGCLIENTMATCHING de la fabrique de connexions sur NO.

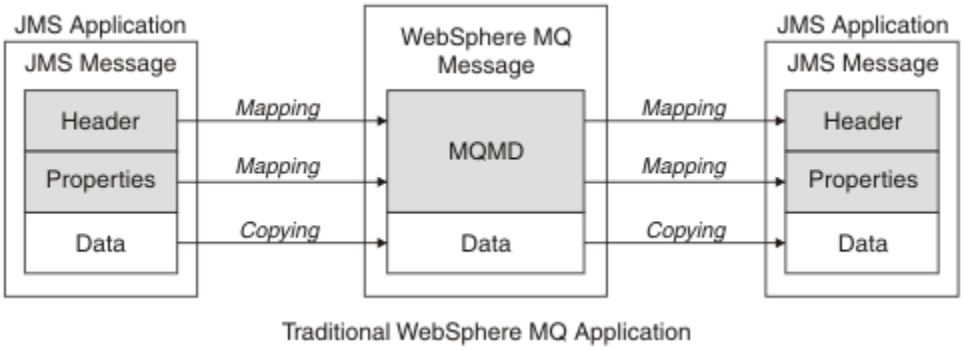


Figure 11. Comment les messages JMS sont transformés en messages IBM MQ sans en-tête MQRFH2

Corps du message JMS
Cette rubrique contient des informations sur le codage du corps du message lui-même. Le codage dépend du type de message JMS .

ObjectMessage
Un ObjectMessage est un objet sérialisé par Java Runtime de manière normale.

TextMessage
Un TextMessage est une chaîne codée. Pour un message sortant, la chaîne est codée dans le jeu de caractères indiqué par l'objet de destination. Par défaut, il s'agit du codage UTF8 (le codage UTF8 commence par le premier caractère du message ; il n'y a pas de zone de longueur au début). Il est toutefois possible de spécifier tout autre jeu de caractères pris en charge par IBM MQ classes for JMS. Ces jeux de caractères sont utilisés principalement lorsque vous envoyez un message à une application nonJMS .

Si le jeu de caractères est un jeu de caractères codés sur deux octets (y compris UTF16), la spécification de codage d'entier de l'objet cible détermine l'ordre des octets.

Un message entrant est interprété à l'aide du jeu de caractères et du codage spécifiés dans le message lui-même. Ces spécifications se trouvent dans le dernier en-tête IBM MQ (ou MQMD s'il n'y a pas d'en-tête). Pour les messages JMS , le dernier en-tête est généralement MQRFH2.

BytesMessage
Un BytesMessage est, par défaut, une séquence d'octets définie par la spécification JMS 1.0.2 et la documentation Java associée.

Pour un message sortant qui a été assemblé par l'application elle-même, la propriété de codage de l'objet de destination peut être utilisée pour remplacer les codages des zones d'entier et de virgule flottante contenues dans le message. Par exemple, vous pouvez demander que les valeurs en virgule flottante soient stockées au format S/390 plutôt qu'au format IEEE).

Un message entrant est interprété à l'aide du codage numérique spécifié dans le message lui-même. Cette spécification se trouve dans le dernier en-tête IBM MQ (ou MQMD s'il n'existe pas d'en-tête). Pour les messages JMS , le dernier en-tête est généralement MQRFH2.

Si un message BytesMessage est reçu et renvoyé sans modification, son corps est transmis octet par octet, tel qu'il a été reçu. La propriété de codage de l'objet de destination n'a aucun effet sur le corps. La seule entité de type chaîne pouvant être envoyée explicitement dans un BytesMessage est une chaîne UTF8 . Il est codé au format Java UTF8 et commence par une zone de longueur de 2 octets. La propriété de jeu de caractères de l'objet de destination n'a aucun effet sur le codage d'un BytesMessagesortant. La valeur de jeu de caractères dans un message IBM MQ entrant n'a aucun effet sur l'interprétation de ce message en tant que message JMS BytesMessage.

Il est peu probable que les applications nonJava reconnaissent le codage Java UTF8 . Par conséquent, pour qu'une application JMS envoie un BytesMessage contenant des données texte,

l'application elle-même doit convertir ses chaînes en tableaux d'octets et écrire ces tableaux d'octets dans BytesMessage.

MapMessage
Un MapMessage est une chaîne contenant des triplets nom / type/valeur XML codés comme suit:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

où `datatype` correspond à l'un des types de données répertoriés dans le Tableau 21, à la page 158. Le type de données par défaut est `string`, de sorte que l'attribut `dt="string"` est omis pour les éléments de chaîne.

Le jeu de caractères utilisé pour coder ou interpréter la chaîne XML qui forme le corps d'un message de mappe est déterminé en fonction des règles qui s'appliquent à un message texte.

Les versions de IBM MQ classes for JMS antérieures à 5.3 ont codé le corps d'un message de mappe au format suivant:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

IBM MQ classes for JMS 5.3 et versions ultérieures peuvent interpréter l'un ou l'autre de ces formats, mais les versions de IBM MQ classes for JMS antérieures à 5.3 ne peuvent pas interpréter le format en cours.

Si une application doit envoyer des messages de mappe à une autre application qui utilise une version de IBM MQ classes for JMS antérieure à 5.3, l'application émettrice doit appeler la méthode de fabrique de connexions `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` pour indiquer que les messages de mappe sont envoyés dans le format précédent. Par défaut, tous les messages de mappe sont envoyés au format en cours.

StreamMessage
Un StreamMessage est similaire à un message de mappe, mais sans noms d'élément:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

où `datatype` correspond à l'un des types de données répertoriés dans le Tableau 21, à la page 158. Le type de données par défaut est `string`, de sorte que l'attribut `dt="string"` est omis pour les éléments de chaîne.

Le jeu de caractères utilisé pour coder ou interpréter la chaîne XML qui constitue le corps StreamMessage est déterminé en suivant les règles qui s'appliquent à un TextMessage.

La zone MQRFH2.format est définie comme suit:

MQFMT_AUCUN
pour ObjectMessage, BytesMessageou des messages sans corps.

MQFMT_STRING
pour TextMessage, StreamMessageou MapMessage.

JMS Conversion de message
La conversion des données de message dans JMS est effectuée lors de l'envoi et de la réception de messages. IBM MQ effectue la plupart des conversions de données automatiquement. Il convertit les données texte et numériques lors du transfert d'un message entre les applications JMS . Le texte est converti lors de l'échange d'un `JMSTextMessage` entre une application JMS et une application IBM MQ .

Si vous prévoyez d'effectuer des échanges de messages plus complexes, les rubriques suivantes vous intéressent. Les échanges de messages complexes incluent:

- Transfert de messages non texte entre une application IBM MQ et une application JMS .
- Echange de données texte au format octet.
- Conversion du texte dans votre application.

JMS Données de message

La conversion de données est nécessaire pour échanger du texte et des données numériques entre des applications, même entre deux applications JMS . La représentation interne du texte et des nombres doit être codée pour pouvoir être transférée dans un message. Le codage force une décision sur la façon dont les nombres et le texte sont représentés. IBM MQ gère le codage du texte et des nombres dans les messages JMS , à l'exception de `JMSObjectMessage`. Voir «JMSObjectMessage», à la page 177. Il utilise trois attributs de message. Les trois attributs sont `CodedCharacterSetId`, `Encoding`et `Format`.

Ces trois attributs de message sont normalement stockés dans l'en-tête JMS , `MQRFH2`, zones d'un message JMS . Si le type de message est `MQ`et non JMS , les attributs sont stockés dans le descripteur de message, `MQMD`. Les attributs sont utilisés pour convertir les données de message JMS . Les données de message JMS sont transférées dans la partie données de message d'un message IBM MQ .

JMS Propriétés du message

Les propriétés de message JMS , telles que `JMS_IBM_CHARACTER_SET`, sont échangées dans la partie d'en-tête `MQRFH2` d'un message JMS , sauf si le message a été envoyé sans `MQRFH2`. Seuls `JMSTextMessage` et `JMSBytesMessage` peuvent être envoyés sans `MQRFH2`. Si une propriété JMS est stockée en tant que propriété de message IBM MQ dans le descripteur de message, `MQMD`, elle est convertie dans le cadre de la conversion `MQMD` . Si une propriété JMS est stockée dans `MQRFH2`, elle est stockée dans le jeu de caractères spécifié par `MQRFH2.NameValueCCSID`. Lorsqu'un message est envoyé ou reçu, les propriétés de message sont converties depuis et vers leur représentation interne dans la machine virtuelle Java. La conversion se fait vers et depuis le jeu de caractères du descripteur de message ou `MQRFH2.NameValueCCSID`. Les données numériques sont converties en texte.

JMS Conversion de message

Les rubriques suivantes contiennent des exemples et des tâches utiles si vous prévoyez d'échanger des messages plus complexes nécessitant une conversion.

Approches de conversion de message JMS
Un certain nombre d'approches de conversion de données sont ouvertes aux concepteurs d'applications JMS . Ces approches ne sont pas exclusives ; certaines applications sont susceptibles d'utiliser une combinaison de ces approches. Si votre application échange uniquement du texte ou des messages uniquement avec d'autres applications JMS , vous n'envisagez normalement pas la conversion de données. La conversion de données est effectuée automatiquement pour vous, par IBM MQ.

Vous pouvez poser un certain nombre de questions sur la façon d'aborder la conversion des messages:

Est-il nécessaire de penser à la conversion des messages?
Dans certains cas, tels que les transferts de messages JMS vers JMS et l'échange de messages texte avec des programmes IBM MQ , IBM MQ effectue automatiquement les conversions nécessaires pour vous. Vous pouvez être amené à contrôler la conversion des données pour des raisons de performances ou vous pouvez échanger des messages complexes ayant un format prédéfini. Dans de tels cas, vous devez comprendre la conversion de message et lire les rubriques suivantes.

Quels types de conversion y a-t-il?

Il existe quatre principaux types de conversion, qui sont expliqués dans les sections suivantes:

1. «Conversion de données client JMS», à la page 171
2. «Conversion des données d'application», à la page 172
3. «Conversion des données du gestionnaire de files d'attente», à la page 172
4. «Conversion de données de canal de transmission de messages», à la page 173

Où la conversion doit-elle être effectuée?

La section, «Choix d'une approche de la conversion de message: le récepteur est bon», à la page 173, décrit l'approche habituelle de "récepteur fait bonne figure". "Les bons résultats du récepteur" s'appliquent également à la conversion de données JMS .

Conversion de données client JMS

JMS client[1]La conversion de données est la conversion des primitives et des objets Java en octets dans un message JMS lorsqu'il est envoyé à une destination, puis la conversion à nouveau, lorsqu'il est reçu. La conversion de données client JMS utilise les méthodes des classes `JMSMessage` . Les méthodes sont répertoriées par type de classe `JMSMessage` dans Tableau 31, à la page 174.

La conversion vers et depuis la représentation JVM interne des nombres et du texte est effectuée pour les méthodes de lecture, d'extraction, de définition et d'écriture. La conversion est effectuée lorsque le message est envoyé et lorsque l'une des méthodes de lecture ou d'extraction est appelée sur un message qui a été reçu.

La page de codes et le codage numérique utilisés pour écrire ou définir le contenu d'un message sont définis en tant qu'attributs de la destination. La page de codes de destination et le codage numérique peuvent être modifiés administrativement. Une application peut également remplacer la page de codes de destination et le codage en définissant les propriétés de message qui contrôlent l'écriture ou la définition du contenu du message.

Si vous souhaitez convertir le codage des nombres lorsqu'un message `JMSBytesMessage` est envoyé à une destination qui n'est pas définie en tant que codage `Native` , vous devez définir la propriété de message `JMS_IBM_ENCODING` avant d'envoyer le message. Si vous suivez le modèle "le récepteur est correct" ou si vous échangez des messages entre des applications JMS , l'application n'a pas besoin de définir `JMS_IBM_ENCODING`. Dans la plupart des cas, vous pouvez conserver la propriété `Encoding` sous la forme `Native`.

Pour les messages `JMSStreamMessage`, `JMSMapMessage`et `JMSTextMessage` , les propriétés d'identificateur de jeu de caractères de la destination sont utilisées. Le codage est ignoré lors de l'envoi car les nombres sont écrits au format texte. Le programme d'application client JMS n'a pas besoin de définir `JMS_IBM_CHARACTER_SET` avant d'envoyer le message si la propriété de jeu de caractères de destination doit être appliquée.

Pour obtenir les données d'un message, une application appelle les méthodes de lecture ou d'obtention de message JMS . Les méthodes font référence à la page de codes et au codage définis dans l'en-tête de message précédent pour créer correctement les primitives et les objets Java .

La conversion de données client JMS répond aux besoins de la plupart des applications JMS qui échangent des messages entre un client JMS et un autre. Vous ne codez aucune conversion de données explicite. Vous n'utilisez pas la classe `java.nio.charset.Charset` , qui est généralement utilisée lors de l'écriture de texte dans un fichier. Les méthodes `writeString` et `setString` font la conversion pour vous.

Pour plus de détails sur la conversion des données client JMS , voir «Conversion et codage des messages du client JMS», à la page 184.

[1] "JMS Client" fait référence au IBM MQ classes for JMS qui implémente l'interface JMS , qui s'exécute en mode client ou en mode liaisons.

Conversion des données d'application

Une application client JMS peut effectuer une conversion de données de type caractères explicite à l'aide de la classe `java.nio.charset.Charset` ; voir les exemples dans Figure 14, à la page 176 et Figure 15, à la page 176. Les données de chaîne sont converties en octets, à l'aide de la méthode `getBytes` , et envoyées en tant qu'octets. Les octets sont reconvertis en texte à l'aide d'un constructeur `String` qui prend un tableau d'octets et un `Charset`. Les données de type caractères sont converties à l'aide des méthodes `encode` et `decode` `Charset` . En règle générale, le message est envoyé ou reçu en tant que `JMSBytesMessage`, car la partie du message d'un `JMSBytesMessage` ne contient rien d'autre que les données écrites par l'application.[2]. Vous pouvez également envoyer et recevoir des octets à l'aide de `JMSStreamMessage`, `JMSMapMessage`ou `JMSObjectMessage`.

Il n'existe aucune méthode Java pour coder et décoder les octets qui contiennent des données numériques représentées dans différents formats de codage. Les données numériques sont codées et décodées automatiquement à l'aide des méthodes de lecture et d'écriture `JMSMessage` numériques. Les méthodes de lecture et d'écriture utilisent la valeur de l'attribut `JMS_IBM_ENCODING` des données de message.

La conversion des données d'application est généralement utilisée si un client JMS envoie ou reçoit un message formaté d'une application nonJMS . Un message formaté contient du texte, des données numériques et des données d'octets organisées en fonction de la longueur des zones de données. A moins que l'application nonJMS n'ait spécifié le format de message "MQSTR ", le message est construit en tant que `JMSBytesMessage`. Pour recevoir des données de message formatées dans un `JMSBytesMessage` , vous devez appeler une séquence de méthodes. Les méthodes doivent être appelées dans l'ordre dans lequel les zones ont été écrites dans le message. Si les zones sont numériques, vous devez connaître le codage et la longueur des données numériques. Si l'une des zones contient des données d'octet ou de texte, vous devez connaître la longueur des données d'octet dans le message. Il existe deux façons de convertir un message formaté en objet Java facile à utiliser.

1. Construisez une classe Java correspondant à l'enregistrement, pour encapsuler la lecture et l'écriture du message. L'accès aux données de l'enregistrement se fait avec les méthodes get et set de la classe.
2. Construisez une classe Java correspondant à l'enregistrement en étendant la classe `com.ibm.mq.headers` . L'accès aux données de la classe se fait avec des accesseurs spécifiques au type du formulaire, `getStringValue(`*`fieldname`*`)` ;

Voir «Echange d'un enregistrement formaté avec une application nonJMS», à la page 192.

Conversion des données du gestionnaire de files d'attente

La conversion de page de codes peut être effectuée par le gestionnaire de files d'attente lorsqu'un programme client JMS reçoit un message. La conversion est la même que celle effectuée pour un programme C. Un programme C définit `MQGMO_CONVERT` en tant qu'option de paramètre `MQGET` `GetMsgOpts` ; voir Figure 13, à la page 176. Un gestionnaire de files d'attente effectue une conversion pour un programme client JMS qui reçoit un message, si la propriété de destination `WMQ_RECEIVE_CONVERSION` est définie sur `WMQ_RECEIVE_CONVERSION_QMGR`, le programme client JMS peut également définir la propriété de destination ; voir Figure 12, à la page 173.

[2] Une exception: les données écrites à l'aide de `writeUTF` commencent par une zone de longueur de 2 octets

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Ou

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figure 12. Activer la conversion des données du gestionnaire de files d'attente

L'avantage principal de la conversion de gestionnaire de files d'attente réside dans l'échange de messages avec des applications nonJMS . Si la zone `Format` du message est définie et que le jeu de caractères cible, ou le codage, est différent du message, le gestionnaire de files d'attente effectue une conversion de données pour l'application cible, si l'application le demande. Le gestionnaire de files d'attente convertit les données de message formatées en fonction de l'un des types de message IBM MQ prédéfinis, tels qu'un en-tête CICS bridge (`MQCIH`). Si la zone `Format` est définie par l'utilisateur, le gestionnaire de files d'attente recherche un exit de conversion de données portant le nom indiqué dans la zone `Format` .

La conversion des données du gestionnaire de files d'attente est utilisée de manière optimale avec le modèle de conception "récepteur rend bon" . Un client JMS émetteur n'a pas besoin d'effectuer de conversion. Un programme de réception nonJMS s'appuie sur l'exit de conversion pour s'assurer que le message est distribué dans la page de codes et le codage requis. Avec un client JMS émetteur et un récepteur nonJMS , l'exemple s'applique à IBM MQ.

Vous pouvez créer un exit de conversion de données, à l'aide de l'utilitaire d'exit de conversion de données, **`crtmqcvx`**, pour permettre au gestionnaire de files d'attente de convertir vos propres données formatées d'enregistrement. Vous pouvez générer votre propre format d'enregistrement, utiliser `com.ibm.mq.headers` pour y accéder en tant que classe Java et utiliser votre propre exit de conversion pour le convertir. Sous z/OS , l'utilitaire est appelé **`CSQUCVX`**et sous IBM i, **`CVTMQMDTA`**. Voir «Echange d'un enregistrement formaté avec une application nonJMS», à la page 192.

Conversion de données de canal de transmission de messages

IBM MQ Les canaux émetteur, serveur, récepteur de cluster et émetteur de cluster ont une option de conversion de message, `CONVERT`. Le contenu d'un message peut éventuellement être converti lorsqu'un message est envoyé. La conversion a lieu à l'extrémité émettrice du canal. La définition de récepteur de cluster est utilisée pour définir automatiquement le canal émetteur de cluster correspondant.

La conversion de données par canaux de message est généralement utilisée s'il n'est pas possible d'utiliser d'autres formes de conversion.

Choix d'une approche de la conversion de message: "le récepteur est bon"

L'approche habituelle dans la conception d'application IBM MQ pour la conversion de code est "le récepteur est bon". "Récepteur fait bon" réduit le nombre de conversions de message. Elle évite également le problème d'erreurs de canal inattendues si la conversion de messages échoue sur un gestionnaire de files d'attente intermédiaire lors du transfert de messages. La règle "receiver make good" n'est pas respectée s'il existe une raison pour laquelle le récepteur ne peut pas la faire. La plateforme de réception peut ne pas avoir le jeu de caractères approprié, par exemple.

"Le récepteur est bon" est également un bon guide général pour les applications client JMS . Mais dans des cas spécifiques, la conversion au jeu de caractères correct à la source peut être plus efficace. La conversion à partir de la représentation interne de la machine virtuelle Java doit être effectuée lorsqu'un message contenant du texte ou des types numériques est envoyé. La conversion vers le jeu de caractères requis par le récepteur, si le récepteur n'est pas un client JMS , peut supprimer la nécessité pour le destinataire nonJMS d'effectuer la conversion. Si le destinataire est un client JMS , il va tout de même

effectuer une nouvelle conversion pour décoder les données de message et créer des primitives et des objets Java .

La différence entre les applications client JMS et les applications écrites dans un langage tel que C est que Java doit effectuer une conversion de données. Une application Java doit convertir les nombres et le texte de leur représentation interne dans un format codé utilisé dans les messages.

En définissant la destination ou les propriétés de message, vous pouvez définir le jeu de caractères et le codage utilisés par IBM MQ pour coder les numéros et le texte dans les messages. Normalement, vous devez conserver le jeu de caractères `1208` et le codage `Native`.

IBM MQ ne convertit pas les tableaux d'octets. Pour coder des chaînes et des tableaux de caractères en tableaux d'octets, utilisez le package `java.nio.charset` . `Charset` indique le jeu de caractères utilisé pour convertir une chaîne ou un tableau de caractères en tableau d'octets. Vous pouvez également décoder un tableau d'octets en chaîne ou en tableau de caractères à l'aide d'un `Charset`. Il n'est pas recommandé de s'appuyer sur `java.nio.charset.Charset.defaultCodePage` lors du codage des chaînes et des tableaux de caractères. La valeur par défaut `Charset` est généralement `windows-1252` sous Windowset `UTF-8` sous AIX and Linux. `windows-1252` est un jeu de caractères à un octet et `UTF-8` est un jeu de caractères à plusieurs octets.

Conservez généralement les valeurs par défaut `UTF-8` et `Native` pour les propriétés de codage et de jeu de caractères de destination lors de l'échange de messages avec d'autres applications JMS . Si vous échangez des messages contenant des nombres ou du texte avec une application JMS , choisissez l'un des types de message `JMSTextMessage`, `JMSStreamMessage`, `JMSMapMessage`ou `JMSObjectMessage` qui vous convient. Il n'y a aucune autre tâche de conversion à effectuer.

Si vous échangez des messages avec des applications nonJMS qui utilisent un format d'enregistrement, cela est plus compliqué. Sauf si l'enregistrement entier contient du texte et peut être transféré en tant que `JMSTextMessage`, vous devez coder et décoder le texte dans l'application. Définissez le type de message de destination sur `MQ`et utilisez `JMSBytesMessage` pour éviter que IBM MQ classes for JMS n'ajoute des informations d'en-tête et de balisage supplémentaires aux données de message. Utilisez les méthodes `JMSBytesMessage` pour écrire des nombres et des octets, et la classe `Charset` convertit explicitement le texte en tableaux d'octets. Un certain nombre de facteurs peuvent influencer votre choix de jeu de caractères:

- Performances: pouvez-vous réduire le nombre de conversions en transformant du texte en un jeu de caractères utilisé sur le plus grand nombre de serveurs?
- Uniformité: transférez tous les messages dans le même jeu de caractères.
- Richesse: Quels jeux de caractères ont tous les points de code que les applications doivent utiliser?
- Simplicité: les jeux de caractères à un octet sont plus simples à utiliser que les jeux de caractères à longueur variable et à plusieurs octets.

Voir «Echange d'un enregistrement formaté avec une application nonJMS», à la page 192. pour des exemples de conversion de messages échangés avec des applications nonJMS .

Exemples

Tableau des types de message et des types de conversion

Tableau 31. Types de message et types de conversion

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
`JMSObjectMessage`				`getObject` `setObject`

Tableau 31. Types de message et types de conversion (suite)

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
JMSTextMessage	getText setText			
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Figure 13. Fragment de code de amqsget0.c

Envoi et réception de texte dans un JMSBytesMessage

Le code dans Figure 14, à la page 176 envoie une chaîne dans un BytesMessage. Par souci de simplicité, l'exemple envoie une chaîne unique, pour laquelle un JMSTextMessage est plus approprié. Pour recevoir une chaîne de texte en octets contenant un mélange de types, vous devez connaître la longueur de la chaîne en octets, appelée *TEXT_LENGTH* dans Figure 15, à la page 176. Même pour une chaîne comportant un nombre fixe de caractères, la longueur de la représentation en octets peut être plus longue.

```
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);
```

Figure 14. Envoi d'un String dans 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);
```

Figure 15. Réception d'un String à partir d'un JMSBytesMessage

Concepts associés

Conversion et codage des messages du client JMS
Les méthodes que vous utilisez pour effectuer la conversion et le codage des messages du client JMS sont répertoriées, avec des exemples de code de chaque type de conversion.

Conversion des données du gestionnaire de files d'attente
La conversion des données du gestionnaire de files d'attente a toujours été disponible pour les applications nonJMS qui reçoivent des messages des clients JMS . Les clients JMS qui reçoivent des

messages utilisent également la conversion des données du gestionnaire de files d'attente, qui est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application nonJMS
Suivez les étapes suggérées dans cette tâche pour concevoir et générer un exit de conversion de données et une application client JMS pouvant échanger des messages avec une application nonJMS à l'aide de JMSBytesMessage. L'échange d'un message formaté avec une application nonJMS peut avoir lieu avec ou sans appel d'un exit de conversion de données.

Référence associée

Types de message JMS et conversion
Le choix du type de message affecte votre approche de la conversion de message. L'interaction entre la conversion de message et le type de message est décrite pour les types de message JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessageet JMSBytesMessage.

Types de message JMS et conversion

Le choix du type de message affecte votre approche de la conversion de message. L'interaction entre la conversion de message et le type de message est décrite pour les types de message JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessageet JMSBytesMessage.

JMSObjectMessage

JMSObjectMessage contient un objet, et tous les objets auxquels il fait référence, sérialisés dans un flot d'octets par la machine virtuelle Java. Le texte est sérialisé dans UTF-8et limité à des chaînes ou à des tableaux de caractères de moins de 65534 octets. Un avantage de JMSObjectMessage est que les applications ne sont pas impliquées dans les problèmes de conversion de données tant qu'elles utilisent uniquement les méthodes et les attributs de l'objet. JMSObjectMessage fournit la conversion de données pour les objets complexes sans que le programmeur d'application ne considère comment coder un objet dans un message. L'inconvénient de l'utilisation de JMSObjectMessage est qu'elle ne peut être échangée qu'avec d'autres applications JMS . En choisissant l'un des autres types de message JMS , il est possible d'échanger des messages JMS avec des applications nonJMS .

La «Envoi et réception d'un JMSObjectMessage», à la page 180 montre un objet String échangé dans un message.

Une application client JMS peut recevoir un JMSObjectMessage uniquement dans un message comportant un corps de style JMS. La destination doit spécifier un corps de style JMS .

JMSTextMessage

JMSTextMessage contient une chaîne de texte unique. Lorsqu'un message texte est envoyé, le texte Format est défini sur "MQSTR ", WMQConstants.MQFMT_STRING. Le CodedCharacterSetId du texte est défini sur l'identificateur de jeu de caractères codés défini pour sa destination. Le texte est codé dans CodedCharacterSetId par IBM MQ. Les zones CodedCharacterSetId et Format sont définies dans le descripteur de message, MQMD, ou dans les zones JMS d'un MQRFH2. Si le message est défini comme ayant un style de corps de message WMQ_MESSAGE_BODY_MQ ou si le style de corps n'est pas spécifié, mais que la destination cible est WMQ_TARGET_DEST_MQ, les zones de descripteur de message sont définies. Dans le cas contraire, le message comporte un JMS RFH2 et les zones sont définies dans la partie fixe du MQRFH2.

Une application peut remplacer l'identificateur de jeu de caractères codés défini pour une destination. Il doit définir la propriété de message JMS_IBM_CHARACTER_SET sur un identificateur de jeu de caractères codés ; voir l'exemple dans «Envoi et réception d'un JMSTextmessage», à la page 180.

Lorsque le client JMS appelle la méthode consumer.receive , la conversion du gestionnaire de files d'attente est facultative. La conversion du gestionnaire de files d'attente est activée en définissant la propriété de destination WMQ_RECEIVE_CONVERSION sur WMQ_RECEIVE_CONVERSION_QMGR. Le gestionnaire de files d'attente convertit le message texte du JMS_IBM_CHARACTER_SET spécifié pour le message avant de le transférer vers le client JMS . Le jeu de caractères du message converti est 1208, UTF-8, sauf si la destination a un WMQ_RECEIVE_CCSIDdifférent. Le CodedCharacterSetId du

message qui fait référence à JMSTextMessage est mis à jour avec l'ID de jeu de caractères cible. Le texte est décodé à partir du jeu de caractères cible en Unicode par la méthode getText ; voir l'exemple dans «Envoi et réception d'un JMSTextmessage», à la page 180.

Un JMSTextMessage peut être envoyé dans un corps de message de style MQ, sans en-tête JMS MQRFH2 . La valeur des attributs de destination, WMQ_MESSAGE_BODY et WMQ_TARGET_DEST , détermine le style du corps du message, sauf s'il est remplacé par l'application. L'application peut remplacer les valeurs définies sur la destination en appelant destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) ou destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Si vous envoyez un JMSTextMessage avec un corps de style MQ en l'envoyant à une destination avec WMQ_MESSAGE_BODY défini sur WMQ_MESSAGE_BODY_MQ, vous ne pouvez pas le recevoir en tant que JMSTextMessage de la même destination. Tous les messages reçus d'une destination avec WMQ_MESSAGE_BODY défini sur WMQ_MESSAGE_BODY_MQ sont reçus en tant que JMSBytesMessage. Si vous tentez de recevoir le message en tant que JMSTextMessage , une exception est générée, ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to jakarta (or javax).jms.TextMessage.

Remarque : Le texte d'un JMSBytesMessage n'est pas converti par le client JMS. Le client peut uniquement recevoir le texte du message sous la forme d'un tableau d'octets. Si la conversion du gestionnaire de files d'attente est activée, le texte est converti par le gestionnaire de files d'attente, mais le client JMS doit toujours le recevoir sous forme de tableau d'octets dans un JMSBytesMessage.

Il est généralement préférable d'utiliser la propriété WMQ_TARGET_DEST pour contrôler si un JMSTextMessage est envoyé avec un style de corps MQ ou JMS. Vous pouvez ensuite recevoir le message d'une destination pour laquelle WMQ_TARGET_DEST est défini sur WMQ_TARGET_DEST_MQ ou WMQ_TARGET_DEST_JMS. WMQ_TARGET_DEST n'a aucun effet sur le récepteur.

JMSMapMessage et JMSStreamMessage

Ces deux types de message JMS sont similaires. Vous pouvez lire et écrire des types primitifs dans les messages à l'aide de méthodes basées sur les interfaces DataInputStream et DataOutputStream ; voir «Tableau des types de message et des types de conversion», à la page 182. Les détails sont décrits dans «Conversion et codage des messages du client JMS», à la page 184. Chaque primitive est balisée ; voir «Corps du message JMS», à la page 168.

Les données numériques sont lues et écrites dans le message codé sous forme de texte XML. Aucune référence n'est faite à la propriété de destination JMS_IBM_ENCODING. Les données texte sont traitées de la même manière que le texte dans un JMSTextMessage. Si vous examinez le contenu du message créé par l'exemple dans Figure 20, à la page 181, toutes les données du message seront au format EBCDIC car elles ont été envoyées avec une valeur de jeu de caractères de 37.

Vous pouvez envoyer plusieurs éléments dans un JMSMapMessage ou un JMSStreamMessage.

Vous pouvez extraire les éléments de données individuels par nom à partir d'un JMSMapMessageou par position à partir d'un JMSStreamMessage. Chaque élément est décodé lorsqu'une méthode get ou read est appelée à l'aide de la valeur CodedCharacterSetId stockée dans le message. Si la méthode utilisée pour extraire l'élément renvoie un type différent de celui qui a été envoyé, le type est converti. Si le type ne peut pas être converti, une exception est émise. Voir Class JMSStreamMessage pour plus de détails. L'exemple dans «Envoi de données dans JMSStreamMessage et JMSMapMessage», à la page 181 illustre la conversion de type et l'extraction du contenu JMSMapMessage hors séquence.

La zone MQRFH2.format pour JMSMapMessage et JMSStreamMessage est définie sur "MQSTR ". Si la propriété de destination WMQ_RECEIVE_CONVERSION est définie sur WMQ_RECEIVE_CONVERSION_QMGR, les données de message sont converties par le gestionnaire de files d'attente avant d'être envoyées au client JMS . Le MQRFH2.CodedCharacterSetId du message est le WMQ_RECEIVE_CCSID de la destination. Le MQRFH2.Encoding est Native. Si WMQ_RECEIVE_CONVERSION est WMQ_RECEIVE_CONVERSION_CLIENT_MSG le CodedCharacterSetId et Encoding de MQRFH2 est la valeur définie par l'expéditeur.

Une application client JMS peut recevoir un JMSMapMessage ou un JMSStreamMessage uniquement dans un message comportant un corps de style JMSet à partir d'une destination qui ne spécifie pas de corps de style MQ .

JMSBytesMessage

Un JMSBytesMessage peut contenir plusieurs types primitifs. Vous pouvez lire et écrire des types primitifs dans les messages à l'aide de méthodes basées sur les interfaces DataInputStream et DataOutputStream ; voir «Tableau des types de message et des types de conversion», à la page 182. Les détails sont décrits dans «Types de message JMS et conversion», à la page 177.

Le codage des données numériques dans le message est contrôlé par la valeur de JMS_IBM_ENCODING qui est définie avant l'écriture des données numériques dans JMSBytesMessage. Une application peut remplacer le codage Native par défaut défini pour JMSBytesMessage en définissant la propriété de message JMS_IBM_ENCODING.

Les données texte peuvent être lues et écrites en UTF-8 à l'aide de readUTF et writeUTF, ou en Unicode à l'aide des méthodes readChar et writeChar . Aucune méthode n'utilise CodedCharacterSetId. Le client JMS peut également coder et décoder le texte en octets à l'aide de la classe Charset . Il transfère les octets entre la machine virtuelle Java et le message sans que IBM MQ classes for JMS n'effectue de conversion ; voir «Envoi et réception de texte dans un JMSBytesMessage», à la page 181.

Une JMSBytesMessage envoyée à une application MQ est généralement envoyée dans un corps de message de style MQ, sans en-tête JMS MQRFH2 . S'il est envoyé à une application JMS , le style de corps du message est généralement JMS. La valeur des attributs de destination, WMQ_MESSAGE_BODY et WMQ_TARGET_DEST , détermine le style du corps du message, sauf s'il est remplacé par l'application. L'application peut remplacer les valeurs définies sur la destination en appelant destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) ou destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Si vous envoyez un JMSBytesMessage avec un corps de style MQ , vous pouvez recevoir le message d'une destination qui définit un style de corps de message MQ ou JMS . Si vous envoyez un JMSBytesMessage avec un corps de style JMS, vous devez recevoir le message d'une destination qui définit un style de corps de message JMS . Si vous ne le faites pas, MQRFH2 est traité comme faisant partie des données de message utilisateur, ce qui peut ne pas être ce que vous attendez.

Qu'un message ait un style de corps MQ ou JMS , la façon dont il est reçu n'est pas affectée par la définition de WMQ_TARGET_DEST.

Le message peut être transformé ultérieurement, par le gestionnaire de files d'attente, si un Format est fourni pour les données de message et que la conversion des données du gestionnaire de files d'attente est activée. N'utilisez pas la zone de format pour autre chose que la spécification du format des données de message, ou laissez cette zone à blanc, MQConstants.MQFMT_NONE

Vous pouvez envoyer plusieurs éléments dans un JMSBytesMessage. Chaque élément numérique est converti lorsque le message est envoyé à l'aide du codage défini pour le message.

Vous pouvez extraire les éléments de données individuels de JMSBytesMessage. Appelez les méthodes de lecture dans le même ordre que les méthodes d'écriture pour créer le message. Chaque élément numérique est converti lorsque le message est appelé à l'aide de la valeur Encoding stockée dans le message.

Contrairement à JMSMapMessage et JMSStreamMessage, JMSBytesMessage contient uniquement des données écrites par l'application. Aucune donnée supplémentaire n'est stockée dans les données de message, telles que les balises XML utilisées pour définir les éléments dans JMSMapMessage et JMSStreamMessage. Pour cette raison, utilisez JMSBytesMessage pour transférer des messages formatés pour d'autres applications.

La conversion entre JMSBytesMessage et DataInputStream et DataOutputStream est utile dans certaines applications. Le code basé sur l'exemple, «Lecture et écriture de messages à l'aide de DataInputStream et DataOutputStream», à la page 181, est nécessaire pour utiliser le package com.ibm.mq.header avec JMS.

Exemples

Envoi et réception d'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
```

Figure 16. Envoi et réception d'un JMSObjectMessage

Envoi et réception d'un JMSTextmessage

Un message texte ne peut pas contenir de texte dans des jeux de caractères différents. L'exemple montre du texte dans des jeux de caractères différents, envoyé dans deux messages différents.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figure 17. Envoyer un message texte dans le jeu de caractères défini par la destination

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figure 18. Envoi d'un message texte dans ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figure 19. Recevoir un message texte

Envoi de données dans JMSStreamMessage et 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
```

Figure 20. Envoi de données dans JMSStreamMessage et JMSMapMessage

Envoi et réception de texte dans un JMSBytesMessage

Le code dans Figure 21, à la page 181 envoie une chaîne dans un BytesMessage. Par souci de simplicité, l'exemple envoie une chaîne unique, pour laquelle un JMSTextMessage est plus approprié. Pour recevoir une chaîne de texte en octets contenant un mélange de types, vous devez connaître la longueur de la chaîne en octets, appelée *TEXT_LENGTH* dans Figure 22, à la page 181. Même pour une chaîne comportant un nombre fixe de caractères, la longueur de la représentation en octets peut être plus longue.

```
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);
```

Figure 21. Envoi d'un String dans 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);
```

Figure 22. Réception d'un String à partir d'un JMSBytesMessage

Lecture et écriture de messages à l'aide de DataInputStream et DataOutputStream

Le code dans Figure 23, à la page 182 crée un JMSBytesMessage à l'aide d'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);
```

Figure 23. Envoyer un JMSBytesMessage à l'aide d'un DataOutputStream

L'instruction qui définit la propriété JMS_IBM_ENCODING est mise en commentaire. L'instruction est valide si elle est écrite directement dans un message JMSBytesMessage, mais n'a aucun effet lors de l'écriture dans DataOutputStream. Les nombres qui sont écrits dans DataOutputStream sont codés dans le codage Native . Le paramètre JMS_IBM_ENCODING n'a aucun effet.

Le code dans Figure 24, à la page 182 reçoit un JMSBytesMessage à l'aide d'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)));
```

Figure 24. Réception d'un JMSBytesMessage à l'aide d'un DataInputStream

La page de codes est imprimée à l'aide de la propriété de page de codes des données de message d'entrée, JMS_IBM_CHARACTER_SET. En entrée, JMS_IBM_CHARACTER_SET est une page de codes Java et non un identificateur de jeu de caractères codés numérique.

Tableau des types de message et des types de conversion

Tableau 32. Types de message et types de conversion

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tableau 32. Types de message et types de conversion (suite)

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
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

Concepts associés

Approches de conversion de message JMS

Un certain nombre d'approches de conversion de données sont ouvertes aux concepteurs d'applications JMS . Ces approches ne sont pas exclusives ; certaines applications sont susceptibles d'utiliser une combinaison de ces approches. Si votre application échange uniquement du texte ou des messages uniquement avec d'autres applications JMS , vous n'envisagez normalement pas la conversion de données. La conversion de données est effectuée automatiquement pour vous, par IBM MQ.

Conversion et codage des messages du client JMS

Les méthodes que vous utilisez pour effectuer la conversion et le codage des messages du client JMS sont répertoriées, avec des exemples de code de chaque type de conversion.

Conversion des données du gestionnaire de files d'attente

La conversion des données du gestionnaire de files d'attente a toujours été disponible pour les applications nonJMS qui reçoivent des messages des clients JMS . Les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente, qui est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application nonJMS
Suivez les étapes suggérées dans cette tâche pour concevoir et générer un exit de conversion de données et une application client JMS pouvant échanger des messages avec une application nonJMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application nonJMS peut avoir lieu avec ou sans appel d'un exit de conversion de données.

Conversion et codage des messages du client JMS
Les méthodes que vous utilisez pour effectuer la conversion et le codage des messages du client JMS sont répertoriées, avec des exemples de code de chaque type de conversion.

La conversion et le codage se produisent lorsque les primitives ou les objets Java sont lus ou écrits dans et à partir des messages JMS . La conversion est appelée conversion de données client JMS pour la distinguer de la conversion de données du gestionnaire de files d'attente et de la conversion de données d'application. La conversion s'effectue strictement lorsque les données sont lues ou écrites dans un message JMS . Le texte est converti vers et depuis la représentation Unicode 16 bits interne[3]au jeu de caractères utilisé pour le texte dans les messages. Les données numériques sont converties en types numériques primitifs Java et dans le codage défini pour le message. L'exécution de la conversion et le type de conversion dépend du type de message JMS et de l'opération de lecture ou d'écriture.

Tableau 33, à la page 184 catégorise les méthodes de lecture et d'écriture pour différents types de message JMS en fonction du type de conversion effectuée. Les types de conversion sont décrits dans le texte qui suit le tableau.

Tableau 33. Types de message et types de conversion

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

`JMSBytesMessage`	`readUTF`
	`writeUTF`
		`readDouble`
		`readFloat`
		`readInt`
		`readLong`
		`readShort`
		`readUnsignedShort`
		`writeDouble`
		`writeFloat`
		`writeInt`
		`writeLong`
		`writeShort`
			`readBoolean`
			`readObject`
			`writeBoolean`
			`writeObject`
				`readByte`
				`readUnsignedByte`
				`readBytes`
				`readChar`
				`writeByte`
				`writeBytes`
				`writeChar`

[3] Certaines représentations Unicode nécessitent plus de 16 bits. Voir une référence Java SE.

Tableau 33. Types de message et types de conversion (suite)				
	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
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

Texte

La valeur par défaut CodedCharacterSetId pour une destination est 1208, UTF-8. Par défaut, le texte est converti à partir d'Unicode et envoyé en tant que chaîne de texte UTF-8 . A la réception, le texte est converti à partir du jeu de caractères codés dans le message reçu par le client, en Unicode.

Les méthodes setText et writeString convertissent le texte Unicode en jeu de caractères défini pour la destination. Une application peut remplacer le jeu de caractères de destination en définissant la propriété de message JMS_IBM_CHARACTER_SET. JMS_IBM_CHARACTER_SET, lors de l'envoi d'un message, doit être un identificateur de jeu de caractères codés numérique[4].

Les fragments de code de «Envoi et réception d'un JMSTextmessage», à la page 188 envoient deux messages. L'un est envoyé dans le jeu de caractères défini pour la destination et l'autre dans le jeu de caractères 37, défini par l'application.

Les méthodes getText et readString convertissent le texte du message à partir du jeu de caractères défini dans le message en Unicode. Les méthodes utilisent la page de codes définie dans la propriété de message JMS_IBM_CHARACTER_SET. La page de codes est mappée à partir de MQRFH2.CodedCharacterSetId sauf si le message est de type MQet qu'il ne comporte pas de MQRFH2. Si le message est de type MQ, sans MQRFH2, la page de codes est mappée à partir de MQMD.CodedCharacterSetId.

Le fragment de code dans Figure 29, à la page 188 reçoit le message qui a été envoyé à la destination. Le texte du message est converti de la page de codes IBM037 en Unicode.

Remarque : Un moyen simple de vérifier que le texte est converti en jeu de caractères codés 37 consiste à utiliser IBM MQ Explorer. Parcourez la file d'attente et affichez les propriétés du message avant de l'extraire.

[4] Lors de la réception d'un message, JMS_IBM_CHARACTER_SET est un nom de page de codes Java Charset .

Comparez le fragment de code dans Figure 28, à la page 188 avec le fragment de code incorrect dans Figure 25, à la page 186. Dans le fragment incorrect, la chaîne de texte est convertie deux fois, une fois par l'application, puis à nouveau par IBM MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figure 25. Conversion de page de codes incorrecte

La méthode `writeUTF` convertit le texte Unicode en 1208, UTF-8. La chaîne de texte est précédée d'une longueur de 2 octets. La longueur maximale de la chaîne de texte est de 65534 octets. La méthode `readUTF` lit un élément dans un message écrit par la méthode `writeUTF` . Il lit exactement le nombre d'octets écrits par la méthode `writeUTF` .

Numérique

Le codage numérique par défaut d'une destination est `Native`. La constante de codage `Native` pour Java a la valeur 273, `x'00000111'`, qui est la même pour toutes les plateformes. A la réception, les nombres du message sont correctement transformés en primitives Java numériques. La transformation utilise le codage défini dans le message et le type renvoyé par la méthode de lecture.

La méthode d'envoi convertit les nombres qui sont ajoutés à un message par `set` et `write` dans le codage numérique défini pour la destination. Le codage de destination peut être remplacé pour un message par une application définissant la propriété de message, `JMS_IBM_ENCODING` ; par exemple:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

Les méthodes numériques `get` et `read` convertissent les nombres du message à partir du codage numérique défini dans le message. Ils convertissent les nombres en type spécifié par la méthode `read` ou `get` ; voir The ENCODING property. Les méthodes utilisent le codage défini dans `JMS_IBM_ENCODING`. Le codage est mappé à partir de `MQRFH2.Encoding` sauf si le message est un message de type MQet qu'il n'a pas de `MQRFH2`. Si le message est de type MQ, sans `MQRFH2`, les méthodes utilisent le codage défini dans `MQMD.Encoding`.

L'exemple de la Figure 30, à la page 188 montre une application codant un nombre au format de destination et l'envoyant dans un `JMSStreamMessage`. Comparez l'exemple dans Figure 30, à la page 188 à l'exemple dans Figure 31, à la page 189. La différence est que `JMS_IBM_ENCODING` doit être défini dans un `JMSBytesMessage`.

Remarque : Un moyen simple de vérifier que le nombre est codé correctement consiste à utiliser IBM MQ Explorer. Parcourez la file d'attente et affichez les propriétés du message avant qu'il ne soit consommé.

Autre

Les méthodes `boolean` codent `true` et `false` en tant que `x'01'` et `x'00'` dans un `JMSByteMessage`, `JMSStreamMessage`et `JMSMapMessage`.

Les méthodes UTF codent et décodent Unicode en chaînes de texte UTF-8 . Les chaînes sont limitées à moins de 65536 caractères et sont précédées de la zone de longueur de 2 octets.

Les méthodes Object encapsulent les types primitifs en tant qu'objets. Les types numérique et texte sont codés ou convertis comme si les types primitifs avaient été lus ou écrits à l'aide des méthodes numérique et texte.

Aucun

Les méthodes `readByte`, `readBytes`, `readUnsignedByte`, `writeByte`et `writeBytes` obtiennent ou placent des octets uniques, ou des tableaux d'octets, entre l'application et le message sans conversion. Les méthodes `readChar` et `writeChar` obtiennent et placent des caractères Unicode de 2 octets entre l'application et le message sans conversion.

A l'aide des méthodes `readBytes` et `writeBytes` , l'application peut effectuer sa propre conversion de point de code, comme dans «Envoi et réception de texte dans un JMSBytesMessage», à la page 189.

IBM MQ n'effectue aucune conversion de page de codes dans le client car le message est un `JMSBytesMessage`et parce que les méthodes `readBytes` et `writeBytes` sont utilisées. Néanmoins, si les octets représentent du texte, assurez-vous que la page de codes utilisée par l'application correspond au jeu de caractères codés de la destination. Le message peut être converti à nouveau par un exit de conversion de gestionnaire de files d'attente. Il est également possible que le programme client JMS récepteur respecte la convention de conversion des tableaux d'octets représentant le texte du message en chaînes ou en caractères à l'aide de la propriété `JMS_IBM_CHARACTER_SET` du message.

Dans cet exemple, le client utilise le jeu de caractères codés de destination pour sa conversion:

```
bytes.writeBytes("In the destination code page".getBytes(
CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Le client peut également avoir choisi une page de codes, puis défini le jeu de caractères codés correspondant dans la propriété `JMS_IBM_CHARACTER_SET` du message. IBM MQ classes for Java Utilisez `JMS_IBM_CHARACTER_SET` pour définir la zone `CodedCharacterSetId` dans les propriétés JMS du `MQRFH2`ou dans le descripteur de message, `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Si un tableau d'octets est écrit dans un `JMSStringMessage` ou `JMSMapMessage`, IBM MQ classes for JMS n'effectue pas de conversion de données, car les octets sont saisis en tant que données hexadécimales et non en tant que texte dans `JMSStringMessage` et `JMSMapMessage`.

Si les octets représentent des caractères dans votre application, vous devez prendre en compte les points de code à lire et à écrire dans le message. Le code de Figure 26, à la page 187 suit la convention d'utilisation du jeu de caractères codés de destination. Si vous créez la chaîne à l'aide du jeu de caractères par défaut de la machine virtuelle Java, le contenu en octets dépend de la plateforme. Une machine virtuelle Java sous Windows a généralement un `Charset` par défaut de `windows-1252`et AIX and Linux a `UTF-8`. Pour l'échange entre Windowset AIX and Linux, vous devez sélectionner une page de codes explicite pour l'échange de texte sous forme d'octets.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
.getIntProperty(WMQConstants.WMQ_CCSID))));
```

Figure 26. Ecriture d'octets représentant une chaîne dans un JMS`StreamMessage` à l'aide du jeu de caractères de destination

Exemples

[5] `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

Envoi et réception d'un JMSTextmessage

Un message texte ne peut pas contenir de texte dans des jeux de caractères différents. L'exemple montre du texte dans des jeux de caractères différents, envoyé dans deux messages différents.

```
TextMessage tmo = session.createTextMessage();
tmo.setText("Sent in the character set defined for the destination");
producer.send(tmo);
```

Figure 27. Envoyer un message texte dans le jeu de caractères défini par la destination

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText("Sent in EBCDIC character set 37");
producer.send(tmo);
```

Figure 28. Envoi d'un message texte dans ccsid 37

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Figure 29. Recevoir un message texte

Exemples de codage

Exemples illustrant un nombre envoyé dans le codage défini pour une destination. Notez que vous devez définir la propriété JMS_IBM_ENCODING d'un JMSBytesMessage sur la valeur spécifiée pour la destination.

```
StreamMessage smo = session.createStreamMessage();
smo.writeInt(256);
producer.send(smo);
...
StreamMessage smi = (StreamMessage)consumer.receive();
System.out.println(smi.readInt());
...
256
```

Figure 30. Envoi d'un nombre à l'aide du codage de destination dans 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
```

Figure 31. Envoi d'un nombre à l'aide du codage de destination dans un JMSBytesMessage

Envoi et réception de texte dans un JMSBytesMessage

Le code dans Figure 32, à la page 189 envoie une chaîne dans un BytesMessage. Par souci de simplicité, l'exemple envoie une chaîne unique, pour laquelle un JMSTextMessage est plus approprié. Pour recevoir une chaîne de texte en octets contenant un mélange de types, vous devez connaître la longueur de la chaîne en octets, appelée *TEXT_LENGTH* dans Figure 33, à la page 189. Même pour une chaîne comportant un nombre fixe de caractères, la longueur de la représentation en octets peut être plus longue.

```
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);
```

Figure 32. Envoi d'un String dans 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);
```

Figure 33. Réception d'un String à partir d'un JMSBytesMessage

Concepts associés

Approches de conversion de message JMS

Un certain nombre d'approches de conversion de données sont ouvertes aux concepteurs d'applications JMS . Ces approches ne sont pas exclusives ; certaines applications sont susceptibles d'utiliser une combinaison de ces approches. Si votre application échange uniquement du texte ou des messages uniquement avec d'autres applications JMS , vous n'envisagez normalement pas la conversion de données. La conversion de données est effectuée automatiquement pour vous, par IBM MQ.

Conversion des données du gestionnaire de files d'attente

La conversion des données du gestionnaire de files d'attente a toujours été disponible pour les applications nonJMS qui reçoivent des messages des clients JMS . Les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente, qui est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application nonJMS

Suivez les étapes suggérées dans cette tâche pour concevoir et générer un exit de conversion de données et une application client JMS pouvant échanger des messages avec une application nonJMS à l'aide de JMSBytesMessage. L'échange d'un message formaté avec une application nonJMS peut avoir lieu avec ou sans appel d'un exit de conversion de données.

Référence associée

Types de message JMS et conversion

Le choix du type de message affecte votre approche de la conversion de message. L'interaction entre la conversion de message et le type de message est décrite pour les types de message JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessageet JMSBytesMessage.

Conversion des données du gestionnaire de files d'attente

La conversion des données du gestionnaire de files d'attente a toujours été disponible pour les applications nonJMS qui reçoivent des messages des clients JMS . Les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente, qui est facultative.

Le gestionnaire de files d'attente peut convertir des données de type caractères et numériques dans des données de message à l'aide des valeurs CodedCharacterSetId, Encodinget Format définies pour les données de message. Pour les applications nonJMS , la fonction de conversion a toujours été disponible en définissant l'option GetMessage, GMO_CONVERT.

Le gestionnaire de files d'attente peut convertir les messages envoyés aux clients JMS . La conversion du gestionnaire de files d'attente est contrôlée en définissant la propriété de destination WMQ_RECEIVE_CONVERSIONsur WMQ_RECEIVE_CONVERSION_QMGRou WMQ_RECEIVE_CONVERSION_CLIENT_MSG. L'application peut modifier le paramètre de destination:

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Ou

```
((MQDestination)destination).setReceiveConversion
              (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figure 34. Activer la conversion des données du gestionnaire de files d'attente

La conversion des données du gestionnaire de files d'attente pour un client JMS a lieu lorsque le client appelle une méthode consumer.receive . Les données texte sont transformées en UTF-8 (1208) par défaut. Les méthodes de lecture et d'obtention suivantes décodent le texte dans les données reçues à partir de UTF-8, en créant des primitives de texte Java dans leur codage Unicode interne. UTF-8 n'est pas le seul jeu de caractères cible issu de la conversion des données du gestionnaire de files d'attente. Vous pouvez choisir un autre CCSID en définissant la propriété de destination WMQ_RECEIVE_CCSID .

Une application peut également modifier le paramètre de destination, par exemple en lui affectant la valeur 437, DOS-US:

```
((MQDestination)destination).setIntProperty
(WMQConstants.WMQ_RECEIVE_CCSID, 437);
```

Ou

```
((MQDestination)destination).setReceiveCCSID(437);
```

Figure 35. Définir le jeu de caractères codés cible pour la conversion du gestionnaire de files d'attente

La raison de la modification de WMQ_RECEIVE_CCSID est spécialisée ; le CCSID choisi ne fait aucune différence avec les objets de texte créés dans la machine virtuelle Java. Toutefois, certaines JVM, sur certaines plateformes, peuvent ne pas être en mesure de gérer la conversion du CCSID du texte du message en Unicode. Cette option vous permet de choisir le CCSID pour tout texte transmis au client dans le message. Certaines plateformes client JMS ont rencontré des problèmes lors de la distribution de texte de message en UTF-8.

Le code JMS est équivalent au texte en gras du code C dans Figure 36, à la page 191,

```
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                      */
```

Figure 36. Fragment de code de amqsget0.c

Remarque :

La conversion du gestionnaire de files d'attente est effectuée uniquement sur les données de message ayant un format IBM MQ connu. MQSTR, ou MQCIH sont des exemples de formats connus prédéfinis. Un format connu peut également être défini par l'utilisateur, à condition que vous ayez fourni un exit de conversion de données.

Les messages construits en tant que JMSTextMessage, JMSMapMessage et JMSStreamMessageont un format MQSTR et peuvent être convertis par le gestionnaire de files d'attente.

Concepts associés

Approches de conversion de message JMS

Un certain nombre d'approches de conversion de données sont ouvertes aux concepteurs d'applications JMS . Ces approches ne sont pas exclusives ; certaines applications sont susceptibles d'utiliser une combinaison de ces approches. Si votre application échange uniquement du texte ou des messages uniquement avec d'autres applications JMS , vous n'envisagez normalement pas la conversion de données. La conversion de données est effectuée automatiquement pour vous, par IBM MQ.

Conversion et codage des messages du client JMS

Les méthodes que vous utilisez pour effectuer la conversion et le codage des messages du client JMS sont répertoriées, avec des exemples de code de chaque type de conversion.

«Appel de l'exit de conversion de données», à la page 1011
Un exit de conversion de données est un exit écrit par l'utilisateur qui reçoit le contrôle lors du traitement d'un appel MQGET.

Tâches associées

Echange d'un enregistrement formaté avec une application nonJMS
Suivez les étapes suggérées dans cette tâche pour concevoir et générer un exit de conversion de données et une application client JMS pouvant échanger des messages avec une application nonJMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application nonJMS peut avoir lieu avec ou sans appel d'un exit de conversion de données.

Référence associée

Types de message JMS et conversion
Le choix du type de message affecte votre approche de la conversion de message. L'interaction entre la conversion de message et le type de message est décrite pour les types de message JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`et `JMSBytesMessage`.

Echange d'un enregistrement formaté avec une application nonJMS
Suivez les étapes suggérées dans cette tâche pour concevoir et générer un exit de conversion de données et une application client JMS pouvant échanger des messages avec une application nonJMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application nonJMS peut avoir lieu avec ou sans appel d'un exit de conversion de données.

Avant de commencer

Vous pouvez concevoir une solution plus simple pour échanger des messages avec une application nonJMS à l'aide d'un `JMSTextMessage`. Éliminez cette possibilité avant de suivre les étapes de cette tâche.

Pourquoi et quand exécuter cette tâche

Un client JMS est plus facile à écrire s'il n'est pas impliqué dans les détails du formatage des messages JMS échangés avec d'autres clients JMS . Tant que le type de message est `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`ou `JMSObjectMessage`, IBM MQ s'occupe des détails du formatage du message. IBM MQ traite des différences de pages de codes et de codage numérique sur différentes plateformes.

Vous pouvez utiliser ces types de message pour échanger des messages avec des applications nonJMS . Pour ce faire, vous devez comprendre comment ces messages sont générés par IBM MQ classes for JMS. Vous pouvez modifier l'application nonJMS pour interpréter les messages ; voir «Mappage de messages JMS en messages IBM MQ», à la page 152.

L'utilisation de l'un de ces types de message présente l'avantage suivant: la programmation du client JMS ne dépend pas du type d'application avec lequel il échange des messages. Un inconvénient est qu'il peut nécessiter une modification d'un autre programme et que vous ne pourrez peut-être pas modifier l'autre programme.

Une autre approche consiste à écrire une application client JMS capable de gérer les formats de message existants. Les messages existants sont souvent au format fixe et contiennent une combinaison de données non formatées, de texte et de nombres. Utilisez les étapes de cette tâche et l'exemple de client JMS dans «Ecriture de classes pour encapsuler une présentation d'enregistrement dans un JMSBytesMessage», à la page 196comme point de départ pour la génération d'un client JMS pouvant échanger des enregistrements formatés avec des applications nonJMS .

Procédure

1. Définissez la présentation de l'enregistrement ou utilisez l'une des classes d'en-tête IBM MQ prédéfinies.

Pour la gestion des en-têtes IBM MQ prédéfinis, voir Gestion des en-têtes de message IBM MQ.

Figure 37, à la page 194 est un exemple de présentation d'enregistrement de longueur fixe définie par l'utilisateur, qui peut être traitée par l'utilitaire de conversion de données.

2. Créez l'exit de conversion de données.

   Suivez les instructions de la rubrique Ecriture d'un programme d'exit de conversion de données pour écrire un exit de conversion de données.

   Pour tester l'exemple dans «Ecriture de classes pour encapsuler une présentation d'enregistrement dans un JMSBytesMessage», à la page 196, nommez l'exit de conversion de données `MYRECORD`.
3. Ecrivez les classes Java pour encapsuler la présentation de l'enregistrement, ainsi que l'enregistrement d'envoi et de réception. Deux approches possibles sont les suivantes:
   - Ecrire une classe qui lit et écrit le fichier `JMSBytesMessage` qui contient l'enregistrement ; voir «Ecriture de classes pour encapsuler une présentation d'enregistrement dans un JMSBytesMessage», à la page 196.
   - Ecrivez une classe étendant `com.ibm.mq.header.Header` pour définir la structure de données de l'enregistrement ; voir Création de classes pour de nouveaux types d'en-tête.
4. Choisissez le jeu de caractères codés dans le cadre de l'échange de messages.

   Voir Choix d'une approche de la conversion de message: le récepteur est bon.
5. Configurez la destination pour l'échange de messages de type MQ, sans en-tête JMS `MQRFH2` .

   La destination d'envoi et de réception doit être configurée pour échanger des messages de type MQ. Vous pouvez utiliser la même destination pour l'envoi et la réception.

   L'application peut remplacer la propriété de corps du message de destination:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   L'exemple de la section «Ecriture de classes pour encapsuler une présentation d'enregistrement dans un JMSBytesMessage», à la page 196 remplace la propriété du corps du message de destination, ce qui garantit l'envoi d'un message de style MQ.
6. Tester la solution avec des applications JMS et nonJMS

   Les outils utiles pour tester un exit de conversion de données sont les suivants:
   - L'exemple de programme `amqsgetc0.c` est utile pour tester la réception d'un message envoyé par un client JMS . Consultez les modifications suggérées pour utiliser l'exemple d'en-tête, `RECORD.h`, dans Figure 38, à la page 195. Avec les modifications, `amqsgetc0.c` reçoit un message envoyé par l'exemple de client JMS , `TryMyRecord.java` ; voir «Ecriture de classes pour encapsuler une présentation d'enregistrement dans un JMSBytesMessage», à la page 196.
   - L'exemple de programme de navigation IBM MQ , `amqsbcg0.c`, est utile pour inspecter le contenu de l'en-tête de message, l'en-tête JMS , `MQRFH2`et le contenu du message.
   - Le programme **`rfhutil`** , précédemment disponible dans SupportPac IH03, permet de capturer et de stocker des messages de test dans des fichiers, puis de les utiliser pour générer des flux de messages. Les messages de sortie peuvent également être lus et affichés dans divers formats. Les formats incluent deux types de XML ainsi qu'une correspondance avec un copybook COBOL. Les données peuvent être au format EBCDIC ou ASCII. Un en-tête RFH2 peut être ajouté au message avant que celui-ci ne soit envoyé.

   Si vous tentez de recevoir des messages à l'aide de l'exemple de programme `amqsgetc0.c` modifié et que vous obtenez une erreur avec le code anomalie `2080`, vérifiez si le message comporte un `MQRFH2`. Les modifications supposent que le message a été envoyé à une destination qui ne spécifie pas `MQRFH2`.

Exemples

```
struct RECORD { MQCHAR StrucID[4];
                MQLONG Version;
                                MQLONG StructLength;
                                MQLONG Encoding;
                                MQLONG CodeCharSetId;
                                MQCHAR Format[8];
                                MQLONG Flags;
                                MQCHAR RecordData[32];
        };
```

Figure 37. RECORD.h

- Déclarez la structure de données 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);
```

- Modifiez l'appel MQGET pour utiliser RECORD,

  1. Avant modification:

```
    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. Après modification:

```
    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           */
```

- Modifiez l'instruction d'impression,

  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);
```

Figure 38. Modifiez amqsget0.c

Concepts associés

Approches de conversion de message JMS

Un certain nombre d'approches de conversion de données sont ouvertes aux concepteurs d'applications JMS . Ces approches ne sont pas exclusives ; certaines applications sont susceptibles d'utiliser une combinaison de ces approches. Si votre application échange uniquement du texte ou des messages uniquement avec d'autres applications JMS , vous n'envisagez normalement pas la conversion de données. La conversion de données est effectuée automatiquement pour vous, par IBM MQ.

Conversion et codage des messages du client JMS
Les méthodes que vous utilisez pour effectuer la conversion et le codage des messages du client JMS sont répertoriées, avec des exemples de code de chaque type de conversion.

Conversion des données du gestionnaire de files d'attente
La conversion des données du gestionnaire de files d'attente a toujours été disponible pour les applications nonJMS qui reçoivent des messages des clients JMS . Les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente, qui est facultative.

Utilitaire permettant de créer un code de sortie de conversion

Référence associée

Types de message JMS et conversion
Le choix du type de message affecte votre approche de la conversion de message. L'interaction entre la conversion de message et le type de message est décrite pour les types de message JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessageet` `JMSBytesMessage`.

Ecriture de classes pour encapsuler une présentation d'enregistrement dans un `JMSBytesMessage`
L'objectif de cette tâche est d'explorer, par exemple, comment combiner la conversion de données et une présentation d'enregistrement fixe dans un `JMSBytesMessage`. Dans la tâche, vous créez des classes Java pour échanger un exemple de structure d'enregistrement dans un `JMSBytesMessage`. Vous pouvez modifier l'exemple pour écrire des classes afin d'échanger d'autres structures d'enregistrement.

Un `JMSBytesMessage` est le meilleur choix de type de message JMS pour échanger des enregistrements de type de données mixtes avec des programmes nonJMS . Aucune donnée supplémentaire n'est insérée dans le corps du message par le fournisseur JMS . Il s'agit donc du meilleur choix de type de message à utiliser si un programme client JMS interagit avec un programme IBM MQ existant. Le défi principal de l'utilisation d'un `JMSBytesMessage` consiste à faire correspondre le codage et le jeu de caractères attendus par l'autre programme. Une solution consiste à créer une classe qui encapsule l'enregistrement. Une classe qui encapsule la lecture et l'écriture d'un `JMSBytesMessage`, pour un type d'enregistrement spécifique, facilite l'envoi et la réception d'enregistrements au format fixe dans un programme JMS . En capturant les aspects génériques de l'interface dans une classe abstraite, une grande partie de la solution peut être réutilisée pour différents formats d'enregistrement. Différents formats d'enregistrement peuvent être implémentés dans des classes qui étendent la classe générique abstraite.

Une autre approche consiste à étendre la classe `com.ibm.mq.headers.Header` . La classe `Header` comporte des méthodes, telles que `addMQLONG`, pour générer un format d'enregistrement de manière plus déclarative. L'utilisation de la classe `Header` présente l'inconvénient d'obtenir et de définir des attributs à l'aide d'une interface d'interprétation plus complexe. Les deux approches aboutissent à une quantité sensiblement identique de code d'application.

Un `JMSBytesMessage` ne peut encapsuler qu'un seul format, en plus d'un `MQRFH2`, dans un seul message, sauf si chaque enregistrement utilise le même format, le même jeu de caractères codés et le même codage. Le format, le codage et le jeu de caractères d'un `JMSBytesMessage` sont les propriétés de tous les messages qui suivent le `MQRFH2`. L'exemple est écrit en supposant qu'un `JMSBytesMessage` ne contient qu'un seul enregistrement utilisateur.

Avant de commencer

1. Votre niveau de compétence: vous devez être familiarisé avec la programmation Java et JMS. Aucune instruction n'est fournie pour la configuration de l'environnement de développement Java . Il est avantageux d'avoir écrit un programme pour échanger un `JMSTextMessage`, un `JMSStreamMessageou` un `JMSMapMessage`. Vous pouvez ensuite voir les différences dans l'échange d'un message à l'aide d'un `JMSBytesMessage`.
2. L'exemple requiert IBM WebSphere MQ 7.0.
3. L'exemple a été créé à l'aide de la perspective Java du plan de travail Eclipse . Il requiert JRE 6.0 ou version ultérieure. Vous pouvez utiliser la perspective Java dans IBM MQ Explorer pour

développer et exécuter les classes Java . Vous pouvez également utiliser votre propre environnement de développement Java .

4. L'utilisation de IBM MQ Explorer facilite la configuration de l'environnement de test et le débogage, par rapport à l'utilisation des utilitaires de ligne de commande.

Pourquoi et quand exécuter cette tâche

Vous êtes guidé lors de la création de deux classes: RECORD et MyRecord. Ensemble, ces deux classes encapsulent un enregistrement à format fixe. Ils possèdent des méthodes permettant d'obtenir et de définir des attributs. La méthode get lit l'enregistrement à partir d'un JMSBytesMessage et la méthode put écrit un enregistrement dans un JMSBytesMessage.

L'objectif de la tâche n'est pas de créer une classe de qualité de production que vous pouvez réutiliser. Vous pouvez choisir d'utiliser les exemples de la tâche pour démarrer sur vos propres classes. L'objectif de la tâche est de vous fournir des notes de conseils, principalement sur l'utilisation des jeux de caractères, des formats et du codage, lors de l'utilisation d'un JMSBytesMessage. Chaque étape de création des classes est expliquée et les aspects de l'utilisation de JMSBytesMessage, qui sont parfois négligés, sont décrits.

La classe RECORD est abstraite et définit des zones communes pour un enregistrement utilisateur. Les zones communes sont modélisées sur la présentation d'en-tête IBM MQ standard d'un identificateur, d'une version et d'une zone de longueur. Les zones de codage, de jeu de caractères et de format, qui se trouvent dans de nombreux en-têtes IBM MQ , sont omises. Un autre en-tête ne peut pas suivre un format défini par l'utilisateur. La classe MyRecord , qui étend la classe RECORD , le fait en étendant littéralement l'enregistrement avec des zones utilisateur supplémentaires. Un JMSBytesMessage, créé par les classes, peut être traité par l'exit de conversion des données du gestionnaire de files d'attente.

«Classes utilisées pour exécuter l'exemple», à la page 203 inclut une liste complète de RECORD et de MyRecord. Il inclut également les listes des classes "scaffolding" supplémentaires pour tester RECORD et MyRecord. Les classes supplémentaires sont les suivantes:

TryMyRecord
Le programme principal pour tester RECORD et MyRecord.

EndPoint
Classe abstraite qui encapsule la connexion, la destination et la session JMS dans une seule classe. Son interface répond simplement aux besoins de test des classes RECORD et MyRecord . Il ne s'agit pas d'un modèle de conception établi pour l'écriture d'applications JMS .

Remarque : La classe Endpoint inclut cette ligne de code après la création d'une destination:

```
((MQDestination)destination).setReceiveConversion
                            (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Dans V7.0, à partir de V7.0.1.5, il est nécessaire d'activer la conversion du gestionnaire de files d'attente. Elle est désactivée par défaut. Dans V7.0, la conversion du gestionnaire de files d'attente jusqu'à V7.0.1.4 est activée par défaut et cette ligne de code génère une erreur.

MyProducer et MyConsumer
Classes qui étendent EndPointet créent un MessageConsumer et un MessageProducer, connectés et prêts à accepter des demandes.

Ensemble, toutes les classes constituent une application complète que vous pouvez générer et utiliser pour comprendre comment utiliser la conversion de données dans un JMSBytesMessage.

Procédure

1. Créez une classe abstraite pour encapsuler les zones standard dans un en-tête IBM MQ , avec un constructeur par défaut. Par la suite, vous étendez la classe pour personnaliser l'en-tête en fonction de vos besoins.

```
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();
   }
```

Remarque :

a. Les attributs, `structID` à `nextFormat`, sont répertoriés dans l'ordre dans lequel ils sont présentés dans un en-tête de message IBM MQ standard.

b. Les attributs, `format`, `messageEncoding`et `messageCharset`, décrivent l'en-tête lui-même et ne font pas partie de l'en-tête.

c. Vous devez décider de stocker l'identificateur de jeu de caractères codés ou le jeu de caractères de l'enregistrement. Java utilise des jeux de caractères et les messages IBM MQ utilisent des identificateurs de jeu de caractères codés. L'exemple de code utilise des jeux de caractères.

d. `int` est sérialisé dans `MQLONG` par IBM MQ. `MQLONG` est de 4 octets.

2. Créez les méthodes d'accès get et set pour les attributs privés.

a) Créez ou générez les méthodes d'accès get:

```
    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) Créez ou générez les méthodes d'accès set:

```
    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. Créez un constructeur pour créer une instance RECORD à partir d'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());
    }
```

Remarque :

a. `messageCharset` et `messageEncoding`sont capturées à partir des propriétés de message, car elles remplacent les valeurs définies pour la destination. `format` n'est pas mis à jour. L'exemple ne vérifie pas les erreurs. Si le constructeur `Record(BytesMessage)` est appelé, il est supposé que `JMSBytesMessage` est un message de type `RECORD` . La ligne "`setStructID(new String(structID, getMessageCharset()))`" permet de définir le eye-catcher.

b. Lignes de code qui complètent les zones de désérialisation de la méthode dans le message, dans l'ordre, en mettant à jour les valeurs par défaut définies dans l'instance RECORD.

4. Créez une méthode d'insertion pour écrire les zones d'en-tête dans 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;
    }
```

Remarque :

a. `MyProducer` encapsule les JMS `Connection`, `Destination`, `Session`et `MessageProducer` dans une seule classe. MyConsumer, utilisé ultérieurement, encapsule les JMS `Connection`, `Destination`, `Session`et `MessageConsumer` dans une seule classe.

b. Pour un `JMSBytesMessage`, si le codage n'est pas `Native`, il doit être défini dans le message. Le codage de destination est copié dans l'attribut de codage de message, `JMS_IBM_CHARACTER_SET`, et sauvegardé en tant qu'attribut de la classe `RECORD` .

   i) "`setMessageEncoding(myProducer.getEncoding());`" appelle "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" pour obtenir le codage de destination.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" définit le codage des messages.

c. Le jeu de caractères utilisé pour transformer le texte en octets est obtenu à partir de la destination et sauvegardé en tant qu'attribut de la classe `RECORD` . Il n'est pas défini dans le message car il n'est pas utilisé par IBM MQ classes for JMS lors de l'écriture d'un `JMSBytesMessage`.

Appels à "`messageCharset = myProducer.getCharset();`"

```
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());
}
```

Il obtient le jeu de caractères Java à partir d'un identificateur de jeu de caractères codés.

" CCSID.getCodepage(ccsid) " se trouve dans le package com.ibm.mq.headers. ccsid est obtenu à partir d'une autre méthode dans MyProducer, qui interroge la destination:

```
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));
    }
```

d. "myProducer.setMQClient(true);" remplace le paramètre de destination pour le type de client, en le forçant à un IBM MQ MQI client. Vous pouvez préférer omettre cette ligne de code, car elle masque une erreur de configuration administrative.

Appels de "myProducer.setMQClient(true);":

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

Le code a pour effet secondaire de définir le style de corps IBM MQ sur une valeur non spécifiée, s'il doit remplacer un paramètre JMS.

Remarque :

Les IBM MQ classes for JMS écrivent le format, le codage et l'identificateur de jeu de caractères du message dans le descripteur de message, MQMD, ou dans l'en-tête JMS , MQRFH2. Cela varie selon que le message comporte ou non un corps de style IBM MQ . Ne définissez pas les zones MQMD manuellement.

Il existe une méthode permettant de définir manuellement les propriétés du descripteur de message. Il utilise les propriétés JMS_IBM_MQMD_* . Vous devez définir la propriété de destination WMQ_MQMD_WRITE_ENABLED pour définir les propriétés JMS_IBM_MQMD_* :

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Vous devez définir la propriété de destination, WMQ_MQMD_READ_ENABLED, pour lire les propriétés.

Utilisez le JMS_IBM_MQMD_* uniquement si vous prenez le contrôle total de la totalité de la charge de message. Contrairement aux propriétés JMS_IBM_* , les propriétés JMS_IBM_MQMD_* ne contrôlent pas la façon dont IBM MQ classes for JMS construit un message JMS . Il est possible de créer des propriétés de descripteur de message en conflit avec les propriétés du message JMS .

e. Les lignes de code qui complètent la méthode sérialisent les attributs de la classe en tant que zones du message.

Les attributs de chaîne sont complétés par des blancs. Les chaînes sont converties en octets à l'aide du jeu de caractères défini pour l'enregistrement et tronquées à la longueur des zones de message.

5. Terminez la classe en ajoutant les importations.

```
 package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. Créez une classe pour étendre la classe RECORD afin d'inclure des zones supplémentaires. Incluez un constructeur par défaut.

```
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);
   }
```

Remarque :

a. La sous-classe RECORD , MyRecord, personnalise l'identificateur, le format et la longueur de l'en-tête.

7. Créez ou générez les méthodes d'accès get et set.

a) Créez les méthodes d'accès get:

```
    public int getFlags() { return flags; }
   public String getRecordData() { return recordData; } .
```

b) Créez les méthodes d'accès set:

```
    public void setFlags(int flags) {
       this.flags = flags; }
   public void setRecordData(String recordData) {
       this.recordData = recordData; }
}
```

8. Créez un constructeur pour créer une instance MyRecord à partir d'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()));
   }
```

Remarque :

a. Les zones qui constituent le modèle de message standard sont lues en premier par la classe RECORD .

b. Le texte recordData est converti en String à l'aide de la propriété de jeu de caractères du message.

9. Créez une méthode statique pour obtenir un message d'un consommateur et créer une nouvelle instance MyRecord .

```
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
           MQDataException, IOException {
       BytesMessage message = (BytesMessage) myConsumer.receive();
       return new MyRecord(message);
   }
```

Remarque :

a. Dans l'exemple, par souci de concision, le constructeur `MyRecord(BytesMessage)` est appelé à partir de la méthode get statique. En règle générale, vous pouvez séparer la réception du message de la création d'une nouvelle instance `MyRecord` .

10. Créez une méthode d'insertion pour ajouter les zones client à un `JMSBytesMessage` contenant un en-tête de message.

```
 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;
}
```

Remarque :

a. Les appels de méthode dans le code sérialisent les attributs de la classe MyRecord en tant que zones dans le message.

- L'attribut `recordData` `String` est rempli avec des blancs, converti en octets à l'aide du jeu de caractères défini pour l'enregistrement et tronqué à la longueur des zones `RecordData` .

11. Terminez la classe en ajoutant les instructions include.

```
 package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Résultats

- Résultats de l'exécution de la classe `TryMyRecord` :
  - Envoi d'un message dans le jeu de caractères codés 37 et utilisation d'un exit de conversion de gestionnaire de files d'attente:

```
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
```

  - Envoi d'un message dans le jeu de caractères codés 37 et non utilisation d'un exit de conversion de gestionnaire de files d'attente:

```
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
```

- Les résultats de la modification de la classe `TryMyRecord` pour ne pas recevoir le message et de sa réception à l'aide de l'exemple `amqsget0.c` modifié. L'exemple modifié accepte un enregistrement formaté ; voir Figure 38, à la page 195 dans «Echange d'un enregistrement formaté avec une application nonJMS», à la page 192.
  - Envoi d'un message dans le jeu de caractères codés 37 et utilisation d'un exit de conversion de gestionnaire de files d'attente:

```
Sample AMQSGET0 start
ccsid <850>, flags <1>, message <ABCDEFGHIJKLMNOPQRSTUVWXYZ012345>
```

```
no more messages
Sample AMQSGET0 end
```

– Envoi d'un message dans le jeu de caractères codés 37 et non utilisation d'un exit de conversion de gestionnaire de files d'attente:

```
Sample AMQSGET0 start
MQGET ended with reason code 2110
ccsid <37>, flags <1>, message <--+-+ãÃ++ÐÊËÈiÐÎÐ+ÔÒõõµþÞÚ-±=¾¶§>
no more messages
Sample AMQSGET0 end
```

Pour tester l'exemple et expérimenter avec différentes pages de codes et un exit de conversion de données. Créez les classes Java , configurez IBM MQet exécutez le programme principal, `TryMyRecord` ; voir «#unique_196/unique_196_Connect_42_Try», à la page 203.

1. Configurez IBM MQ et JMS pour exécuter l'exemple. Les instructions permettent d'exécuter l'exemple sous Windows.
   a. Création d'un gestionnaire de files d'attente

   ```
   crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
   strmqm QM1
   ```

   b. Création d'une file d'attente

   ```
   echo DEFINE QL('Q1') REPLACE | runmqsc QM1
   ```

   c. Créer un répertoire JNDI

   ```
   cd c:\
   md JNDI-Directory
   ```

   d. Passez dans le répertoire bin JMS

   Le programme d'administration JMS doit être exécuté à partir d'ici. Le chemin d'accès est *MQ_INSTALLATION_PATH*`\java\bin`.

   e. Créez les définitions JMS suivantes dans un fichier appelé `JMSQM1Q1.txt`

   ```
   DEF CF(QM1) PROVIDERVERSION(7) QMANAGER(QM1)
   DEF Q(Q1) CCSID(37) ENCODING(RRR) MSGBODY(MQ) QMANAGER(QM1) QUEUE(Q1) TARGCLIENT(MQ)
   VERSION(7)
   END
   ```

   f. Exécutez le programme JMSAdmin pour créer les ressources JMS

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. Vous pouvez créer, modifier et parcourir les définitions que vous avez créées à l'aide de l'explorateur IBM MQ .
3. Exécutez `TryMyRecord`.

Classes utilisées pour exécuter l'exemple

Les classes répertoriées dans les blocs de code suivants sont également disponibles dans un fichier compressé. Téléchargez jm25529_.zip ou jm25529_.tar.gz.

`TryMyRecord`

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
```

```
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

RECORD

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
```

```
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }
    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }

    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
```

```
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

MyRecord

JM 3.0

```
package com.ibm.mq.id;
import java.io.IOException;
import jakarta.jms.BytesMessage;
import jakarta.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                         + DATA_LENGTH + "s",getRecordData())
                         .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
```

```
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }
    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }
    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }
    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; } .

    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }
    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

EndPoint

JM 3.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import jakarta.jms.Connection;
import jakarta.jms.ConnectionFactory;
import jakarta.jms.Destination;
import jakarta.jms.JMSException;
import jakarta.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
```

```
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

JMS 2.0

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
```

```
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                   == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
        }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

MyProducer

JM 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
```

```
import jakarta.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

MyConsumer

JM 3.0

```
package com.ibm.mq.id;
import jakarta.jms.JMSException;
import jakarta.jms.Message;
import jakarta.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

JMS 2.0

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
```

```
        return consumer.receive();  }
}
```

Création et configuration de fabriques de connexions et de destinations

Une application IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging peut créer des fabriques de connexions et des destinations en les extrayant en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), à l'aide des extensions IBM JMS ou à l'aide des extensions IBM MQ JMS . Une application peut également utiliser les extensions IBM JMS ou IBM MQ JMS pour définir les propriétés des fabriques de connexions et des destinations.

Les fabriques de connexions et les destinations sont des points de départ dans le flux logique d'une application JMS ou Jakarta Messaging . Une application utilise un objet ConnectionFactory pour créer une connexion à un serveur de messagerie et utilise un objet Queue ou Topic comme cible pour l'envoi de messages ou comme source de réception de messages. Une application doit donc créer au moins une fabrique de connexions et une ou plusieurs destinations. Après avoir créé une fabrique de connexions ou une destination, l'application peut avoir besoin de configurer l'objet en définissant une ou plusieurs de ses propriétés.

En résumé, une application peut créer et configurer des fabriques de connexions et des destinations comme suit:

Utilisation de JNDI pour extraire des objets gérés

Un administrateur peut utiliser l'outil d'administration IBM MQ JMS comme décrit dans Configuration des objets JMS et Jakarta Messaging à l'aide des outils d'administration, ou IBM MQ Explorer comme décrit dans Configuration des objets JMS 2.0 à l'aide d' IBM MQ Explorer, pour créer et configurer des fabriques de connexions et des destinations en tant qu'objets gérés dans un espace de nom JNDI. Une application peut ensuite extraire les objets gérés de l'espace de nom JNDI. Après avoir extrait un objet géré, l'application peut, si nécessaire, définir ou modifier une ou plusieurs de ses propriétés à l'aide des extensions IBM JMS ou IBM MQ JMS .

Remarque : JM 3.0 Pour Jakarta Messaging 3.0, vous ne pouvez pas administrer JNDI à l'aide de IBM MQ Explorer. L'administration JNDI est prise en charge par la variante Jakarta Messaging 3.0 de **JMSAdmin**, qui est **JMS30Admin**.

Utilisation des extensions IBM JMS

Une application peut utiliser les extensions IBM JMS pour créer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution. L'application crée d'abord un objet de fabrique JmsFactory, puis utilise les méthodes de cet objet pour créer des fabriques de connexions et des destinations. Après avoir créé une fabrique de connexions ou une destination, l'application peut utiliser des méthodes héritées de l'interface de contexte JmsPropertypour définir ses propriétés. L'application peut également utiliser un URI (Uniform Resource Identifier) pour spécifier une ou plusieurs propriétés d'une destination lorsqu'elle crée la destination.

Utilisation des extensions IBM MQ JMS

Une application peut également utiliser les extensions IBM MQ JMS pour créer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution. L'application utilise les constructeurs fournis pour créer des fabriques de connexions et des destinations. Après avoir créé une fabrique de connexions ou une destination, l'application peut utiliser les méthodes de l'objet pour définir ses propriétés. L'application peut également utiliser un URI pour spécifier une ou plusieurs propriétés d'une destination lorsqu'elle crée la destination.

Tâches associées

Configuration des ressources JMS et Jakarta Messaging

Utilisation de JNDI pour extraire des objets gérés dans une application JMS ou Jakarta Messaging

Pour extraire des objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), une application JMS ou Jakarta Messaging doit créer un contexte initial, puis utiliser la méthode lookup () pour extraire les objets.

Avant qu'une application puisse récupérer les objets gérés d'un espace de nom JNDI, un administrateur doit d'abord créer les objets gérés.

JMS 2.0 Pour JMS 2.0, l'administrateur peut utiliser l'outil d'administration IBM MQ JMS , **JMSAmin**ou IBM MQ Explorer pour créer et gérer des objets gérés dans un espace de nom JNDI. Pour plus d'informations, voir Configuration de fabriques de connexions et de destinations dans un espace de nom JNDI.

JM 3.0 Pour Jakarta Messaging 3.0, vous ne pouvez pas administrer JNDI à l'aide de IBM MQ Explorer. L'administration JNDI est prise en charge par la variante Jakarta Messaging 3.0 de **JMSAdmin**, qui est **JMS30Admin**.

Un serveur d'applications fournit généralement son propre référentiel pour les objets gérés et ses propres outils pour la création et la gestion des objets.

Pour extraire des objets gérés d'un espace de nom JNDI, une application doit d'abord créer un contexte initial, comme illustré dans l'exemple suivant:

JM 3.0
```
import jakarta.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

JMS 2.0
```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
.
.
.
String url = "ldap://server.company.com/o=company_us,c=us";
String icf = "com.sun.jndi.ldap.LdapCtxFactory";
.
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.PROVIDER_URL, url);
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
Context ctx = new InitialDirContext(environment);
```

Dans ce code, les variables de chaîne `url` et `icf` ont les significations suivantes:

url
Adresse URL du service d'annuaire. L'URL peut avoir l'un des formats suivants:

- `ldap://`*`hostname`*`/`*`contextName`* , pour un service d'annuaire basé sur un serveur LDAP
- `file:/`*`directoryPath`* , pour un service d'annuaire basé sur le système de fichiers local

ICF-OS/400
Nom de classe de la fabrique de contexte initial, qui peut être l'une des valeurs suivantes:

- `com.sun.jndi.ldap.LdapCtxFactory`, pour un service d'annuaire basé sur un serveur LDAP
- `com.sun.jndi.fscontext.RefFSContextFactory`, pour un service d'annuaire basé sur le système de fichiers local

Notez que certaines combinaisons d'un package JNDI et d'un fournisseur de services LDAP (Lightweight Directory Access Protocol) peuvent provoquer l'erreur LDAP 84. Pour résoudre ce problème, insérez la ligne de code suivante avant l'appel à InitialDirContext ():

```
environment.put(Context.REFERRAL, "throw");
```

Une fois qu'un contexte initial est obtenu, l'application peut extraire des objets gérés de l'espace de nom JNDI à l'aide de la méthode lookup (), comme illustré dans l'exemple suivant:

```
ConnectionFactory factory;
Queue queue;
Topic topic;
.
.
.
factory = (ConnectionFactory)ctx.lookup("cn=myCF");
queue = (Queue)ctx.lookup("cn=myQ");
topic = (Topic)ctx.lookup("cn=myT");
```

Ce code extrait les objets suivants d'un espace-noms LDAP:

- Un objet ConnectionFactory lié avec le nom myCF
- Un objet Queue lié avec le nom myQ
- Un objet Topic lié avec le nom myT

Pour plus d'informations sur l'utilisation de JNDI, voir la documentation JNDI fournie par Oracle Corporation.

Tâches associées

Configuration d'objets JMS 2.0 à l'aide d' IBM MQ Explorer

Configuration d'objets JMS et Jakarta Messaging à l'aide des outils d'administration

Configuration des ressources JMS 2.0 dans WebSphere Application Server

Utilisation des extensions IBM JMS

IBM MQ classes for JMS (JMS 2.0) et IBM MQ classes for Jakarta Messaging (Jakarta Messaging 3.0) contiennent chacun un ensemble d'extensions identiques sur le plan fonctionnel à l'API JMS appelée extensions IBM JMS . Une application peut utiliser ces extensions pour créer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution et pour définir les propriétés des objets IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging . Les extensions peuvent être utilisées avec n'importe quel fournisseur de messagerie.

Les extensions IBM JMS sont un ensemble d'interfaces et de classes dans les packages suivants:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Pour Jakarta Messaging 3.0, ces packages se trouvent dans `com.ibm.jakarta.client.jar`.

JMS 2.0 Pour JMS 2.0, ces packages se trouvent dans `com.ibm.mqjms.jar` ou `com.ibm.mq.allclient.jar`.

Ces extensions fournissent la fonction suivante:

- Mécanisme basé sur des fabriques permettant de créer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution, au lieu de les extraire en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface)
- Ensemble de méthodes permettant de définir les propriétés des objets IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging
- Ensemble de classes d'exception avec des méthodes permettant d'obtenir des informations détaillées sur un problème
- Ensemble de méthodes de contrôle de la fonction de trace
- Ensemble de méthodes permettant d'obtenir des informations de version sur IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

Pour la création dynamique de fabriques de connexions et de destinations lors de l'exécution, ainsi que la définition et l'obtention de leurs propriétés, les extensions IBM JMS fournissent un autre ensemble d'interfaces aux extensions IBM MQ JMS . Toutefois, alors que les extensions IBM MQ JMS sont spécifiques au fournisseur de messagerie IBM MQ , les extensions IBM JMS ne sont pas spécifiques à IBM

MQ et peuvent être utilisées avec n'importe quel fournisseur de messagerie au sein de l'architecture en couches décrite dans IBM MQ classes for JMS architecture.

L'interface com.ibm.msg.client.wmq.WMQConstants (JMS 2.0) ou com.ibm.msg.jakarta.client.wmq.WMQConstants (Jakarta Messaging 3.0) contient les définitions des constantes qu'une application peut utiliser lors de la définition des propriétés des objets IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging à l'aide des extensions IBM JMS . L'interface contient des constantes pour le fournisseur de messagerie IBM MQ et des constantes JMS qui sont indépendantes de tout fournisseur de messagerie.

Les exemples de code qui suivent supposent que les instructions d'importation suivantes sont incluses dans la classe Java :

JM 3.0
```
import com.ibm.msg.jakarta.client.jms.*;
import com.ibm.msg.jakarta.client.services.*;
import com.ibm.msg.jakarta.client.wmq.WMQConstants;
```

JMS 2.0
```
import com.ibm.msg.client.jms.*;
import com.ibm.msg.client.services.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Création de fabriques de connexions et de destinations

Pour qu'une application puisse créer des fabriques de connexions et des destinations à l'aide des extensions IBM JMS , elle doit d'abord créer un objet de fabrique JmsFactory. Pour créer un objet de fabrique JmsFactory, l'application appelle la méthode getInstance() de la classe de fabrique JmsFactory, comme illustré dans l'exemple suivant:

JM 3.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.JAKARTA_WMQ_PROVIDER);
```

JMS 2.0
```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Le paramètre de l'appel getInstance() est une constante qui identifie le fournisseur de messagerie IBM MQ comme le fournisseur de messagerie choisi. L'application peut ensuite utiliser l'objet JmsFactoryFactory pour créer des fabriques de connexions et des destinations.

Pour créer une fabrique de connexions, l'application appelle la méthode createConnectionFactory () de l'objet de fabrique JmsFactory, comme illustré dans l'exemple suivant:

```
JmsConnectionFactory factory = ff.createConnectionFactory();
```

Cette instruction crée un objet de fabrique JmsConnectionavec les valeurs par défaut pour toutes ses propriétés, ce qui signifie que l'application se connecte au gestionnaire de files d'attente par défaut en mode liaisons. Si vous souhaitez qu'une application se connecte en mode client ou à un gestionnaire de files d'attente autre que le gestionnaire de files d'attente par défaut, l'application doit définir les propriétés appropriées de l'objet JmsConnectionFactory avant de créer la connexion. Pour savoir comment procéder, voir «Définition des propriétés des objets IBM MQ classes for JMS», à la page 215.

La classe de fabrique JmsFactorycontient également des méthodes permettant de créer des fabriques de connexions des types suivants:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- Fabrique JmsXAConnection
- JmsXAQueueConnectionFactory

- JmsXATopicConnectionFactory

Pour créer un objet Queue, l'application appelle la méthode createQueue() de l'objet Factory JmsFactory, comme illustré dans l'exemple suivant:

```
JmsQueue q1 = ff.createQueue("Q1");
```

Cette instruction crée un objet JmsQueue avec les valeurs par défaut pour toutes ses propriétés. L'objet représente une file d'attente IBM MQ appelée Q1 qui appartient au gestionnaire de files d'attente local. Cette file d'attente peut être une file d'attente locale, une file d'attente alias ou une définition de file d'attente éloignée.

La méthode createQueue() peut également accepter un identificateur URI (uniform resource identifier) de file d'attente comme paramètre. Un URI de file d'attente est une chaîne qui spécifie le nom d'une file d'attente IBM MQ et, en option, le nom du gestionnaire de files d'attente propriétaire de la file d'attente, ainsi qu'une ou plusieurs propriétés de l'objet JmsQueue . L'instruction suivante contient un exemple d'URI de file d'attente:

```
JmsQueue q2 = ff.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

L'objet JmsQueue créé par cette instruction représente une file d'attente IBM MQ appelée Q2 qui appartient au gestionnaire de files d'attente QM2, et tous les messages envoyés à cette destination sont persistants et ont une priorité de 5. Pour plus d'informations sur les URI de file d'attente, voir «uniform resource identifier (URI)», à la page 229. Pour une autre méthode de définition des propriétés d'un objet JmsQueue , voir «Définition des propriétés des objets IBM MQ classes for JMS», à la page 215.

Pour créer un objet Topic, une application peut utiliser la méthode createTopic() de l'objet Factory JmsFactory, comme illustré dans l'exemple suivant:

```
JmsTopic t1 = ff.createTopic("Sport/Football/Results");
```

Cette instruction crée un objet JmsTopic avec les valeurs par défaut pour toutes ses propriétés. L'objet représente un sujet appelé Sport / Football/Résultats.

La méthode createTopic() peut également accepter un URI de rubrique comme paramètre. Un URI de rubrique est une chaîne qui spécifie le nom d'une rubrique et, en option, une ou plusieurs propriétés de l'objet JmsTopic . Les instructions suivantes contiennent un exemple d'URI de rubrique:

```
String s1 = "topic://Sport/Tennis/Results?persistence=1&priority=0";
JmsTopic t2 = ff.createTopic(s1);
```

L'objet JmsTopic créé par ces instructions représente une rubrique appelée Sport / Tennis/Results, et tous les messages envoyés à cette destination sont non persistants et ont une priorité de 0. Pour plus d'informations sur les URI de rubrique, voir «uniform resource identifier (URI)», à la page 229. Pour une autre méthode de définition des propriétés d'un objet JmsTopic , voir «Définition des propriétés des objets IBM MQ classes for JMS», à la page 215.

Une fois qu'une application a créé une fabrique de connexions ou une destination, cet objet ne peut être utilisé qu'avec le fournisseur de messagerie sélectionné.

Définition des propriétés des objets IBM MQ classes for JMS

Pour définir les propriétés des objets IBM MQ classes for JMS à l'aide des extensions IBM JMS , une application utilise les méthodes de l'interface com.ibm.msg.client.JmsPropertyContext . De même, pour définir les propriétés des objets IBM MQ classes for Jakarta Messaging à l'aide des extensions IBM JMS , une application utilise les méthodes de l'interface com.ibm.msg.jakarta.client.JmsPropertyContext .

Pour chaque type de données Java , l'interface de contexte JmsPropertycontient une méthode permettant de définir la valeur d'une propriété avec ce type de données et une méthode permettant d'obtenir la valeur d'une propriété avec ce type de données. Par exemple, une application appelle la

méthode setIntProperty () pour définir une propriété avec une valeur entière et appelle la méthode getIntProperty () pour obtenir une propriété avec une valeur entière.

Les instances des classes dans les packages com.ibm.mq.jms et com.ibm.mq.jakarta.jms héritent des méthodes des interfaces de contexte JmsPropertycorrespondantes. Une application peut donc utiliser ces méthodes pour définir les propriétés des objets MQConnectionFactory, MQQueue et MQTopic.

Lorsqu'une application crée un objet IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , toutes les propriétés avec des valeurs par défaut sont définies automatiquement. Lorsqu'une application définit une propriété, la nouvelle valeur remplace toute valeur précédente de la propriété. Une fois qu'une propriété a été définie, elle ne peut pas être supprimée, mais sa valeur peut être modifiée.

Si une application tente de définir une propriété sur une valeur qui n'est pas une valeur valide pour la propriété, IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging émet une exception JMSException. Si une application tente d'obtenir une propriété qui n'a pas été définie, le comportement est décrit dans la spécification JMS . IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging émettent une exception NumberFormatpour les types de données primitives et renvoient la valeur null pour les types de données référencés.

Outre les propriétés prédéfinies d'un objet IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , une application peut définir ses propres propriétés. Ces propriétés définies par l'application sont ignorées par IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging.

Pour plus d'informations sur les propriétés des objets IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging , voir Propriétés des objets IBM MQ classes for JMS.

Le code suivant est un exemple de définition des propriétés à l'aide des extensions IBM JMS . Le code définit cinq propriétés d'une fabrique de connexions.

```
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");
```

La définition de ces propriétés a pour effet que l'application se connecte au gestionnaire de files d'attente QM1 en mode client, à l'aide d'un canal MQI appelé QM1.SVR. Le gestionnaire de files d'attente s'exécute sur un système dont le nom d'hôte est HOST1et le programme d'écoute du gestionnaire de files d'attente est en mode écoute sur le port numéro 1415. Cette connexion et les autres connexions de gestionnaire de files d'attente associées aux sessions qui la contiennent sont associées au nom d'application "Mon application".

Remarque : Les gestionnaires de files d'attente s'exécutant sur des plateformes z/OS ne prennent pas en charge la définition des noms d'application et ce paramètre est donc ignoré.

L'interface de contexte JmsPropertycontient également la méthode setObjectProperty (), qu'une application peut utiliser pour définir des propriétés. Le second paramètre de la méthode est un objet qui encapsule la valeur de la propriété. Par exemple, le code suivant crée un objet Integer qui encapsule l'entier 1415, puis appelle la propriété setObjectProperty () pour définir la propriété PORT d'une fabrique de connexions sur la valeur 1415:

```
Integer port = new Integer(1415);
factory.setObjectProperty(WMQConstants.WMQ_PORT, port);
```

Ce code est donc équivalent à l'instruction suivante:

```
factory.setIntProperty(WMQConstants.WMQ_PORT, 1415);
```

A l'inverse, la méthode getObjectProperty () renvoie un objet qui encapsule la valeur d'une propriété.

Conversion implicite d'une valeur de propriété d'un type de données à un autre

Lorsqu'une application utilise une méthode de l'interface de contexte JmsPropertypour définir ou obtenir la propriété d'un objet IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , la valeur de la propriété peut être implicitement convertie d'un type de données à un autre.

Par exemple, l'instruction suivante définit la propriété PRIORITY de l'objet JmsQueue q1:

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

La propriété PRIORITY ayant une valeur entière, l'appel setStringProperty () convertit implicitement la chaîne "5" (valeur source) en entier 5 (valeur cible), qui devient alors la valeur de la propriété PRIORITY.

A l'inverse, l'instruction suivante extrait la propriété PRIORITY de l'objet JmsQueue q1:

```
String s1 = q1.getStringProperty(WMQConstants.WMQ_PRIORITY);
```

L'entier 5 (valeur source), qui est la valeur de la propriété PRIORITY, est implicitement converti en chaîne "5" (valeur cible) par l'appel de la propriété getString().

Les conversions prises en charge par IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont présentées dans Tableau 34, à la page 217.

Tableau 34. Conversions prises en charge d'un type de données à un autre

Type de données source	**Types de données cible pris en charge**
boolean	String
byte	int, long, court, Chaîne
char	String
double	String
float	double, chaîne
int	long, chaîne
long	String
short	int, long, Chaîne
String	boolean, byte, double, float, int, long, short

Les règles générales régissant les conversions prises en charge sont les suivantes:

- Les valeurs numériques peuvent être converties d'un type de données à un autre, à condition qu'aucune donnée ne soit perdue lors de la conversion. Par exemple, une valeur avec le type de données `int` peut être convertie en valeur avec le type de données `long`, mais ne peut pas être convertie en valeur avec le type de données `short`.
- Une valeur de n'importe quel type de données peut être convertie en chaîne.
- Une chaîne peut être convertie en une valeur de tout autre type de données (sauf `char`) à condition que la chaîne soit au format correct pour la conversion. Si une application tente de convertir une chaîne dont le format n'est pas correct, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging émettent une exception NumberFormat.
- Si une application tente une conversion qui n'est pas prise en charge, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging émettent une exception MessageFormat.

Les règles spécifiques de conversion d'une valeur d'un type de données à un autre sont les suivantes:

- Lors de la conversion d'une valeur booléenne en chaîne, la valeur `true` est convertie en chaîne "true"et la valeur `false` est convertie en chaîne "false".

- Lors de la conversion d'une chaîne en valeur booléenne, la chaîne "true" (non sensible à la casse) est convertie en `true`et la chaîne "false" (non sensible à la casse) est convertie en `false`. Toute autre chaîne est convertie en `false`.
- Lors de la conversion d'une chaîne en valeur avec le type de données `byte`, `int`, `long`ou `short`, la chaîne doit avoir le format suivant:

  [*blancs*] [*signe*] *chiffres*

  Les significations des composants de la chaîne sont les suivantes:

  blancs
  Caractères blancs facultatifs de début.

  signe
  Signe plus (+) ou signe moins (-) facultatif.

  chiffres
  Séquence contiguë de chiffres (0-9). Au moins un chiffre doit être présent.

  Après la séquence de chiffres, la chaîne peut contenir d'autres caractères qui ne sont pas des chiffres, mais la conversion s'arrête dès que le premier de ces caractères est atteint. La chaîne est supposée représenter un entier décimal.

  Si la chaîne n'est pas au format correct, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging émettent une exception NumberFormat.
- Lors de la conversion d'une chaîne en une valeur avec le type de données `double` ou `float`, la chaîne doit avoir le format suivant:

  [*blancs*] [*signe*] *chiffres* [*e_char* [*e_signe*] *e_digits*]

  Les significations des composants de la chaîne sont les suivantes:

  blancs
  Caractères blancs facultatifs de début.

  signe
  Signe plus (+) ou signe moins (-) facultatif.

  chiffres
  Séquence contiguë de chiffres (0-9). Au moins un chiffre doit être présent.

  e_car
  Caractère exposant, qui peut être *E* ou *e*.

  e_signe
  Signe plus (+) ou signe moins (-) facultatif pour l'exposant.

  e_chiffres
  Séquence contiguë de chiffres (0-9) pour l'exposant. Au moins un chiffre doit être présent si la chaîne contient un caractère d'exposant.

  Après la séquence de chiffres ou les caractères facultatifs représentant un exposant, la chaîne peut contenir d'autres caractères qui ne sont pas des chiffres, mais la conversion s'arrête dès que le premier de ces caractères est atteint. La chaîne est supposée représenter un nombre décimal en virgule flottante avec un exposant puissance de 10.

  Si la chaîne n'est pas au format correct, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging émettent une exception NumberFormat.
- Lors de la conversion d'une valeur numérique (y compris une valeur avec le type de données `byte`) dans une chaîne, la valeur est convertie en représentation de chaîne de la valeur sous la forme d'un nombre décimal, et non la chaîne contenant le caractère ASCII pour cette valeur. Par exemple, l'entier 65 est converti en chaîne "65", et non en chaîne "A".

Définition de plusieurs propriétés dans un même appel

L'interface de contexte JmsPropertycontient également la méthode setBatchProperties (), qu'une application peut utiliser pour définir plusieurs propriétés dans un même appel. Le paramètre de la méthode est un objet Map qui encapsule un ensemble de paires nom-valeur de propriété.

Par exemple, le code suivant utilise la méthode setBatchProperties () pour définir les cinq mêmes propriétés d'une fabrique de connexions, comme illustré dans la «Définition des propriétés des objets IBM MQ classes for JMS», à la page 215. Le code crée une instance de la classe HashMap , qui implémente l'interface 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);
```

Notez que le second paramètre de la méthode Map.put() doit être un objet. Par conséquent, une valeur de propriété avec un type de données primitif doit être encapsulée dans un objet ou représentée par une chaîne, comme illustré dans l'exemple.

La méthode setBatchProperties () valide chaque propriété. Si la méthode setBatchProperties () ne peut pas définir une propriété car, par exemple, sa valeur n'est pas valide, aucune des propriétés spécifiées n'est définie.

Noms et valeurs de propriété

Si une application utilise les méthodes de l'interface de contexte JmsPropertyappropriée pour définir et obtenir les propriétés des objets IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , elle peut spécifier les noms et les valeurs des propriétés de l'une des manières suivantes. Chacun des exemples qui l'accompagnent montre comment définir la propriété PRIORITY de l'objet JmsQueue q1 de sorte qu'un message envoyé à la file d'attente ait la priorité spécifiée dans l'appel send ().

Utilisation des noms et des valeurs de propriété définis en tant que constantes dans l'interface com.ibm.msg.client.wmq.WMQConstants

L'instruction suivante est un exemple de spécification des noms et des valeurs des propriétés de cette manière:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Utilisation des noms et des valeurs de propriété pouvant être utilisés dans les identificateurs URI (Uniform Resource Identifier) de file d'attente et de rubrique

L'instruction suivante est un exemple de spécification des noms et des valeurs des propriétés de cette manière:

```
q1.setIntProperty("priority", -2);
```

Seuls les noms et les valeurs des propriétés des destinations peuvent être spécifiés de cette manière.

Utilisation des noms et des valeurs de propriété reconnus par l'outil d'administration de IBM MQ JMS

L'instruction suivante est un exemple de spécification des noms et des valeurs des propriétés de cette manière:

```
q1.setStringProperty("PRIORITY", "APP");
```

La forme abrégée du nom de propriété est également acceptable, comme indiqué dans l'instruction suivante:

```
q1.setStringProperty("PRI", "APP");
```

Lorsqu'une application obtient une propriété, la valeur renvoyée dépend de la manière dont l'application spécifie le nom de la propriété. Par exemple, si une application spécifie la constante WMQConstants.WMQ_PRIORITY comme nom de propriété, la valeur renvoyée est l'entier -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

La même valeur est renvoyée si l'application spécifie la chaîne "priority" comme nom de propriété:

```
int n2 = getIntProperty("priority");
```

Toutefois, si l'application spécifie la chaîne "PRIORITY" ou "PRI" comme nom de propriété, la valeur renvoyée est la chaîne "APP":

```
String s1 = getStringProperty("PRI");
```

En interne, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging stockent les noms et les valeurs des propriétés en tant que valeurs littérales définies dans l'interface WMQConstants correspondante. Il s'agit du format canonique défini pour les noms et les valeurs de propriété. En règle générale, si une application définit des propriétés à l'aide de l'une des deux autres méthodes de spécification des noms et des valeurs de propriété, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging doivent convertir les noms et les valeurs du format d'entrée spécifié au format canonique. De même, si une application obtient des propriétés à l'aide de l'une des deux autres méthodes de spécification des noms et des valeurs de propriété, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging doivent convertir les noms du format d'entrée spécifié dans le format canonique et convertir les valeurs du format canonique dans le format de sortie requis. L'exécution de ces conversions peut avoir des conséquences sur les performances.

Les noms de propriété et les valeurs renvoyés par les exceptions, dans les fichiers de trace ou dans le journal IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging , sont toujours au format canonique.

Utilisation de l'interface Map

L'interface de contexte JmsPropertyétend l'interface java.util.Map . Une application peut donc utiliser les méthodes de l'interface Carte pour accéder aux propriétés d'un objet IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging .

Par exemple, le code suivant imprime les noms et les valeurs de toutes les propriétés d'une fabrique de connexions. Le code utilise uniquement les méthodes de l'interface Map pour obtenir les noms et les valeurs des propriétés.

```
// 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));
}
```

L'utilisation des méthodes de l'interface de mappe ne permet pas de contourner les validations de propriété ou les conversions.

Utilisation des extensions IBM MQ JMS
IBM MQ classes for JMS contient un ensemble d'extensions de l'API JMS appelées extensions IBM MQ JMS . Une application peut utiliser ces extensions pour créer des fabriques de connexions et des

destinations de manière dynamique lors de l'exécution et pour définir les propriétés des fabriques de connexions et des destinations.

IBM MQ classes for JMS contient un ensemble de classes dans les packages com.ibm.jms et com.ibm.mq.jms. Ces classes implémentent les interfaces JMS et contiennent les extensions IBM MQ JMS . Les exemples de code qui suivent supposent que ces packages ont été importés par les instructions suivantes:

```
import com.ibm.jms.*;
import com.ibm.mq.jms.*;
import com.ibm.msg.client.wmq.WMQConstants;
```

Une application peut utiliser les extensions IBM MQ JMS pour exécuter les fonctions suivantes:

- Créer des fabriques de connexions et des destinations de manière dynamique lors de l'exécution, au lieu de les extraire en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface)
- Définir les propriétés des fabriques de connexions et des destinations

Création de fabriques de connexions

Pour créer une fabrique de connexions, une application peut utiliser le constructeur MQConnectionFactory , comme illustré dans l'exemple suivant:

```
MQConnectionFactory factory = new MQConnectionFactory();
```

Cette instruction crée un objet MQConnectionFactory avec les valeurs par défaut pour toutes ses propriétés, ce qui signifie que l'application se connecte au gestionnaire de files d'attente par défaut en mode liaisons. Si vous souhaitez qu'une application se connecte en mode client ou à un gestionnaire de files d'attente autre que le gestionnaire de files d'attente par défaut, l'application doit définir les propriétés appropriées de l'objet MQConnectionFactory avant de créer la connexion. Pour savoir comment procéder, voir «Définition des propriétés des fabriques de connexions», à la page 221.

Une application peut créer des fabriques de connexions des types suivants de la même manière:

- Fabrique MQQueueConnection
- Fabrique MQTopicConnection
- MQXAConnectionFactory
- Fabrique MQXAQueueConnection
- Fabrique MQXATopicConnection

Définition des propriétés des fabriques de connexions

Une application peut définir les propriétés d'une fabrique de connexions en appelant les méthodes appropriées de la fabrique de connexions. La fabrique de connexions peut être un objet géré ou un objet créé dynamiquement lors de l'exécution.

Prenez le code suivant, par exemple:

```
MQConnectionFactory factory = new MQConnectionFactory();
.
factory.setTransportType(WMQConstants.WMQ_CM_CLIENT);
factory.setQueueManager("QM1");
factory.setHostName("HOST1");
factory.setPort(1415);
factory.setChannel("QM1.SVR");
```

Ce code crée un objet MQConnectionFactory , puis définit cinq propriétés de l'objet. La définition de ces propriétés a pour effet que l'application se connecte au gestionnaire de files d'attente QM1 en mode client à l'aide d'un canal MQI appelé QM1.SVR. Le gestionnaire de files d'attente s'exécute sur un système

dont le nom d'hôte est HOST1et le programme d'écoute du gestionnaire de files d'attente est en mode écoute sur le port numéro 1415.

Une application qui utilise une connexion en temps réel à un courtier peut utiliser uniquement le style de messagerie de publication / abonnement. Il ne peut pas utiliser le style de messagerie point-à-point.

Seules certaines combinaisons de propriétés d'une fabrique de connexions sont valides. Pour plus d'informations sur les combinaisons valides, voir Dépendances entre les propriétés des objets IBM MQ classes for JMS.

Pour plus d'informations sur les propriétés d'une fabrique de connexions et sur les méthodes utilisées pour définir ses propriétés, voir Propriétés des objets IBM MQ classes for JMS.

Création de destinations

Pour créer un objet Queue, une application peut utiliser le constructeur MQQueue, comme illustré dans l'exemple suivant:

```
MQQueue q1 = new MQQueue("Q1");
```

Cette instruction crée un objet MQQueue avec les valeurs par défaut pour toutes ses propriétés. L'objet représente une file d'attente IBM MQ appelée Q1 qui appartient au gestionnaire de files d'attente local. Cette file d'attente peut être une file d'attente locale, une file d'attente alias ou une définition de file d'attente éloignée.

Une autre forme du constructeur MQQueue comporte deux paramètres, comme illustré dans l'exemple suivant:

```
MQQueue q2 = new MQQueue("QM2", "Q2");
```

L'objet MQQueue créé par cette instruction représente une file d'attente IBM MQ appelée Q2 qui appartient au gestionnaire de files d'attente QM2. Le gestionnaire de files d'attente identifié de cette manière peut être le gestionnaire de files d'attente local ou un gestionnaire de files d'attente éloignées. S'il s'agit d'un gestionnaire de files d'attente éloignées, IBM MQ doit être configuré de sorte que, lorsque l'application envoie un message à cette destination, WebSphere MQ puisse acheminer le message du gestionnaire de files d'attente locales vers le gestionnaire de files d'attente éloignées.

Le constructeur MQQueue peut également accepter un URI (uniform resource identifier) de file d'attente en tant que paramètre unique. Un URI de file d'attente est une chaîne qui spécifie le nom d'une file d'attente IBM MQ et, en option, le nom du gestionnaire de files d'attente propriétaire de la file d'attente, ainsi qu'une ou plusieurs propriétés de l'objet MQQueue. L'instruction suivante contient un exemple d'URI de file d'attente:

```
MQQueue q3 = new MQQueue("queue://QM3/Q3?persistence=2&priority=5");
```

L'objet MQQueue créé par cette instruction représente une file d'attente IBM MQ appelée Q3 qui appartient au gestionnaire de files d'attente QM3, et tous les messages envoyés à cette destination sont persistants et ont une priorité de 5. Pour plus d'informations sur les URI de file d'attente, voir «uniform resource identifier (URI)», à la page 229. Pour une autre méthode de définition des propriétés d'un objet MQQueue, voir «Définition des propriétés des destinations», à la page 223.

Pour créer un objet Topic, une application peut utiliser le constructeur MQTopic, comme illustré dans l'exemple suivant:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
```

Cette instruction crée un objet MQTopic avec les valeurs par défaut pour toutes ses propriétés. L'objet représente un sujet appelé Sport / Football/Résultats.

Le constructeur MQTopic peut également accepter un URI de rubrique comme paramètre. Un URI de rubrique est une chaîne qui spécifie le nom d'une rubrique et, en option, une ou plusieurs propriétés de l'objet MQTopic. L'instruction suivante contient un exemple d'URI de rubrique:

```
MQTopic t2 = new MQTopic("topic://Sport/Tennis/Results?persistence=1&priority=0");
```

L'objet MQTopic créé par cette instruction représente une rubrique appelée Sport / Tennis/Results, et tous les messages envoyés à cette destination sont non persistants et ont une priorité de 0. Pour plus d'informations sur les URI de rubrique, voir «uniform resource identifier (URI)», à la page 229. Pour une autre méthode de définition des propriétés d'un objet MQTopic, voir «Définition des propriétés des destinations», à la page 223.

Définition des propriétés des destinations

Une application peut définir les propriétés d'une destination en appelant les méthodes appropriées de la destination. La destination peut être un objet géré ou un objet créé dynamiquement lors de l'exécution.

Prenez le code suivant, par exemple:

```
MQQueue q1 = new MQQueue("Q1");
.
q1.setPersistence(WMQConstants.WMQ_PER_PER);
q1.setPriority(5);
```

Ce code crée un objet MQQueue, puis définit deux propriétés de l'objet. La définition de ces propriétés a pour effet que tous les messages envoyés à la destination sont persistants et ont une priorité de 5.

Une application peut définir les propriétés de l'objet MQTopic de la même manière, comme illustré dans l'exemple suivant:

```
MQTopic t1 = new MQTopic("Sport/Football/Results");
.
t1.setPersistence(WMQConstants.WMQ_PER_NON);
t1.setPriority(0);
```

Ce code crée un objet MQTopic, puis définit deux propriétés de l'objet. La définition de ces propriétés a pour effet que tous les messages envoyés à la destination sont non persistants et ont une priorité de 0.

Pour plus d'informations sur les propriétés d'une destination et sur les méthodes utilisées pour définir ses propriétés, voir Propriétés des objets IBM MQ classes for JMS.

Linux AIX *Connexion à IBM MQ à partir d'une application JMS*

Pour générer une connexion, une application JMS utilise un objet **ConnectionFactory** pour créer un objet **Connection** , puis démarre la connexion.

Pour JMS 2.0 et versions ultérieures, les applications se connectent généralement à un fournisseur de messagerie à l'aide d'un objet **ConnectionFactory** et de la méthode `createContext()` .

Dans les versions antérieures de JMS, vous deviez d'abord utiliser `createConnection` pour créer un objet **Connection** , puis démarrer l'appel de connexion `getSession()` pour créer un objet **Session** pouvant effectuer des opérations de messagerie.

Un objet **JMSContext** encapsule efficacement les objets **Connection** et **Session** . Si vous souhaitez utiliser l'approche traditionnelle et créer directement les objets de connexion et de session, voir «Génération d'une connexion dans une application JMS», à la page 224 et «Création d'une session dans une application JMS», à la page 225.

Pour créer un objet **JMSContext** , une application utilise la méthode `createContext()` d'un objet **ConnectionFactory** , comme illustré dans l'exemple suivant:

```
ConnectionFactory factory;
Connection connection;
```

```
.
.
.
connection = factory.createContext();
```

Lorsqu'une connexion JMS est créée, IBM MQ classes for JMS crée un descripteur de connexion (Hconn) et démarre une conversation avec le gestionnaire de files d'attente.

Remarque : Notez que l'ID de processus d'application est utilisé comme identité d'utilisateur par défaut à transmettre au gestionnaire de files d'attente. Si l'application s'exécute en mode de transport client, cet ID de processus doit exister sur le serveur, avec les autorisations appropriées. Si vous souhaitez utiliser une autre identité, utilisez la méthode `createConnection(username,password)` .

V 9.4.0 Ce mécanisme peut également être utilisé pour fournir un jeton d'authentification. Voir Obtention d'un jeton d'authentification auprès de l'émetteur de jeton de votre choix.

JMS 1.0 *Génération d'une connexion dans une application JMS*

Pour générer une connexion dans JMS 1.0, une application JMS utilise un objet ConnectionFactory pour créer un objet Connection, puis démarre la connexion.

Pour créer un objet Connection, une application utilise la méthode createConnection() d'un objet ConnectionFactory , comme illustré dans l'exemple suivant:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Lorsqu'une connexion JMS est créée, IBM MQ classes for JMS crée un descripteur de connexion (Hconn) et démarre une conversation avec le gestionnaire de files d'attente.

L'interface de fabrique QueueConnectionet l'interface de fabrique TopicConnectionhéritent chacune de la méthode createConnection() de l'interface ConnectionFactory . Vous pouvez donc utiliser la méthode createConnection() pour créer un objet spécifique au domaine, comme illustré dans l'exemple suivant:

```
QueueConnectionFactory qcf;
Connection connection;
.
.
.
connection = qcf.createConnection();
```

Ce fragment de code crée un objet QueueConnection . Une application peut désormais effectuer une opération indépendante du domaine sur cet objet ou une opération applicable uniquement au domaine point à point. Toutefois, si l'application tente d'effectuer une opération applicable uniquement au domaine de publication / abonnement, une exception IllegalStateest émise avec le message suivant:

```
JMSMQ1112: Operation for a domain specific object was not valid.
           Operation createProducer() is not valid for type com.ibm.mq.jms.MQTopic
```

Cela est dû au fait que la connexion a été créée à partir d'une fabrique de connexions spécifique à un domaine.

Remarque : Notez que l'ID de processus d'application est utilisé comme identité d'utilisateur par défaut à transmettre au gestionnaire de files d'attente. Si l'application s'exécute en mode de transport client, cet ID de processus doit exister sur le serveur, avec les autorisations appropriées. Si vous souhaitez utiliser une autre identité, utilisez la méthode createConnection(nom d'utilisateur, mot de passe).

La spécification JMS indique qu'une connexion est créée à l'état `stopped` . Tant qu'une connexion n'est pas établie, un consommateur de message associé à la connexion ne peut pas recevoir de messages.

Pour démarrer une connexion, une application utilise la méthode start () d'un objet Connection, comme illustré dans l'exemple suivant:

```
connection.start();
```

V 9.4.0 Ce mécanisme peut également être utilisé pour fournir un jeton d'authentification. Voir Obtention d'un jeton d'authentification auprès de l'émetteur de jeton de votre choix.

JMS 1.0 *Création d'une session dans une application JMS*
Pour créer une session dans JMS 1.0, une application JMS utilise la méthode createSession() d'un objet Connection.

La méthode createSession() comporte deux paramètres:

1. Paramètre qui indique si la session est ou non transactionnelle
2. Paramètre spécifiant le mode d'accusé de réception pour la session

Par exemple, le code suivant crée une session qui n'est pas transactionnelle et dont le mode d'accusé de réception est AUTO_ACCUSE réception:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Lorsqu'une session JMS est créée, IBM MQ classes for JMS crée un descripteur de connexion (Hconn) et démarre une conversation avec le gestionnaire de files d'attente.

Un objet Session et tout objet MessageProducer ou MessageConsumer créé à partir de cet objet ne peuvent pas être utilisés simultanément par différentes unités d'exécution d'une application à unités d'exécution multiples. Le moyen le plus simple de s'assurer que ces objets ne sont pas utilisés simultanément consiste à créer un objet Session distinct pour chaque unité d'exécution.

V 9.4.0 Ce mécanisme peut également être utilisé pour fournir un jeton d'authentification. Voir Obtention d'un jeton d'authentification auprès de l'émetteur de jeton de votre choix.

Sessions de transaction dans les applications JMS
Les applications JMS peuvent exécuter des transactions locales en créant d'abord une session transactionnelle. Une application peut valider ou annuler une transaction.

Les applications JMS peuvent exécuter des transactions locales. Une transaction locale est une transaction qui implique des modifications apportées uniquement aux ressources du gestionnaire de files d'attente auquel l'application est connectée. Pour exécuter des transactions locales, une application doit d'abord créer une session transactionnelle en appelant la méthode createSession() d'un objet Connection, en spécifiant comme paramètre que la session est transactionnelle. Par la suite, tous les messages envoyés et reçus au cours de la session sont groupés dans une séquence de transactions. Une transaction se termine lorsque l'application valide ou annule les messages qu'il a envoyés et reçus depuis le début de la transaction.

Pour valider une transaction, une application appelle la méthode commit () de l'objet Session. Lorsqu'une transaction est validée, tous les messages envoyés au cours de cette transaction deviennent disponibles pour la distribution à d'autres applications ; de même, un accusé de réception est envoyé pour tous les messages reçus au cours de cette transaction, de sorte que le serveur de messagerie n'essaie pas de les distribuer à nouveau à l'application. Dans le domaine point-à-point, le serveur de messagerie retire également les messages reçus de leurs files d'attente.

Pour annuler une transaction, une application appelle la méthode rollback () de l'objet Session. Lorsqu'une transaction est annulée, tous les messages envoyés au cours de cette transaction sont supprimés du serveur de messagerie, et tous les messages reçus au cours de cette transaction sont à

nouveau disponible pour la distribution. Dans le domaine point-à-point, les messages reçus sont replacés dans leurs filesd'attente et les autres applications peuvent à nouveau les voir.

Une nouvelle transaction démarre automatiquement lorsqu'une application crée une session transactionnelle ou appelle la méthode commit () ou rollback (). Une session transactionnelle a donc toujours une transaction active.

Lorsqu'une application ferme une session transactionnelle, une annulation implicite se produit. Lorsqu'une application ferme une connexion, une annulation implicite de toutes les sessions transactionnelles de la connexion se produit.

Si une application se termine sans fermer de connexion, une annulation implicite se produit également pour toutes les sessions de transaction de la connexion.

Une transaction est entièrement contenue dans une session transactionnelle. Une transaction ne peut pas s'étendre à d'autres sessions. Cela signifie qu'une application ne peut pas envoyer ou recevoir des messages dans plusieurs sessions transactionnelles, puis valider ou annuler toutes ces actions comme une transaction unique.

Modes d'accusé de réception des sessions JMS
Chaque session non transactionnelle possède un mode d'accusé de réception qui détermine la façon dont l'application accuse réception des messages. Trois modes d'accusé de réception sont disponibles, et le choix du mode d'accusé de réception a un impact sur la conception de l'application.

Si une session n'est pas transactionnelle, la façon dont l'application accuse réception des messages dépend du mode d'accusé de réception de la session. Les trois modes d'accusé de réception sont décrits dans les paragraphes suivants :

ACCUSATION_AUTO_RÉCEPTION

La session accuse automatiquement réception de chaque message reçu par l'application.

Si des messages sont distribués de manière synchrone à l'application, la session accuse réception chaque fois qu'un appel Receive aboutit. Si les messages sont distribués de manière asynchrone, la session accuse réception d'un message chaque fois qu'un appel à la méthode onMessage() d'un programme d'écoute de messages aboutit.

Si l'application reçoit un message, mais qu'un incident empêche l'émission de l'accusé de réception, le message est à nouveau disponible pour la distribution. L'application doit donc pouvoir gérer un message qui est redistribué.

DUPS_OK_ACCUSE de réception

La session accuse réception des messages reçus par l'application au moment de leur sélection.

Ce mode d'accusé de réception réduit le volume de travail que la session doit accomplir, mais si un incident empêche d'accuser réception du message, il se peut que plusieurs messages deviennent disponibles pour une nouvelle distribution. L'application doit donc pouvoir gérer les messages qui sont redistribués.

Restriction : Dans les modes AUTO_ACKNOWLEDGE et DUPS_OK_ACKNOWLEDGE, JMS ne prend pas en charge une application qui émet une exception non gérée dans un programme d'écoute de message. Cela signifie que les messages sont toujours pris en compte lorsque le programme d'écoute des messages est renvoyé, qu'ils aient été traités avec succès ou non (à condition que les échecs ne soient pas fatals et n'empêchent pas la poursuite de l'application). Si vous avez besoin d'un contrôle plus précis de l'accusé de réception de message, utilisez les modes CLIENT_ACQUITTEMENT ou de transaction, qui donnent à l'application le contrôle total des fonctions d'accusé de réception.

CLIENT_ACCUSÉ de réception

L'application accuse réception des messages qu'elle reçoit en appelant la méthode Acknowledge de la classe Message.

L'application peut accuser réception de chaque message individuellement ou recevoir un lot de messages et appeler la méthode Acknowledge seulement pour le dernier message reçu. Dans ce

cas, l'accusé de réception est émis pour tous les messages reçus depuis l'appel précédent à cette méthode.

Conjointement avec ces modes d'accusé de réception, une application peut arrêter et redémarrer la distribution des messages dans une session en appelant la méthode Recover de la classe Session. Les messages reçus mais qui n'ont pas fait l'objet d'un accusé de réception sont redistribués. Toutefois, il est possible qu'ils ne soient pas redistribués dans le même ordre que la fois précédente. Entre temps, des messages de priorité plus élevée peuvent être arrivés, et certains messages originaux peuvent avoir expiré. Dans le domaine point-à-point, certains messages originaux peuvent avoir été consommés par une autre application.

Une application peut déterminer si un message est en cours de redistribution en examinant le contenu de la zone d'en-tête JMSRedelivered de celui-ci. Pour ce faire, l'application appelle la méthode getJMSRedelivered() de la classe Message.

Création de destinations dans une application JMS

Au lieu d'extraire des destinations en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), une application JMS peut utiliser une session pour créer des destinations de manière dynamique lors de l'exécution. Une application peut utiliser un identificateur URI (Uniform Resource Identifier) pour identifier une file d'attente IBM MQ ou une rubrique et, éventuellement, pour spécifier une ou plusieurs propriétés d'un objet de file d'attente ou de rubrique.

Utilisation d'une session pour créer des objets File d'attente

Pour créer un objet Queue, une application peut utiliser la méthode createQueue() d'un objet Session, comme illustré dans l'exemple suivant:

```
Session session;
.
Queue q1 = session.createQueue("Q1");
```

Ce code crée un objet File d'attente avec les valeurs par défaut pour toutes ses propriétés. L'objet représente une file d'attente IBM MQ appelée Q1 qui appartient au gestionnaire de files d'attente local. Cette file d'attente peut être une file d'attente locale, une file d'attente alias ou une définition de file d'attente éloignée.

La méthode createQueue() accepte également un URI de file d'attente comme paramètre. Un URI de file d'attente est une chaîne qui spécifie le nom d'une file d'attente IBM MQ et, en option, le nom du gestionnaire de files d'attente qui possède la file d'attente et une ou plusieurs propriétés de l'objet Queue. L'instruction suivante contient un exemple d'URI de file d'attente:

```
Queue q2 = session.createQueue("queue://QM2/Q2?persistence=2&priority=5");
```

L'objet File d'attente créé par cette instruction représente une file d'attente IBM MQ appelée Q2 qui appartient à un gestionnaire de files d'attente appelé QM2, et tous les messages envoyés à cette destination sont persistants et ont une priorité de 5. Le gestionnaire de files d'attente identifié de cette manière peut être le gestionnaire de files d'attente local ou un gestionnaire de files d'attente éloignées. S'il s'agit d'un gestionnaire de files d'attente éloignées, IBM MQ doit être configuré de sorte que, lorsque l'application envoie un message à cette destination, WebSphere MQ puisse acheminer le message du gestionnaire de files d'attente local vers le gestionnaire de files d'attente QM2. Pour plus d'informations sur les URI, voir «uniform resource identifier (URI)», à la page 229.

Notez que le paramètre de la méthode createQueue() contient des informations spécifiques au fournisseur. Par conséquent, l'utilisation de la méthode createQueue() pour créer un objet Queue, au lieu d'extraire un objet Queue en tant qu'objet géré à partir d'un espace de nom JNDI, peut rendre votre application moins portable.

Une application peut créer un objet TemporaryQueue à l'aide de la méthode createTemporaryQueue () d'un objet Session, comme illustré dans l'exemple suivant:

```
TemporaryQueue q3 = session.createTemporaryQueue();
```

Bien qu'une session soit utilisée pour créer une file d'attente temporaire, la portée d'une file d'attente temporaire est la connexion qui a été utilisée pour créer la session. Toutes les sessions de la connexion peuvent créer des expéditeurs et des consommateurs de messages pour la file d'attente temporaire. La file d'attente temporaire est conservée jusqu'à la fin de la connexion ou jusqu'à ce que l'application supprime explicitement la file d'attente temporaire à l'aide de la méthode TemporaryQueue.delete (), la date la plus proche étant retenue.

Lorsqu'une application crée une file d'attente temporaire, IBM MQ classes for JMS crée une file d'attente dynamique dans le gestionnaire de files d'attente auquel l'application est connectée. La propriété TEMPMODEL de la fabrique de connexions indique le nom de la file d'attente modèle utilisée pour créer la file d'attente dynamique et la propriété TEMPQPREFIX de la fabrique de connexions indique le préfixe utilisé pour former le nom de la file d'attente dynamique.

Utilisation d'une session pour créer des objets de rubrique

Pour créer un objet Topic, une application peut utiliser la méthode createTopic() d'un objet Session, comme illustré dans l'exemple suivant:

```
Session session;
.
Topic t1 = session.createTopic("Sport/Football/Results");
```

Ce code crée un objet Topic avec les valeurs par défaut pour toutes ses propriétés. L'objet représente un sujet appelé Sport / Football/Résultats.

La méthode createTopic() accepte également un URI de rubrique en tant que paramètre. Un URI de rubrique est une chaîne qui spécifie le nom d'une rubrique et, en option, une ou plusieurs propriétés de l'objet Rubrique. Le code suivant contient un exemple d'URI de rubrique:

```
String uri = "topic://Sport/Tennis/Results?persistence=1&priority=0";
Topic t2 = session.createTopic(uri);
```

L'objet Topic créé par ce code représente une rubrique appelée Sport / Tennis/Results, et tous les messages envoyés à cette destination sont non persistants et ont une priorité de 0. Pour plus d'informations sur les URI de rubrique, voir «uniform resource identifier (URI)», à la page 229.

Notez que le paramètre de la méthode createTopic() contient des informations spécifiques au fournisseur. Par conséquent, l'utilisation de la méthode createTopic() pour créer un objet Topic, au lieu d'extraire un objet Topic en tant qu'objet géré à partir d'un espace de nom JNDI, peut rendre votre application moins portable.

Une application peut créer un objet TemporaryTopic à l'aide de la méthode createTemporaryTopic () d'un objet Session, comme illustré dans l'exemple suivant:

```
TemporaryTopic t3 = session.createTemporaryTopic();
```

Bien qu'une session soit utilisée pour créer une rubrique temporaire, la portée d'une rubrique temporaire est la connexion qui a été utilisée pour créer la session. Toutes les sessions de la connexion peuvent créer des expéditeurs et des consommateurs de message pour la rubrique temporaire. La rubrique temporaire est conservée jusqu'à la fin de la connexion ou jusqu'à ce que l'application supprime explicitement la rubrique temporaire à l'aide de la méthode TemporaryTopic.delete (), la date la plus proche étant retenue.

Lorsqu'une application crée une rubrique temporaire, IBM MQ classes for JMS crée une rubrique dont le nom commence par les caractères TEMP/*tempTopicPrefix*, où *tempTopicPrefix* est la valeur de la propriété TEMPTOPICPREFIX de la fabrique de connexions.

uniform resource identifier (URI)

Un URI de file d'attente est une chaîne qui spécifie le nom d'une file d'attente IBM MQ et, en option, le nom du gestionnaire de files d'attente qui possède la file d'attente et une ou plusieurs propriétés de l'objet File d'attente créé par l'application. Un URI de rubrique est une chaîne qui spécifie le nom d'une rubrique et, éventuellement, une ou plusieurs propriétés de l'objet de rubrique créé par l'application.

Le format d'un URI de file d'attente est le suivant:

```
queue://[ qMgrName ]/qName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Le format d'un URI de rubrique est le suivant:

```
topic://topicName [? propertyName1 = propertyValue1
& propertyName2 = propertyValue2
&...]
```

Les variables de ces formats ont les significations suivantes:

nom_gest_files_attente
Nom du gestionnaire de files d'attente propriétaire de la file d'attente identifiée par l'URI.

Le gestionnaire de files d'attente peut être le gestionnaire de files d'attente local ou un gestionnaire de files d'attente éloignées. S'il s'agit d'un gestionnaire de files d'attente éloignées, IBM MQ doit être configuré de sorte que, lorsqu'une application envoie un message à la file d'attente, WebSphere MQ puisse acheminer le message du gestionnaire de files d'attente locales vers le gestionnaire de files d'attente éloignées.

Si aucun nom n'est spécifié, le gestionnaire de files d'attente local est utilisé.

qName
Nom de la file d'attente IBM MQ .

La file d'attente peut être une file d'attente locale, une file d'attente alias ou une définition de file d'attente éloignée.

Pour connaître les règles de création de noms de file d'attente, voir Règles de dénomination des objets IBM MQ.

topicName
Nom de la rubrique.

Pour connaître les règles de création de noms de rubrique, voir Règles de dénomination des objets IBM MQ. Evitez d'utiliser les caractères génériques +, #, * et? dans les noms de rubrique. Les noms de rubrique contenant ces caractères peuvent générer des résultats inattendus lorsque vous vous y abonnez. Voir Combinaison de chaînes de rubrique.

***propertyName1*, *propertyName2*, ...**
Noms des propriétés de l'objet de file d'attente ou de rubrique créé par l'application. Le Tableau 35, à la page 230 répertorie les noms de propriété valides pouvant être utilisés dans un URI.

Si aucune propriété n'est spécifiée, l'objet Queue ou Topic possède les valeurs par défaut de toutes ses propriétés.

***propertyValue1*, *propertyValue2*, ...**
Valeurs des propriétés de l'objet Queue ou Topic créé par l'application. Le Tableau 35, à la page 230 répertorie les valeurs de propriété valides qui peuvent être utilisées dans un URI.

Les crochets ([ ]) indiquent un composant facultatif et les points de suspension (...) indiquent que la liste des paires nom-valeur de propriété, si elle est présente, peut contenir une ou plusieurs paires nom-valeur.

Le Tableau 35, à la page 230 répertorie les noms de propriété et les valeurs valides qui peuvent être utilisés dans les URI de file d'attente et de rubrique. Bien que l'outil d'administration de IBM MQ JMS

utilise des constantes symboliques pour les valeurs des propriétés, les URI ne peuvent pas contenir de constantes symboliques.

Tableau 35. Noms de propriété et valeurs valides à utiliser dans les URI de file d'attente et de rubrique

Nom de la propriété	**Description**	**Valeur valides**
CCSID	Représentation des données de type caractères dans le corps d'un message lorsque IBM MQ classes for JMS transfère le message à la destination	• Tout identificateur de jeu de caractères codés pris en charge par IBM MQ.
codage	Représentation des données numériques dans le corps d'un message lorsque IBM MQ classes for JMS transfère le message à la destination	• Toute valeur valide pour la zone *Codage* dans un descripteur de message IBM MQ .
expiration	Durée de vie des messages envoyés à la destination	• -2-Comme spécifié sur l'appel send () ou, s'il n'est pas spécifié sur l'appel send (), la durée de vie par défaut de l'expéditeur de message. • 0-Un message envoyé à la destination n'expire jamais. • Entier positif indiquant la durée de vie en millisecondes.
multidiffusion	Paramètre de multidiffusion pour une rubrique lors de l'utilisation d'une connexion en temps réel à un courtier	La liste suivante contient les valeurs valides. La valeur correspondante de la propriété MULTICAST, telle qu'elle est utilisée dans l'outil d'administration de IBM MQ JMS , est associée à chaque valeur. Pour une description de la propriété MULTICAST et de ses valeurs valides, voir Propriétés des objets IBM MQ classes for JMS. • -1-ASCF • 0 - désactivé • 3-NOTR • 5-FIABLE • 7-ACTIVE

Tableau 35. Noms de propriété et valeurs valides à utiliser dans les URI de file d'attente et de rubrique (suite)

Nom de la propriété	Description	Valeur valides
persistance	Persistance des messages envoyés à la destination	• -2-Comme spécifié dans l'appel send () ou, s'il n'est pas spécifié dans l'appel send (), la persistance par défaut de l'expéditeur de message. • -1-Comme indiqué par l'attribut *DefPersistence* de la file d'attente ou de la rubrique IBM MQ . • 1-Non persistant. • 2-Persistant. • 3-Equivalent à la valeur HIGH de la propriété PERSISTENCE telle qu'elle est utilisée dans l'outil d'administration de IBM MQ JMS . Pour une explication de cette valeur, voir «JMS messages persistants», à la page 260.
priority	Priorité des messages envoyés à la destination	• -2-Comme indiqué dans l'appel send () ou, s'il n'est pas spécifié dans l'appel send (), la priorité par défaut de l'expéditeur de message. • -1-Comme spécifié par l'attribut *DefPriority* de la file d'attente ou de la rubrique IBM MQ . • Entier compris entre 0 et 9 indiquant la priorité des messages envoyés à la destination.
targetClient	Indique si les messages envoyés à la destination contiennent un en-tête MQRFH2	• 0-Les messages contiennent un en-tête MQRFH2 . • 1-Les messages ne contiennent pas d'en-tête MQRFH2 .

Par exemple, l'URI suivant identifie une file d'attente IBM MQ appelée Q1 qui appartient au gestionnaire de files d'attente local. Un objet de file d'attente créé à l'aide de cet URI possède les valeurs par défaut pour toutes ses propriétés.

```
queue:///Q1
```

L'URI suivant identifie une file d'attente IBM MQ appelée Q2 qui appartient à un gestionnaire de files d'attente appelé QM2. Tous les messages envoyés à cette destination ont une priorité de 6. Les autres propriétés de l'objet File d'attente créé à l'aide de cet URI ont leurs valeurs par défaut.

```
queue://QM2/Q2?priority=6
```

L'URI suivant identifie une rubrique appelée Sport / Athlétiques / Résultats. Tous les messages envoyés à cette destination sont non persistants et ont une priorité de 0. Les autres propriétés de l'objet Topic créé à l'aide de cet URI ont leurs valeurs par défaut.

```
topic://Sport/Athletics/Results?persistence=1&priority=0
```

Envoi de messages dans une application JMS

Pour qu'une application JMS puisse envoyer des messages à une destination, elle doit d'abord créer un objet MessageProducer pour la destination. Pour envoyer un message à la destination, l'application crée un objet Message, puis appelle la méthode send () de l'objet MessageProducer .

Une application utilise un objet MessageProducer pour envoyer des messages. Une application crée normalement un objet MessageProducer pour une destination spécifique, qui peut être une file d'attente ou une rubrique, de sorte que tous les messages envoyés à l'aide de l'expéditeur de message soient envoyés à la même destination. Par conséquent, pour qu'une application puisse créer un objet MessageProducer , elle doit d'abord créer un objet Queue ou Topic. Pour plus d'informations sur la création d'un objet de file d'attente ou de rubrique, voir les rubriques suivantes:

- «Utilisation de JNDI pour extraire des objets gérés dans une application JMS ou Jakarta Messaging», à la page 211
- «Utilisation des extensions IBM JMS», à la page 213
- «Utilisation des extensions IBM MQ JMS», à la page 220
- «Création de destinations dans une application JMS», à la page 227

Pour créer un objet MessageProducer , une application utilise la méthode createProducer() d'un objet Session, comme illustré dans l'exemple suivant:

```
MessageProducer producer = session.createProducer(destination);
```

Le paramètre `destination` est un objet de file d'attente ou de rubrique que l'application a créé précédemment.

Pour qu'une application puisse envoyer un message, elle doit créer un objet Message. Le corps d'un message contient les données d'application et JMS définit cinq types de corps de message:

- Octets
- Mappe
- Objet
- Flux
- Texte

Chaque type de corps de message possède sa propre interface JMS , qui est une sous-interface de l'interface Message, et une méthode dans l'interface Session pour créer un message avec ce type de corps. Par exemple, l'interface d'un message texte est appelée TextMessageet une application utilise la méthode createTextMessage () d'un objet Session pour créer un message texte, comme illustré dans l'instruction suivante:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Pour plus d'informations sur les messages et les corps de message, voir «Messages JMS», à la page 148.

Pour envoyer un message, une application utilise la méthode send () d'un objet MessageProducer , comme illustré dans l'exemple suivant:

```
producer.send(outMessage);
```

Une application peut utiliser la méthode send () pour envoyer des messages dans l'un ou l'autre domaine de messagerie. La nature de la destination détermine le domaine de messagerie utilisé. Toutefois, TopicPublisher, la sous-interface de MessageProducer qui est spécifique au domaine de publication /

abonnement, comporte également une méthode publish () qui peut être utilisée à la place de la méthode send (). Les deux méthodes sont fonctionnellement identiques.

Une application peut créer un objet MessageProducer sans destination spécifiée. Dans ce cas, l'application doit spécifier la destination lors de l'appel de la méthode send ().

Si une application envoie un message dans une transaction, le message n'est pas distribué à sa destination tant que la transaction n'est pas validée. Cela signifie qu'une application ne peut pas envoyer de message et recevoir une réponse au message dans la même transaction.

Une destination peut être configurée de sorte que lorsqu'une application lui envoie des messages, IBM MQ classes for JMS réachemine le message et renvoie le contrôle à l'application sans déterminer si le gestionnaire de files d'attente a reçu le message en toute sécurité. Cette opération est parfois appelée *insertion asynchrone*. Pour plus d'informations, voir «Insertion de messages de manière asynchrone dans IBM MQ classes for JMS», à la page 328.

Réception de messages dans une application JMS

Une application utilise un consommateur de message pour recevoir des messages. Un abonné à une rubrique durable est un consommateur de message qui reçoit tous les messages envoyés à une destination, y compris ceux envoyés alors que le consommateur est inactif. Une application peut sélectionner les messages qu'elle souhaite recevoir à l'aide d'un sélecteur de message et peut recevoir des messages de manière asynchrone à l'aide d'un programme d'écoute de message.

Une application utilise un objet MessageConsumer pour recevoir des messages. Une application crée un objet MessageConsumer pour une destination spécifique, qui peut être une file d'attente ou une rubrique, de sorte que tous les messages reçus à l'aide du destinataire de message soient reçus de la même destination. Par conséquent, pour qu'une application puisse créer un objet MessageConsumer , elle doit d'abord créer un objet Queue ou Topic. Pour plus d'informations sur la création d'un objet de file d'attente ou de rubrique, voir les rubriques suivantes:

- «Utilisation de JNDI pour extraire des objets gérés dans une application JMS ou Jakarta Messaging», à la page 211
- «Utilisation des extensions IBM JMS», à la page 213
- «Utilisation des extensions IBM MQ JMS», à la page 220
- «Création de destinations dans une application JMS», à la page 227

Pour créer un objet MessageConsumer , une application utilise la méthode createConsumer() d'un objet Session, comme illustré dans l'exemple suivant:

```
MessageConsumer consumer = session.createConsumer(destination);
```

Le paramètre `destination` est un objet de file d'attente ou de rubrique que l'application a créé précédemment.

L'application utilise ensuite la méthode receive () de l'objet MessageConsumer pour recevoir un message de la destination, comme illustré dans l'exemple suivant:

```
Message inMessage = consumer.receive(1000);
```

Le paramètre de l'appel receive () indique la durée en millisecondes pendant laquelle la méthode attend l'arrivée d'un message approprié si aucun message n'est disponible immédiatement. Si vous omettez ce paramètre, l'appel se bloque indéfiniment jusqu'à ce qu'un message approprié arrive. Si vous ne souhaitez pas que l'application attende un message, utilisez la méthode receiveNoWait () à la place.

La méthode receive () renvoie un message d'un type spécifique. Par exemple, lorsqu'une application reçoit un message texte, l'objet renvoyé par l'appel receive () est un objet TextMessage .

Cependant, le type d'objet déclaré renvoyé par un appel receive () est un objet Message. Par conséquent, pour extraire les données du corps d'un message qui vient d'être reçu, l'application doit être transtypée de la classe Message vers la sous-classe plus spécifique, telle que TextMessage. Si le type du message est inconnu, l'application peut utiliser l'opérateur `instanceof` pour déterminer le type. Il est toujours

recommandé pour une application de déterminer le type d'un message avant le transtypage afin que les erreurs puissent être traitées correctement.

Le code suivant utilise l'opérateur `instanceof` et montre comment extraire les données du corps d'un message texte:

```
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 une application envoie un message dans une transaction, le message n'est pas distribué à sa destination tant que la transaction n'est pas validée. Cela signifie qu'une application ne peut pas envoyer de message et recevoir une réponse au message dans la même transaction.

Si un destinataire de message reçoit des messages d'une destination configurée pour la lecture anticipée, tous les messages non persistants qui se trouvent dans la mémoire tampon de lecture anticipée à la fin de l'application sont supprimés.

Dans le domaine de publication / abonnement, JMS identifie deux types de consommateur de message, un abonné de rubrique non durable et un abonné de rubrique durable, qui sont décrits dans les deux sections suivantes.

Abonnés aux rubriques non durables

Un abonné de rubrique non durable reçoit uniquement les messages qui sont publiés alors que l'abonné est actif. Un abonnement non durable démarre lorsqu'une application crée un abonné de rubrique non durable et se termine lorsque l'application ferme l'abonné ou lorsque l'abonné est hors de portée. En tant qu'extension dans IBM MQ classes for JMS, un abonné de rubrique non durable reçoit également des publications conservées.

Pour créer un abonné de rubrique non durable, une application peut utiliser la méthode createConsumer() indépendante du domaine, en spécifiant un objet de rubrique comme destination. Une application peut également utiliser la méthode createSubscriber() spécifique au domaine, comme illustré dans l'exemple suivant:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Le paramètre `topic` est un objet de rubrique créé précédemment par l'application.

Abonnés durables aux rubriques

Restriction : Une application ne peut pas créer d'abonnés durables à une rubrique lors de l'utilisation d'une connexion en temps réel à un courtier.

Un abonné de rubrique durable reçoit tous les messages publiés pendant la durée de vie d'un abonnement durable. Ces messages incluent tous ceux qui sont publiés alors que l'abonné n'est pas actif. En tant qu'extension dans IBM MQ classes for JMS, un abonné de rubrique durable reçoit également des publications conservées.

Pour créer un abonné à une rubrique durable, une application utilise la méthode createDurableSubscriber () d'un objet Session, comme illustré dans l'exemple suivant:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

Dans l'appel createDurableSubscriber (), le premier paramètre est un objet de rubrique créé précédemment par l'application et le second paramètre est un nom utilisé pour identifier l'abonnement durable.

La session utilisée pour créer un abonné durable à une rubrique doit être associée à un identificateur de client. L'identificateur de client associé à une session est identique à l'identificateur de client de la connexion utilisée pour créer la session. L'identificateur de client peut être spécifié en définissant la propriété CLIENTID de l'objet ConnectionFactory . Une application peut également spécifier l'identificateur de client en appelant la méthode setClientID () de l'objet Connection.

Le nom utilisé pour identifier un abonnement durable doit être unique uniquement au sein de l'identificateur client. Par conséquent, l'identificateur client fait partie de l'identificateur unique complet d'un abonnement durable. Pour continuer à utiliser un abonnement durable créé précédemment, une application doit créer un abonné de rubrique durable en utilisant une session avec le même identificateur de client que celui associé à l'abonnement durable et en utilisant le même nom d'abonnement.

Un abonnement durable démarre lorsqu'une application crée un abonné de rubrique durable à l'aide d'un identificateur client et d'un nom d'abonnement pour lequel aucun abonnement durable n'existe actuellement. Toutefois, un abonnement durable ne s'arrête pas lorsque l'application ferme l'abonné durable à la rubrique. Pour mettre fin à un abonnement durable, une application doit appeler la méthode unsubscribe () d'un objet Session ayant le même identificateur client que celui associé à l'abonnement durable. Le paramètre de l'appel unsubscribe () est le nom de l'abonnement, comme illustré dans l'exemple suivant:

```
session.unsubscribe("D_SUB_000001");
```

La portée d'un abonnement durable est un gestionnaire de files d'attente. Si un abonnement durable existe sur un gestionnaire de files d'attente et qu'une application connectée à un autre gestionnaire de files d'attente crée un abonnement durable avec le même identificateur de client et le même nom d'abonnement, les deux abonnements durables sont complètement indépendants.

Sélecteurs de message

Une application peut spécifier que seuls les messages qui répondent à certains critères sont renvoyés par des appels receive () successifs. Lors de la création d'un objet MessageConsumer , l'application peut spécifier une expression SQL (Structured Query Language) qui détermine quels messages sont extraits. Cette expression SQL est appelée *sélecteur de message*. Le sélecteur de message peut contenir les noms des zones d'en-tête de message JMS et les propriétés de message. Pour plus d'informations sur la construction d'un sélecteur de message, voir «Sélecteurs de message dans JMS», à la page 149.

L'exemple suivant montre comment une application peut sélectionner des messages en fonction d'une propriété définie par l'utilisateur appelée myProp:

```
MessageConsumer consumer;
.
consumer = session.createConsumer(destination, "myProp = 'blue'");
```

La spécification JMS ne permet pas à une application de modifier le sélecteur de message d'un consommateur de message. Une fois qu'une application a créé un consommateur de message avec un sélecteur de message, le sélecteur de message est conservé pendant toute la durée de vie de ce consommateur. Si une application requiert plusieurs sélecteurs de message, elle doit créer un consommateur de message pour chaque sélecteur de message.

Notez que lorsqu'une application est connectée à un gestionnaire de files d'attente version 7, la propriété MSGSELECTION de la fabrique de connexions n'a aucun effet. Pour optimiser les performances, la sélection de tous les messages est effectuée par le gestionnaire de files d'attente.

Suppression des publications locales

Une application peut créer un consommateur de message qui ignore les publications publiées sur sa propre connexion. L'application le fait en définissant le troisième paramètre sur un appel createConsumer() à `true`, comme illustré dans l'exemple suivant:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

Sur un appel createDurableSubscriber (), l'application effectue cette opération en définissant le quatrième paramètre sur `true`, comme illustré dans l'exemple suivant:

```
String selector = "company = 'IBM'";
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001",
                                                              selector, true);
```

Distribution asynchrone des messages

Une application peut recevoir des messages de manière asynchrone en enregistrant un programme d'écoute de messages auprès d'un consommateur de messages. Le programme d'écoute de message possède une méthode appelée onMessage, qui est appelée de manière asynchrone lorsqu'un message approprié est disponible et dont l'objectif est de traiter le message. Le code suivant illustre le mécanisme:

JM 3.0

```
import jakarta.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

JMS 2.0

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that is called asynchronously when a suitable message is available
  public void onMessage(Message message)
  {
    System.out.println("Message is "+message);

    // The code to process the message
    .
    .
    .
  }
}
.
.
.
```

```
// Main program (possibly in another class)
.
// Creating the message listener
MyClass listener = new MyClass();

// Registering the message listener with a message consumer
consumer.setMessageListener(listener);

// The main program now continues with other processing
```

Une application peut utiliser une session soit pour recevoir des messages de manière synchrone à l'aide d'appels receive (), soit pour recevoir des messages de manière asynchrone à l'aide de programmes d'écoute de messages, mais pas pour les deux. Si une application doit recevoir des messages de manière synchrone et asynchrone, elle doit créer des sessions distinctes.

Une fois qu'une session est configurée pour recevoir des messages de manière asynchrone, les méthodes suivantes ne peuvent pas être appelées sur cette session ou sur les objets créés à partir de cette session:

- MessageConsumer.receive ()
- MessageConsumer.receive (long)
- MessageConsumer.receiveNoWait ()
- Session.acknowledge()
- MessageProducer.send (Destination, Message)
- MessageProducer.send (Destination, Message, int, int, long)
- MessageProducer.send (Message)
- MessageProducer.send (Message, int, int, long)
- MessageProducer.send (Destination, Message, CompletionListener)
- MessageProducer.send (Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send (Message, CompletionListener)
- MessageProducer.send (Message, int, int, long, CompletionListener)
- Session.commit()
- Session.createBrowser(File d'attente)
- Session.createBrowser(file d'attente, chaîne)
- Session.createBytesMessage()
- Session.createConsumer(Destination)
- Session.createConsumer(Destination, chaîne, valeur booléenne)
- Session.createDurableSubscriber(sujet, chaîne)
- Session.createDurableSubscriber(sujet, chaîne, chaîne, booléen)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(sérialisable)
- Session.createProducer(Destination)
- Session.createQueue(chaîne)
- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(chaîne)
- Session.createTopic()

- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(chaîne)

Si l'une de ces méthodes est appelée, une exception JMSException contenant le message:

```
JMSCC0033: Un appel de méthode synchrone n'est pas autorisé lorsqu'une session est utilisée de
manière asynchrone: 'nom de méthode'
```

est émise.

Réception de messages incohérents

Une application peut recevoir un message qui ne peut pas être traité. Il peut y avoir plusieurs raisons pour lesquelles le message ne peut pas être traité, par exemple le message peut avoir un format incorrect. Ces messages sont décrits comme des messages incohérents et nécessitent un traitement spécial pour empêcher leur traitement récursif.

Pour plus de détails sur la gestion des messages incohérents, voir «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.

Personnalisation des tailles de mémoire tampon en fonction des messages reçus

Lorsqu'un message est reçu de IBM MQ par une application nonJMS , une mémoire tampon de message doit être fournie par l'application pour que le message puisse être écrit. Les applications JMS n'ont pas besoin de créer manuellement une mémoire tampon. Le IBM MQ classes for JMS crée et dimensionne automatiquement des mémoires tampon de messages en fonction de la taille des messages reçus. Pour la plupart des applications, les tampons gérés automatiquement offrent un équilibre adéquat des performances et de la commodité pour le développeur d'applications. Dans certaines circonstances, il peut être utile de spécifier manuellement la taille initiale de la mémoire tampon de messages. La taille initiale par défaut d'une mémoire tampon de réception IBM MQ JMS est de 4 Ko. Si une application reçoit toujours des messages d'une taille de 256 Ko, il peut être préférable de configurer la taille initiale de la mémoire tampon à 256 Ko. Cela peut éviter que le IBM MQ classes for JMS tente de recevoir le message dans une mémoire tampon de 4 Ko avant de le redimensionner à 256 Ko et de le recevoir avec succès. Pour une application connectée au client, cela peut éviter d'avoir à effectuer une boucle réseau potentiellement perdue pendant que le IBM MQ classes for JMS détermine la taille de mémoire tampon correcte à utiliser.

La taille de mémoire tampon initiale peut être configurée en définissant la propriété com.ibm.mq.jmqi.defaultMaxMsgSize Java sur la valeur que vous avez choisie, en octets. Notez que cette propriété affecte toutes les applications IBM MQ JMS qui s'exécutent dans le Java Virtual Machine. Par conséquent, veillez à ne pas affecter les autres destinataires de message qui reçoivent des messages d'une taille différente.

Le IBM MQ classes for JMS tente toujours de réduire automatiquement la taille de la mémoire tampon si plusieurs messages inférieurs à la taille configurée sont reçus. Par défaut, cela se produit si 10 messages sont reçus qui sont tous plus petits que la taille de la mémoire tampon. Par exemple, si 10 messages sont reçus dans une ligne dont la taille est de 128 Ko, la mémoire tampon est réduite de 256 Ko à 128 Ko. Elle est ensuite à nouveau augmentée lorsque des messages de plus grande taille sont reçus. Il est possible de configurer le nombre de messages qui doivent être reçus avant qu'une mémoire tampon ne soit réduite en taille. Par exemple, cela peut être utile si l'application est connue pour recevoir cinq messages de grande taille suivis de 10 messages de plus petite taille, puis de cinq autres messages de grande taille. Avec les paramètres par défaut, la mémoire tampon est réduite après la réception des 10 messages plus petits et doit être à nouveau accrue pour les messages plus volumineux. La propriété système Java com.ibm.mq.jmqi.smallMsgBufferReductionThreshold peut être définie sur le nombre de messages qui doivent être reçus avant que la taille de la mémoire tampon ne soit réduite. Dans cet exemple, il peut être défini sur 20 pour empêcher 10 messages plus petits de réduire la taille de la mémoire tampon.

Les propriétés peuvent être définies indépendamment les unes des autres. Par exemple, vous pouvez choisir de laisser la taille de la mémoire tampon initiale à sa valeur par défaut de 4 ko, mais d'augmenter la valeur de com.ibm.mq.jmqi.smallMsgBufferReductionThreshold de sorte qu'une fois la taille de la mémoire tampon augmentée, elle reste plus longtemps.

Si un grand nombre de codes retour MQRC_TRUNCATED_MSG_FAILED (2080) sont visibles pour vos applications JMS dans les enregistrements de statistiques MQI, cela peut indiquer que vous profiteraient de la configuration d'une taille de mémoire tampon initiale plus élevée pour ces applications ou de la réduction de la fréquence à laquelle les tailles de mémoire tampon sont réduites. Toutefois, il est important de noter que pour une application à exécution longue, vous ne verrez probablement qu'un très petit nombre de codes retour MQRC_TRUNCATED_MSG_FAILED. En effet, la taille de la mémoire tampon est généralement augmentée à la taille correcte immédiatement après la réception du premier message de grande taille et elle n'est pas réduite à moins qu'un certain nombre de messages plus petits ne soient reçus. Il est donc possible qu'un grand nombre de messages MQRC_TRUNCATED_MSG_FAILED indique d'autres mauvaises pratiques d'application telles que la connexion à IBM MQ pour recevoir un ou deux messages avant la déconnexion.

Extraction des données utilisateur d'abonnement

Si les messages qu'une application IBM MQ classes for JMS consomme à partir d'une file d'attente sont insérés par un abonnement durable défini par l'administrateur, l'application doit accéder aux informations de données utilisateur associées à l'abonnement. Ces informations sont ajoutées au message en tant que propriété.

Lorsqu'un message est consommé à partir d'une file d'attente contenant un en-tête RFH2 avec le dossier MQPS, la valeur associée à la clé Sud, si elle existe, est ajoutée en tant que propriété String à l'objet Message JMS renvoyé à l'application IBM MQ classes for JMS . Pour activer l'extraction de cette propriété à partir du message, la constante JMS_IBM_SUBSCRIPTION_USER_DATA dans l'interface JmsConstants peut être utilisée avec la méthode suivante pour obtenir les données utilisateur de l'abonnement:

- JM 3.0 jakarta.jms.Message.getStringProperty(java.lang.String)
- JMS 2.0 javax.jms.Message.getStringProperty(java.lang.String)

Dans l'exemple suivant, un abonnement durable d'administration est défini à l'aide de la commande 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')
```

Les copies des messages publiés dans la chaîne de rubrique PUBLIC sont placées dans la file d'attente, MY.SUBSCRIPTION.Q. Les données utilisateur associées à l'abonnement durable sont ensuite ajoutées en tant que propriété au message, qui est stocké dans le dossier MQPS de l'en-tête RFH2 avec la clé Sud.

L'application IBM MQ classes for JMS peut appeler:

JM 3.0
```
jakarta.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

JMS 2.0
```
javax.jms.Message.getStringProperty(JmsConstants.JMS_IBM_SUBSCRIPTION_USER_DATA);
```

La chaîne suivante est ensuite renvoyée:

```
Administrative durable subscription to put message to the queue MY.SUBSCRIPTION.Q
```

Concepts associés

«L'en-tête MQRFH2 et JMS», à la page 153

Tâches associées

Définition d'un abonnement d'administration

Référence associée

DEFINE SUB

Interface JmsConstants

Fermeture d'une application IBM MQ classes for JMS

Il est important qu'une application IBM MQ classes for JMS ferme explicitement certains objets JMS avant de s'arrêter. Les finaliseurs pouvant ne pas être appelés, ne les utilisez pas pour libérer des ressources. N'autorisez pas une application à s'arrêter avec la trace compressée active.

La récupération de place seule ne peut pas libérer toutes les ressources IBM MQ classes for JMS et IBM MQ en temps utile, en particulier si une application crée de nombreux objets JMS de courte durée au niveau de la session ou à un niveau inférieur. Il est donc important qu'une application ferme un objet Connection, Session, MessageConsumerou MessageProducer lorsqu'il n'est plus nécessaire.

Si une application se termine sans fermer de connexion, une annulation implicite se produit pour toutes les sessions de transaction de la connexion. Pour vous assurer que les modifications apportées par l'application sont validées, fermez la connexion explicitement avant de fermer l'application.

N'utilisez pas de finaliseurs dans une application pour fermer les objets JMS . Etant donné que les finaliseurs peuvent ne pas être appelés, les ressources peuvent ne pas être libérées. Lorsqu'une connexion est fermée, elle ferme toutes les sessions qui ont été créées à partir de cette connexion. De même, les MessageConsumers et MessageProducers créés à partir d'une session sont fermés lorsque la session est fermée. Toutefois, envisagez de fermer les sessions, MessageConsumerset MessageProducers explicitement pour vous assurer que les ressources sont libérées dans les délais.

Si la compression de trace est activée, les arrêts de System.Halt() et les arrêts anormaux et incontrôlés de la machine virtuelle Java sont susceptibles de générer un fichier de trace endommagé. Dans la mesure du possible, désactivez la fonction de trace lorsque vous avez collecté les informations de trace dont vous avez besoin. Si vous tracez une application jusqu'à une fin anormale, utilisez la sortie de trace non compressée.

Remarque : Pour se déconnecter d'un gestionnaire de files d'attente, une application JMS appelle la méthode close () sur l'objet de connexion.

Traitement des messages incohérents dans IBM MQ classes for JMS

Un message incohérent est un message qui ne peut pas être traité par une application de réception. Si un message incohérent est distribué à une application et annulé un certain nombre de fois, IBM MQ classes for JMS peut le déplacer dans une file d'attente d'annulation.

Un message incohérent est un message qui ne peut pas être traité par une application de réception. Le message peut avoir un type inattendu ou contenir des informations qui ne peuvent pas être traitées par la logique de l'application. Si un message incohérent est distribué à une application, l'application ne peut pas le traiter et l'annule dans la file d'attente d'où il provient. Par défaut, IBM MQ classes for JMS redistribue à plusieurs reprises le message à l'application. Cela peut entraîner le blocage de l'application dans une boucle qui tente en permanence de traiter le message incohérent et de le rétromigrage.

Pour éviter que cela ne se produise, IBM MQ classes for JMS peut détecter des messages incohérents et les déplacer vers une autre destination. Pour ce faire, IBM MQ classes for JMS utilise les propriétés suivantes:

- Valeur de la zone BackoutCount dans le MQMD du message qui a été détecté.
- Les IBM MQ attributs de file d'attente **BOTHRESH** (seuil d'annulation) et **BOQNAME** (file de remise en file d'attente d'annulation) pour la file d'attente d'entrée contenant le message.

Chaque fois qu'un message est annulé par une application, le gestionnaire de files d'attente incrémente automatiquement la valeur de la zone BackoutCount du message.

Lorsque le IBM MQ classes for JMS détecte un message dont le BackoutCount est supérieur à zéro, il compare la valeur de BackoutCount à la valeur de l'attribut **BOTHRESH** .

- Si BackoutCount est inférieur à la valeur de l'attribut **BOTHRESH** , IBM MQ classes for JMS le distribue à l'application pour traitement.
- Toutefois, si BackoutCount est supérieur ou égal à **BOTHRESH**, le message est considéré comme un message incohérent. Dans ce cas, IBM MQ classes for JMS déplace ensuite le message dans la file

d'attente spécifiée par l'attribut **BOQNAME** . Si le message ne peut pas être inséré dans la file d'attente d'annulation, il est déplacé dans la file d'attente de rebut du gestionnaire de files d'attente ou supprimé, en fonction des options de rapport du message.

Remarque :

- Si l'attribut **BOTHRESH** reste à sa valeur par défaut `0`, la gestion des messages incohérents est désactivée. Cela signifie que les messages incohérents sont renvoyés à la file d'attente d'entrée.
- L'autre chose à noter est que IBM MQ classes for JMS interroge les attributs **BOTHRESH** et **BOQNAME** de la file d'attente la première fois qu'ils détectent un message dont le BackoutCount est supérieur à zéro. Les valeurs de ces attributs sont ensuite mises en cache et utilisées chaque fois que le IBM MQ classes for JMS rencontre un message dont le BackoutCount est supérieur à zéro.

Configuration de votre système pour le traitement des messages incohérents

La file d'attente utilisée par IBM MQ classes for JMS lors de l'interrogation des attributs **BOTHRESH** et **BOQNAME** dépend du style de messagerie utilisé:

- Pour la messagerie point-à-point, il s'agit de la file d'attente locale sous-jacente. Ceci est important lorsqu'une application JMS consomme des messages provenant de files d'attente alias ou de files d'attente de cluster.
- Pour la messagerie de publication/abonnement, une file d'attente gérée est créée afin de contenir les messages pour une application. La IBM MQ classes for JMS interroge la file d'attente gérée pour déterminer les valeurs des attributs **BOTHRESH** et **BOQNAME** .

  La file d'attente gérée est créée à partir d'une file d'attente modèle associée à l'objet Topic auquel l'application s'est abonnée et hérite des valeurs des attributs **BOTHRESH** et **BOQNAME** spécifiés dans la file d'attente modèle. La file d'attente modèle qui est utilisée varie selon que l'application de réception a souscrit un abonnement durable ou non durable :

  – La file d'attente modèle utilisée pour les abonnements durables est spécifiée par l'attribut **MDURMDL** de l'objet Topic. La valeur par défaut de cet attribut est `SYSTEM.DURABLE.MODEL.QUEUE`.

  – Pour les abonnements non durables, la file d'attente modèle utilisée est spécifiée par l'attribut **MNDURMDL**. La valeur par défaut de l'attribut **MNDURMDL** est `SYSTEM.NDURABLE.MODEL.QUEUE`.

Lors de l'interrogation des attributs **BOTHRESH** et **BOQNAME** , IBM MQ classes for JMS:

- Ouvrez la file d'attente locale ou la file d'attente cible pour une file d'attente d'alias.
- Consultez les attributs **BOTHRESH** et **BOQNAME** .
- Fermez la file d'attente locale ou la file d'attente cible pour une file d'attente alias.

Les options d'ouverture utilisées lors de l'ouverture de la file d'attente locale ou de la file d'attente cible pour une file d'attente alias dépendent de la version de IBM MQ classes for JMS utilisée:

- Pour IBM MQ classes for JMS dans IBM MQ 9.1.0 Fix Pack 1 et les versions antérieures ou IBM MQ 9.1.1, si la file d'attente locale ou la file d'attente cible d'une file d'attente d'alias est une file d'attente de cluster, IBM MQ classes for JMS ouvrez la file d'attente avec les options `MQOO_INPUT_AS_Q_DEF`, `MQOO_INQUIRE` et `MQOOOO_FAIL_IF_QUIESCING` . Cela signifie que l'utilisateur qui exécute l'application de réception doit disposer de l'accès en interrogation et en obtention dans l'instance locale de la file d'attente de cluster.

  Le IBM MQ classes for JMS ouvre tous les autres types de file d'attente locale avec les options d'ouverture `MQOO_INQUIRE` et `MQOO_FAIL_IF_QUIESCING`. Pour que le IBM MQ classes for JMS puisse interroger les valeurs des attributs, l'utilisateur exécutant l'application de réception doit disposer d'un accès en interrogation sur la file d'attente locale.
- Lors de l'utilisation de IBM MQ classes for JMS dans IBM MQ 9.1.0 Fix Pack 2 et les versions ultérieures, ou pour IBM MQ 9.1.2 et les versions ultérieures, l'utilisateur exécutant l'application de réception doit disposer d'un accès en interrogation sur la file d'attente locale, quel que soit le type de la file d'attente.

Pour déplacer des messages incohérents dans une file d'attente de remise en file d'attente d'annulation ou dans la file d'attente de rebut du gestionnaire de files d'attente, vous devez accorder à l'utilisateur exécutant l'application les droits `put` et `passall` .

Traitement des messages incohérents pour les applications synchrones

Si une application reçoit des messages de manière synchrone, en appelant l'une des méthodes suivantes, le IBM MQ classes for JMS refile un message incohérent dans l'unité de travail qui était active lorsque l'application a tenté d'obtenir le message:

- JMSConsumer.receive()
- JMSConsumer.receive(délai d'attente long)
- JMSConsumer.receiveBody(classe < T> c)
- JMSConsumer.receiveBody(classe < T> c, délai d'attente long)
- JMSConsumer.receiveBodyNoWait Classe < T> c)
- JMSConsumer.receiveNoWait()
- MessageConsumer.receive ()
- MessageConsumer.receive (délai d'attente long)
- MessageConsumer.receiveNoWait ()
- QueueReceiver.receive ()
- QueueReceiver.receive (délai d'attente long)
- QueueReceiver.receiveNoWait ()
- TopicSubscriber.receive ()
- TopicSubscriber.receive (délai d'attente long)
- TopicSubscriber.receiveNoWait ()

Cela signifie que si l'application utilise un contexte ou une session JMS transactionnelle, le déplacement du message vers la file d'attente d'annulation n'est pas validé tant que la transaction n'est pas validée.

Si l'attribut **BOTHRESH** est défini sur une valeur différente de zéro, l'attribut **BOQNAME** doit également être défini. Si **BOTHRESH** est défini sur une valeur supérieure à zéro et que **BOQNAME** n'a pas été défini, le comportement est déterminé par les options de rapport du message:

- Si l'option de rapport `MQRO_DISCARD_MSG` est définie pour le message, celui-ci est supprimé.
- Si l'option de rapport `MQRO_DEAD_LETTER_Q` est spécifiée pour le message, IBM MQ classes for JMS tente de déplacer le message dans la file d'attente de rebut du gestionnaire de files d'attente.
- Si `MQRO_DISCARD_MSG` ou `MQRO_DEAD_LETTER_Q` n'est pas défini pour le message, IBM MQ classes for JMS tente d'insérer le message dans la file d'attente de rebut du gestionnaire de files d'attente.

Si la tentative d'insertion du message dans la file d'attente des messages non livrés échoue pour une raison quelconque, ce qui arrive au message est déterminé par le fait que l'application de réception utilise un contexte ou une session JMS transactionnelle ou non transactionnelle:

- Si l'application de réception utilise un contexte ou une session JMS transactionnelle et que la transaction est validée, le message est supprimé.
- Si l'application de réception utilise un contexte ou une session JMS transactionnelle et annule la transaction, le message est renvoyé à la file d'attente d'entrée.
- Si l'application de réception a créé un contexte ou une session JMS non transactionnelle, le message est supprimé.

Traitement des messages incohérents pour les applications asynchrones

Si une application reçoit des messages de manière asynchrone via un MessageListener, la IBM MQ classes for JMS refile les messages incohérents sans affecter la distribution des messages. Le processus de

remise en file d'attente se déroule en dehors de toute unité d'oeuvre associée à la distribution réelle des messages à l'application.

Si **BOTHRESH** est défini sur une valeur supérieure à zéro et que **BOQNAME** n'a pas été défini, le comportement est déterminé par les options de rapport du message:

- Si l'option de rapport `MQRO_DISCARD_MSG` est définie pour le message, celui-ci est supprimé.
- Si l'option de rapport `MQRO_DEAD_LETTER_Q` est spécifiée pour le message, IBM MQ classes for JMS tente de déplacer le message dans la file d'attente de rebut du gestionnaire de files d'attente.
- Si `MQRO_DISCARD_MSG` ou `MQRO_DEAD_LETTER_Q` n'est pas défini pour le message, IBM MQ classes for JMS tente d'insérer le message dans la file d'attente de rebut du gestionnaire de files d'attente.

Si la tentative d'insertion du message dans la file d'attente des messages non livrés échoue pour une raison quelconque, IBM MQ classes for JMS renvoie le message à la file d'attente d'entrée.

Pour plus d'informations sur la façon dont les spécifications d'activation et les ConnectionConsumers traitent les messages incohérents, voir Suppression de messages de la file d'attente dans ASF.

Qu'arrive-t-il à un message lorsqu'il est déplacé dans la file d'attente d'annulation?

Lorsqu'un message incohérent est replacé dans la file d'attente de remise en file d'attente d'annulation, IBM MQ classes for JMS ajoutez-y un en-tête RFH2 (s'il n'en existe pas déjà un) et mettez à jour certaines des zones du descripteur de message (MQMD).

Si le message incohérent contient un en-tête RFH2 (car il s'agissait d'un message JMS , par exemple), IBM MQ classes for JMS modifie les zones suivantes dans le MQMD lors du déplacement du message vers la file d'attente de remise en file d'attente d'annulation:

- La zone BackoutCount est réinitialisée sur zéro.
- La zone Expiration du message est mise à jour pour refléter l'expiration restante au moment où le message incohérent a été reçu par l'application JMS .

Si le message incohérent ne contient pas d'en-tête RFH2 , le IBM MQ classes for JMS en ajoute un et met à jour les zones suivantes dans le MQMD dans le cadre du traitement d'annulation:

- La zone BackoutCount est réinitialisée sur zéro.
- La zone Expiration du message est mise à jour pour refléter l'expiration restante au moment où le message incohérent a été reçu par l'application JMS .
- La zone Format du message est remplacée par MQHRF2.
- La valeur de la zone CCSID est remplacée par 1208.
- Le champ Codage est modifié pour être 273.

En outre, les zones CCSID et Encoding du message incohérent sont copiées dans les zones CCSID et Encoding de l'en-tête RFH2 afin de s'assurer que le chaînage d'en-tête du message dans la file d'attente de remise en file d'attente d'annulation est correct.

Concepts associés

«Gestion des messages incohérents dans ASF», à la page 346
Dans les utilitaires du serveur d'applications, la gestion des messages incohérents est traitée de manière légèrement différente de celle effectuée ailleurs dans IBM MQ classes for JMS.

Exceptions dans IBM MQ classes for JMS

Une application IBM MQ classes for JMS doit traiter les exceptions qui sont émises par des appels d'API JMS ou qui sont distribuées à un gestionnaire d'exceptions.

IBM MQ classes for JMS signale les problèmes d'exécution en générant des exceptions. Le type des exceptions émises et la manière dont ces exceptions doivent être traitées dépendent de la version de la spécification JMS utilisée par votre application:

- Les méthodes des interfaces définies dans JMS 1.1 et les versions antérieures émettent des exceptions vérifiées. La classe de base de ces exceptions est `JMSException`. Pour plus d'informations sur la gestion des exceptions vérifiées, voir «Traitement des exceptions vérifiées», à la page 244.
- Les méthodes des interfaces ajoutées dans JMS 2.0 émettent des exceptions non vérifiées. La classe de base de ces exceptions est `JMSRuntimeException`. Pour plus d'informations sur la gestion des exceptions non vérifiées, voir «Traitement des exceptions non vérifiées», à la page 247.

Vous pouvez également enregistrer un `ExceptionListener` auprès d'un `Connection` JMS ou d'un `JMSContext`. Les classes MQ pour JMS avertissent ensuite `ExceptionListener` si un problème est détecté avec une connexion au gestionnaire de files d'attente ou si un problème se produit lors de la tentative de distribution asynchrone d'un message. Pour plus d'informations, voir «ExceptionListeners», à la page 250.

Concepts associés

IBM MQ classes for JMS

Référence associée

ASYNCEXCEPTION

Traitement des exceptions vérifiées

Les méthodes des interfaces définies dans JMS 1.1 ou version antérieure émettent des exceptions vérifiées. La classe de base de ces exceptions est `JMSException`. Par conséquent, l'interception de `JMSExceptions` fournit un moyen générique de traiter ces types d'exceptions.

Chaque `JMSException` encapsule les informations suivantes:

- Un message d'exception spécifique au fournisseur, que votre application peut obtenir en appelant la méthode `Throwable.getMessage()`.
- Un code d'erreur spécifique au fournisseur, que votre application peut obtenir en appelant la méthode `JMSException.getErrorCode()`.
- Une exception associée. Une exception émise par un appel API JMS 1.1 est souvent le résultat d'un problème de niveau inférieur qui est signalé par une autre exception liée à cette exception. Votre application peut obtenir une exception liée en appelant la méthode `JMSException.getLinkedException()` ou la méthode `Throwable.getCause()`.

Lorsque vous utilisez l'API JMS 1.1 , la plupart des exceptions émises par IBM MQ classes for JMS sont des instances de sous-classes de `JMSException`. Ces sous-classes implémentent l'interface `com.ibm.msg.client.jms.JmsExceptionDetail` , qui fournit les informations supplémentaires suivantes:

- Explication du message d'exception. Votre application peut obtenir ce message en appelant la méthode `JmsExceptionDetail.getExplanation()`.
- Une réponse utilisateur recommandée à l'exception. Votre application peut obtenir ce message en appelant la méthode `JmsExceptionDetail.getUserAction()`.
- Les clés des insertions de message dans le message d'exception. Votre application peut obtenir un itérateur pour toutes les clés en appelant la méthode `JmsExceptionDetail.getKeys()`.
- Les insertions de message dans le message d'exception. Par exemple, une insertion de message peut être le nom de la file d'attente à l'origine de l'exception et il peut être utile pour votre application d'accéder à ce nom. Votre application peut obtenir l'insertion de message correspondant à une clé spécifiée en appelant la méthode `JmsExceptionDetail.getValue()`.

Toutes les méthodes de l'interface `JmsExceptionDetail` renvoient la valeur null si aucun détail n'est disponible.

Par exemple, si une application tente de créer un expéditeur de message pour une file d'attente IBM MQ qui n'existe pas, une exception est émise avec les informations suivantes:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
```

```
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

L'exception émise, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, est une sous-classe de la classe suivante et implémente l'interface `com.ibm.msg.client.jms.JmsExceptionDetail` .

- JM 3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Exceptions associées

Une exception associée fournit des informations supplémentaires sur un incident d'exécution. Par conséquent, pour chaque `JMSException` émise, une application doit vérifier l'exception liée.

L'exception liée elle-même peut avoir une autre exception liée, de sorte que les exceptions liées forment une chaîne qui renvoie au problème sous-jacent d'origine. Une exception liée est implémentée à l'aide du mécanisme d'exception chaînée de la classe `java.lang.Throwable` , et votre application peut obtenir une exception liée en appelant la méthode `Throwable.getCause()` . Pour un `JMSException`, la méthode `getLinkedException()` délègue à la méthode `Throwable.getCause()` .

Par exemple, si une application indique un numéro de port incorrect lors de la connexion à un gestionnaire de files d'attente, les exceptions forment la chaîne suivante :

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

En général, chaque exception d'une chaîne est émise à partir d'une couche différente du code. Par exemple, les exceptions de la chaîne précédente sont émises par les couches suivantes :

- La première exception, une instance d'une sous-classe de `JMSException`, est émise par la couche commune dans IBM MQ classes for JMS.
- L'exception suivante, une instance de `com.ibm.mq.MQException`, est émise par le fournisseur de messagerie IBM MQ .
- Les deux exceptions suivantes, qui sont toutes deux des instances de `com.ibm.mq.jmqi.JmqiException`, sont émises par l'interface JMQI (Java Message Queueing Interface). JMQI est le composant utilisé par IBM MQ classes for JMS pour communiquer avec un gestionnaire de files d'attente.
- L'exception finale, une instance de `java.net.ConnectionException`, est émise par la bibliothèque de classes Java .

Pour plus d'informations sur l'architecture en couches de IBM MQ classes for JMS, voir IBM MQ classes for JMS architecture.

Vous pouvez coder votre application pour effectuer une itération dans cette chaîne afin d'extraire toutes les informations appropriées, comme illustré dans l'exemple suivant:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
```

```
import jakarta.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSException;
.
.
.
catch (JMSException je) {
  System.err.println("Caught JMSException");
  // Check for linked exceptions in JMSException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSException) {
      JMSException je1 = (JMSException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
```

```
    t = t.getCause();
  }
}
```

Notez que votre application doit toujours vérifier le type de chaque exception dans une chaîne car le type d'exception peut varier et les exceptions de types différents encapsulent des informations différentes.

Obtention d'informations spécifiques à IBM MQ sur un problème

Les instances de `com.ibm.mq.MQException` et `com.ibm.mq.jmqi.JmqiException` encapsulent IBM MQ des informations spécifiques à un problème.

Un `MQException` encapsule les informations suivantes:

- Un code achèvement, que votre application peut obtenir en appelant la méthode `getCompCode()` .
- Code anomalie que votre application peut obtenir en appelant la méthode `getReason()` .

Pour des exemples d'utilisation de ces méthodes, voir l'exemple de code dans exceptions liées.

Un `JmqiException` encapsule également un code achèvement et un code anomalie. En outre, un `JmqiException` contient les informations dans un message AMQ *nnnn* ou CSQ *nnnn* , s'il est associé à l'exception. Votre application peut obtenir les différents composants de ce message en appelant les méthodes suivantes:

- La méthode `getWmqMsgExplanation()` renvoie l'explication du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgSeverity()` renvoie la gravité du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgSummary()` renvoie le récapitulatif du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgUserResponse()` renvoie la réponse utilisateur associée au message AMQ *nnnn* ou CSQ *nnnn* .

Traitement des exceptions non vérifiées
Les méthodes des interfaces définies dans JMS 2.0 émettent des exceptions non vérifiées. La classe de base de ces exceptions est `JMSRuntimeException`. Par conséquent, l'interception de `JMSRuntimeExceptions` fournit un moyen générique de traiter ces types d'exceptions.

Chaque `JMSRuntimeException` encapsule les informations suivantes:

- Un message d'exception spécifique au fournisseur, que votre application peut obtenir en appelant la méthode `JMSRuntimeException.getMessage()` .
- Un code d'erreur spécifique au fournisseur, que votre application peut obtenir en appelant la méthode `JMSRuntimeException.getErrorCode()` .
- Une exception associée. Une exception émise par un appel API JMS 2.0 est souvent le résultat d'un problème de niveau inférieur qui est signalé par une autre exception liée à cette exception. Votre application peut obtenir une exception liée en appelant la méthode `JMSRuntimeException.getCause()` .

Lorsque vous appelez des méthodes sur les interfaces fournies par l'API JMS 2.0 , la plupart des exceptions émises par IBM MQ classes for JMS sont des instances de sous-classes de `JMSRuntimeException`. Ces sous-classes implémentent l'interface `com.ibm.msg.client.jms.JmsExceptionDetail` , qui fournit les informations supplémentaires suivantes:

- Explication du message d'exception. Votre application peut obtenir ce message en appelant la méthode `JmsExceptionDetail.getExplanation()` .
- Une réponse utilisateur recommandée à l'exception. Votre application peut obtenir ce message en appelant la méthode `JmsExceptionDetail.getUserAction()` .
- Les clés des insertions de message dans le message d'exception. Votre application peut obtenir un itérateur pour toutes les clés en appelant la méthode `JmsExceptionDetail.getKeys()` .
- Les insertions de message dans le message d'exception. Par exemple, une insertion de message peut être le nom de la file d'attente à l'origine de l'exception et il peut être utile pour votre application

d'accéder à ce nom. Votre application peut obtenir l'insertion de message correspondant à une clé spécifiée en appelant la méthode `JmsExceptionDetail.getValue()` .

Toutes les méthodes de l'interface `JmsExceptionDetail` renvoient la valeur null si aucun détail n'est disponible.

Par exemple, si une application tente de créer un `JMSProducer` pour une file d'attente IBM MQ qui n'existe pas, une exception est émise avec les informations suivantes:

```
Message : JMSWMQ2008: Failed to open MQ queue 'Q_test'.
Class : class com.ibm.msg.client.jms.DetailedInvalidDestinationException
Error Code : JMSWMQ2008
Explanation : JMS attempted to perform an MQOPEN, but IBM MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

L'exception émise, `com.ibm.msg.client.jms.DetailedInvalidDestinationException`, est une sous-classe de la classe suivante et implémente l'interface `com.ibm.msg.client.jms.JmsExceptionDetail` .

- JMS 3.0 `jakarta.jms.InvalidDestinationException`
- JMS 2.0 `javax.jms.InvalidDestinationException`

Exceptions chaînées

Généralement, les exceptions sont causées par d'autres exceptions. Par conséquent, pour chaque `JMSRuntimeException` émise, votre application doit vérifier l'exception liée.

La cause de `JMSRuntimeException` peut être une autre exception. Ces exceptions forment une chaîne qui renvoie au problème sous-jacent d'origine. La cause d'une exception est implémentée à l'aide du mécanisme d'exception chaînée de la classe `java.lang.Throwable` , et votre application peut obtenir une exception liée en appelant la méthode `Throwable.getCause()` .

Par exemple, si une application indique un numéro de port incorrect lors de la connexion à un gestionnaire de files d'attente, les exceptions forment la chaîne suivante :

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+--->
    com.ibm.mq.MQException
    |
    +--->
        com.ibm.mq.jmqi.JmqiException
        |
        +--->
            com.ibm.mq.jmqi.JmqiException
            |
            +--->
                java.net.ConnectionException
```

En général, chaque exception d'une chaîne est émise à partir d'une couche différente du code. Par exemple, les exceptions de la chaîne précédente sont émises par les couches suivantes :

- La première exception, une instance d'une sous-classe de `JMSRuntimeException`, est émise par la couche commune dans IBM MQ classes for JMS.
- L'exception suivante, une instance de `com.ibm.mq.MQException`, est émise par le fournisseur de messagerie IBM MQ .
- Les deux exceptions suivantes, qui sont toutes deux des instances de `com.ibm.mq.jmqi.JmqiException`, sont émises par l'interface JMQI (Java Message Queueing Interface). JMQI est le composant utilisé par IBM MQ classes for JMS pour communiquer avec un gestionnaire de files d'attente.

- L'exception finale, une instance de `java.net.ConnectionException`, est émise par la bibliothèque de classes Java .

Pour plus d'informations sur l'architecture en couches de IBM MQ classes for JMS, voir IBM MQ classes for JMS architecture.

Vous pouvez coder votre application pour effectuer une itération dans cette chaîne afin d'extraire toutes les informations appropriées, comme illustré dans l'exemple suivant:

JM 3.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import jakarta.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

JMS 2.0

```
import com.ibm.msg.client.jms.JmsExceptionDetail;
import com.ibm.mq.MQException;
import com.ibm.mq.jmqi.JmqiException;
import javax.jms.JMSRuntimeException;
.
.
.
catch (JMSRuntimeException je) {
  System.err.println("Caught JMSRuntimeException");
  // Check for linked exceptions in JMSRuntimeException
  Throwable t = je;
  while (t != null) {
    // Write out the message that is applicable to all exceptions
    System.err.println("Exception Msg: " + t.getMessage());
    // Write out the exception stack trace
    t.printStackTrace(System.err);

    // Add on specific information depending on the type of exception
    if (t instanceof JMSRuntimeException) {
      JMSRuntimeException je1 = (JMSRuntimeException) t;
      System.err.println("JMS Error code: " + je1.getErrorCode());
      if (t instanceof JmsExceptionDetail){
        JmsExceptionDetail jed = (JmsExceptionDetail)je1;
```

```
        System.err.println("JMS Explanation: " + jed.getExplanation());
        System.err.println("JMS Explanation: " + jed.getUserAction());
      }
    } else if (t instanceof MQException) {
      MQException mqe = (MQException) t;
      System.err.println("WMQ Completion code: " + mqe.getCompCode());
      System.err.println("WMQ Reason code: " + mqe.getReason());
    } else if (t instanceof JmqiException){
      JmqiException jmqie = (JmqiException)t;
      System.err.println("WMQ Log Message: " + jmqie.getWmqLogMessage());
      System.err.println("WMQ Explanation: " + jmqie.getWmqMsgExplanation());
      System.err.println("WMQ Msg Summary: " + jmqie.getWmqMsgSummary());
      System.err.println("WMQ Msg User Response: " + jmqie.getWmqMsgUserResponse());
      System.err.println("WMQ Msg Severity: " + jmqie.getWmqMsgSeverity());
    }
    // Get the next cause
    t = t.getCause();
  }
}
```

Notez que votre application doit toujours vérifier le type de chaque exception dans une chaîne car le type d'exception peut varier et les exceptions de types différents encapsulent des informations différentes.

Obtention d'informations spécifiques à IBM MQ sur un problème

Les instances de `com.ibm.mq.MQException` et `com.ibm.mq.jmqi.JmqiException` encapsulent IBM MQ des informations spécifiques à un problème.

Un `MQException` encapsule les informations suivantes:

- Un code achèvement, que votre application peut obtenir en appelant la méthode `getCompCode()` .
- Code anomalie que votre application peut obtenir en appelant la méthode `getReason()` .

Pour des exemples d'utilisation de ces méthodes, voir l'exemple de code dans exceptions chaînées.

Un `JmqiException` encapsule également un code achèvement et un code anomalie. En outre, un `JmqiException` contient les informations dans un message AMQ *nnnn* ou CSQ *nnnn* , s'il est associé à l'exception. Votre application peut obtenir les différents composants de ce message en appelant les méthodes suivantes:

- La méthode `getWmqMsgExplanation()` renvoie l'explication du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgSeverity()` renvoie la gravité du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgSummary()` renvoie le récapitulatif du message AMQ *nnnn* ou CSQ *nnnn* .
- La méthode `getWmqMsgUserResponse()` renvoie la réponse utilisateur associée au message AMQ *nnnn* ou CSQ *nnnn* .

ExceptionListeners
Les objets JMS `Connection` et `JMSContext` sont associés à une connexion à un gestionnaire de files d'attente. Votre application peut enregistrer un `ExceptionListener` auprès de JMS `Connection` ou `JMSContext`. Si un problème se produit et rend la connexion associée à `Connection` ou à `JMSContext` inutilisable, IBM MQ classes for JMS envoie une exception à `ExceptionListener` en appelant sa méthode `onException()` . Votre application a ensuite la possibilité de rétablir la connexion.

IBM MQ classes for JMS peut également envoyer une exception au programme d'écoute des exceptions si un problème se produit lors de la tentative de distribution asynchrone d'un message.

Programmes d'écoute des exceptions

Depuis IBM MQ 8.0.0 Fix Pack 2, pour conserver le comportement des applications JMS en cours qui configurent un JMS `MessageListener` et un JMS `ExceptionListener`, et pour garantir que les IBM MQ classes for JMS sont cohérents avec la spécification JMS , la valeur par défaut de la propriété `ConnectionFactory` ASYNCEXCEPTION est remplacée par ASYNC_EXCEPTIONS_CONNECTIONBROKEN. Par conséquent, seules les exceptions qui correspondent à des codes d'erreur de connexion interrompue sont envoyées à l' `ExceptionListener` d'une application.

L'APAR IT14820, inclus depuis IBM MQ 9.0.0 Fix Pack 1, met à jour IBM MQ classes for JMS aux fins suivantes :

- Un `ExceptionListener` enregistré par une application est appelé pour toutes les exceptions de connexion interrompue, que l'application utilise des consommateurs de messages synchrones ou asynchrones.
- Les exceptions non liées à une connexion interrompue (par exemple, MQRC_GET_INHIBÉE) qui surviennent lors de la distribution des messages sont distribuées à l' `ExceptionListener` d'une application lorsque l'application utilise des consommateurs de messages asynchrones et que la propriété ConnectionFactory JMS utilisée par l'application a la valeur ASYNC_EXCEPTION définie sur ASYNC_EXCEPTIONS_ALL.

Remarque : Un `ExceptionListener` est appelé une seule fois pour une exception de connexion interrompue, même si deux connexions TCP/IP (une utilisée par une connexion JMS et une utilisée par une session JMS) sont interrompues.

Pour tout autre type de problème, une exception est émise par l'appel API JMS en cours. Le type d'exception émise dépend de la version de l'API JMS utilisée par l'application:

- Si l'application utilise les interfaces fournies par la spécification JMS 1.1 , l'exception est un `JMSException`. Pour plus d'informations sur la gestion de ces exceptions, voir «Traitement des exceptions vérifiées», à la page 244.
- Si l'application utilise des interfaces JMS 2.0 , l'exception est un `JMSRuntimeException`. Pour plus d'informations sur la gestion de ces exceptions, voir «Traitement des exceptions non vérifiées», à la page 247.

Si une application n'enregistre pas de programme d'écoute des exceptions auprès d'un `Connection` ou d'un `JMSContext`, toutes les exceptions qui seraient distribuées au programme d'écoute des exceptions sont consignées dans le journal IBM MQ classes for JMS .

Accès aux fonctions IBM MQ à partir d'une application IBM MQ classes for JMS

IBM MQ classes for JMS fournit des fonctions permettant d'exploiter un certain nombre de fonctions d' IBM MQ.

Avertissement : Ces fonctions sont en dehors de la spécification JMS ou, dans certains cas, violent la spécification JMS . Si vous les utilisez, il est peu probable que votre application soit compatible avec d'autres fournisseurs JMS . Les fonctions qui ne sont pas conformes à la spécification JMS sont étiquetées avec un avis d'attention.

Lecture et écriture du descripteur de message à partir d'une application IBM MQ classes for JMS

Vous contrôlez la possibilité d'accéder au descripteur de message (MQMD) en définissant des propriétés sur une destination et un message.

Certaines applications IBM MQ nécessitent que des valeurs spécifiques soient définies dans le MQMD des messages qui leur sont envoyés. IBM MQ classes for JMS fournit des attributs de message qui permettent aux applications JMS de définir des zones MQMD et d'activer ainsi les applications JMS pour "piloter" les applications IBM MQ .

Vous devez définir la propriété d'objet de destination WMQ_MQMD_WRITE_ENABLED sur true pour que la définition des propriétés MQMD ait un effet. Vous pouvez ensuite utiliser les méthodes de définition de propriété du message (par exemple, la propriété setString) pour affecter des valeurs aux zones MQMD. Toutes les zones MQMD sont exposées, à l'exception de StrucId et de Version ; BackoutCount peut être lu mais n'est pas accessible en écriture.

Cet exemple génère un message placé dans une file d'attente ou une rubrique avec MQMD.UserIdentifier défini sur "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
// ...
```

Il est nécessaire de définir WMQ_MQMD_MESSAGE_CONTEXT avant de définir JMS_IBM_MQMD_UserIdentifier. Pour plus d'informations sur l'utilisation de WMQ_MQMD_MESSAGE_CONTEXT, voir «Propriétés de l'objet de message JMS», à la page 254.

De même, vous pouvez extraire le contenu des zones MQMD en définissant WMQ_MQMD_READ_ENABLED sur true avant de recevoir un message, puis en utilisant les méthodes get du message, telles que la propriété getString. Les propriétés reçues sont en lecture seule.

Dans cet exemple, la zone *value* contient la valeur de MQMD.ApplIdentityData d'un message provenant d'une file d'attente ou d'une rubrique.

```
// 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");
```

Propriétés de l'objet de destination JMS
Deux propriétés de l'objet Destination contrôlent l'accès au MQMD à partir de JMS, et une troisième contrôle le contexte de message.

Tableau 36. Noms et descriptions des propriétés

Propriété	**Forme abrégée**	**Description**
WMQ_MQMD_WRITE_ENABLED	MDW	Indique si une application JMS peut définir les valeurs des zones MQMD
WMQ_MQMD_READ_ENABLED	MDR	Indique si une application JMS peut extraire les valeurs des zones MQMD
WMQ_MQMD_MESSAGE_CONTEXTE	MDCTX	Niveau de contexte de message à définir par l'application JMS . L'application doit être exécutée avec les droits de contexte appropriés pour que cette propriété soit prise en compte

Tableau 37. Noms de propriété, valeurs et méthodes de définition

Propriété	Valeurs valides dans l'outil d'administration (valeurs par défaut en gras)	Valeurs admises dans les programmes	Méthode de définition
WMQ_MQMD_WRITE ACTIVÉ	• **NO** Toutes les propriétés JMS_IBM_MQMD* sont ignorées et leurs valeurs ne sont pas copiées dans la structure MQMD sous-jacente. • YES Les propriétés JMS_IBM_MQMD* sont traitées. Leurs valeurs sont copiées dans la structure sous-jacente.	• **faux** • True	setMQMDWriteactivé
WMQ_MQMD_READ ACTIVÉ	• **NO** Lors de l'envoi de messages, les propriétés JMS_IBM_MQMD* d'un message envoyé ne sont pas mises à jour pour refléter les valeurs de zone mises à jour dans le MQMD. Lors de la réception de messages, aucune des propriétés JMS_IBM_MQMD* n'est disponible sur un message reçu, même si l'émetteur les a définies partiellement ou intégralement. • YES Lors de l'envoi de messages, toutes les propriétés JMS_IBM_MQMD* d'un message envoyé sont mises à jour pour refléter les valeurs de zone mises à jour dans le MQMD, y compris celles que l'expéditeur n'a pas définies explicitement. Lors de la réception de messages, toutes les propriétés JMS_IBM_MQMD* sont disponibles sur un message reçu, y compris celles que l'émetteur n'a pas définies de manière explicite.	• **faux** • True	setMQMDReadactivé

Tableau 37. Noms de propriété, valeurs et méthodes de définition (suite)			
Propriété	**Valeurs valides dans l'outil d'administration (valeurs par défaut en gras)**	**Valeurs admises dans les programmes**	**Méthode de définition**
WMQ_MQMD _MESSAGE_CONTEXT	• **Par défaut** L'appel d'API MQOPEN et la structure MQPMO ne spécifient aucune option de contexte de message explicite • SET_IDENTITY_CONTEXT L'appel d'API MQOPEN indique l'option de contexte de message MQOO_SET_IDENTITY_CONTEXT et la structure MQPMO indique MQPMO_SET_IDENTITY_CONTEXT • SET_ALL_CONTEXT L'appel d'API MQOPEN spécifie l'option de contexte de message MQOO_SET_ALL_CONTEXT et la structure MQPMO spécifie MQPMO_SET_ALL_CONTEXT	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	Contexte setMQMDMessage

Propriétés de l'objet de message JMS
Les propriétés d'objet de message préfixées JMS_IBM_MQMD vous permettent de définir ou de lire la zone MQMD correspondante.

Envoi de messages

Toutes les zones MQMD à l'exception de StrucId et Version sont représentées. Ces propriétés font référence uniquement aux zones MQMD ; lorsqu'une propriété se trouve à la fois dans le MQMD et dans l'en-tête MQRFH2, la version de MQRFH2 n'est ni définie ni extraite.

Toutes ces propriétés peuvent être définies, à l'exception de JMS_IBM_MQMD_BackoutCount. Toute valeur définie JMS_IBM_MQMD_BackoutCount est ignorée.

Si une propriété a une longueur maximale et que vous indiquez une valeur trop longue, celle-ci est tronquée.

Pour certaines propriétés, vous devez également définir la propriété WMQ_MQMD_MESSAGE_CONTEXT sur l'objet Destination. L'application doit être exécutée avec les droits appropriés pour que cette propriété soit appliquée. Si vous ne définissez pas WMQ_MQMD_MESSAGE_CONTEXT sur une valeur appropriée, la valeur de la propriété est ignorée. Si vous affectez à WMQ_MQMD_MESSAGE_CONTEXT une valeur appropriée mais que vous ne disposez pas de droits de contexte suffisants pour le gestionnaire de files d'attente, une exception JMSException est émise. Les propriétés nécessitant des valeurs spécifiques de WMQ_MQMD_MESSAGE_CONTEXT sont les suivantes.

Les propriétés suivantes requièrent que WMQ_MQMD_MESSAGE_CONTEXT soit défini sur WMQ_MDCTX_SET_IDENTITY_CONTEXT ou WMQ_MDCTX_SET_ALL_CONTEXT:

- JMS_IBM_MQMD_UserIdentifier
- JMS_IBM_MQMD_AccountingToken
- JMS_IBM_MQMD_ApplIdentityData

Les propriétés suivantes nécessitent que WMQ_MQMD_MESSAGE_CONTEXT soit défini sur 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

Réception de messages

Toutes ces propriétés sont disponibles dans un message reçu si la propriété WMQ_MQMD_READ_ENABLED est définie sur true, quelles que soient les propriétés réelles définies par l'application productrice. Une application ne peut pas modifier les propriétés d'un message reçu sauf si toutes les propriétés sont effacées en premier, conformément à la spécification JMS . Les messages reçus peuvent être réacheminés sans modification des propriétés.

Avertissement : Si votre application reçoit un message d'une destination avec la propriété WMQ_MQMD_READ_ENABLED définie sur true et le transmet à une destination avec la propriété WMQ_MQMD_WRITE_ENABLED définie sur true, toutes les valeurs de zone MQMD du message reçu sont copiées dans le message transféré.

Table des propriétés

Ce tableau répertorie les propriétés de l'objet Message représentant les zones MQMD. Consultez les liens pour obtenir une description complète des zones et de leurs valeurs admises.

Tableau 38. Noms, descriptions et types de propriété

Propriété	Description	Java Tapez	Lien vers la description complète
JMS_IBM_MQMD_REPORT	Options des messages de rapport	Entier	Rapport
JMS_IBM_MQMD_MSGTYPE	Type de message	Entier	MsgType
JMS_IBM_MQMD_EXPIRY	Durée de vie des messages	Entier	Expiration
JMS_IBM_MQMD_FEEDBACK	Commentaires en retour ou code anomalie	Entier	Commentaires
JMS_IBM_MQMD_ENCODING	Codage numérique des données de message	Entier	Codage
JMS_IBM_MQMD_CODEDCHARSETID	Identificateur de jeu de caractères des données de message	Entier	CodedCharSetId
JMS_IBM_MQMD_FORMAT	Nom de format des données de message	String	Format
JMS_IBM_MQMD_Priority [1]	Priorité de message	Entier	Priorité
JMS_IBM_MQMD_PERSISTENCE	Persistance des messages	Entier	Persistance
JMS_IBM_MQMD_MsgId [2]	Identificateur de message	Objet (byte [ ]) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Identificateur de corrélation	Objet (byte [ ]) [4]	CorrelId

Tableau 38. Noms, descriptions et types de propriété (suite)

Propriété	Description	Java Tapez	Lien vers la description complète
JMS_IBM_MQMD_BACKOUTCOUNT	Nombre d'annulations	Entier	BackoutCount
JMS_IBM_MQMD_REPLYTOQ	Nom de la file d'attente de réponses	String	ReplyToQ
JMS_IBM_MQMD_REPLYTOQMGR	Nom du gestionnaire de files d'attente de réponses	String	ReplyToQMgr
JMS_IBM_MQMD_UserIdentifier	ID utilisateur	String	UserIdentifier
JMS_IBM_MQMD_AccountingToken	Jeton de comptabilité	Objet (byte [ ]) [4]	AccountingToken
JMS_IBM_MQMD_ApplIdentityData	Données d'application relatives à l'identité	String	ApplIdentityData
JMS_IBM_MQMD_PutApplType	Type de l'application ayant placé le message en file d'attente	Entier	PutApplType
JMS_IBM_MQMD_PutApplName	Nom de l'application ayant placé le message en file d'attente	String	PutApplName
JMS_IBM_MQMD_PutDate	Date à laquelle le message a été placé en file d'attente	String	PutDate
JMS_IBM_MQMD_PutTime	Heure à laquelle le message a été placé en file d'attente	String	PutTime
JMS_IBM_MQMD_ApplOriginData	Données d'application relatives à l'origine	String	ApplOriginData
JMS_IBM_MQMD_GROUPID	Identificateur de groupe	Objet (byte [ ]) [4]	GroupId
JMS_IBM_MQMD_MSGSEQNUMBER	Numéro de séquence du message logique dans le groupe	Entier	MsgSeqNumber
JMS_IBM_MQMD_OFFSET	Décalage des données du message physique à partir du début du message logique	Entier	offset
JMS_IBM_MQMD_MSGFLAGS	Indicateurs de message	Entier	MsgFlags
JMS_IBM_MQMD_ORIGINALLENGTH	Longueur du message d'origine	Entier	OriginalLength

1. **Avertissement :** Si vous affectez à JMS_IBM_MQMD_Priority une valeur qui n'est pas comprise entre 0 et 9, cela enfreint la spécification JMS .
2. **Avertissement :** La spécification JMS indique que l'ID message doit être défini par le fournisseur JMS et qu'il doit être unique ou null. Si vous affectez une valeur à

JMS_IBM_MQMD_MSGID, celle-ci est copiée dans JMSMessageID. Par conséquent, il n'est pas défini par le fournisseur JMS et peut ne pas être unique: cela enfreint la spécification JMS .

3. **Avertissement :** Si vous affectez une valeur à JMS_IBM_MQMD_CorrelId qui commence par la chaîne'ID:', cela enfreint la spécification JMS .

4. **Avertissement :** L'utilisation de propriétés de tableau d'octets sur un message enfreint la spécification JMS .

Accès aux données de message IBM MQ à partir d'une application à l'aide de IBM MQ classes for JMS
Vous pouvez accéder à l'ensemble des données de message IBM MQ dans une application à l'aide de IBM MQ classes for JMS. Pour accéder à toutes les données, le message doit être un JMSBytesMessage. Le corps du fichier JMSBytesMessage inclut tout en-tête MQRFH2 , tout autre en-tête IBM MQ et les données de message suivantes.

Définissez la propriété WMQ_MESSAGE_BODY de la destination sur WMQ_MESSAGE_BODY_MQpour recevoir toutes les données de corps de message dans JMSBytesMessage.

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_JMS ou WMQ_MESSAGE_BODY_UNSPECIFIED, le corps du message est renvoyé sans l'en-tête JMS MQRFH2 et les propriétés de JMSBytesMessage reflètent les propriétés définies dans RFH2.

Certaines applications ne peuvent pas utiliser les fonctions décrites dans cette rubrique. Si une application est connectée à un gestionnaire de files d'attente IBM MQ V6 ou si elle a défini PROVIDERVERSION sur 6, les fonctions ne sont pas disponibles.

Envoi d'un message

Lors de l'envoi de messages, la propriété de destination WMQ_MESSAGE_BODYest prioritaire sur WMQ_TARGET_CLIENT.

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_JMS, IBM MQ classes for JMS génère automatiquement un en-tête MQRFH2 en fonction des paramètres des propriétés JMSMessage et des zones d'en-tête.

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_MQ, aucun en-tête supplémentaire n'est ajouté au corps du message

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_UNSPECIFIED, IBM MQ classes for JMS envoie un en-tête MQRFH2 , sauf si WMQ_TARGET_CLIENT est défini sur WMQ_TARGET_DEST_MQ. Lors de la réception, la définition de WMQ_TARGET_CLIENT sur WMQ_TARGET_DEST_MQ entraîne la suppression de tous les MQRFH2 du corps du message.

Remarque : JMSBytesMessage et JMSTextMessage ne nécessitent pas de MQRFH2, contrairement à JMSStreamMessage, JMSMapMessageet JMSObjectMessage .

WMQ_MESSAGE_BODY_UNSPECIFIED est le paramètre par défaut pour WMQ_MESSAGE_BODYet WMQ_TARGET_DEST_JMS est le paramètre par défaut pour WMQ_TARGET_CLIENT.

Si vous envoyez un JMSBytesMessage, vous pouvez remplacer les paramètres par défaut du corps du message JMS lorsque le message IBM MQ est généré. Utilisez les propriétés suivantes :

- JMS_IBM_Format ou JMS_IBM_MQMD_Format: cette propriété indique le format de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.
- JMS_IBM_Character_Set ou JMS_IBM_MQMD_CodedCharSetId: cette propriété indique le CCSID de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: cette propriété indique le codage de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.

Si les deux types de propriété sont spécifiés, les propriétés JMS_IBM_MQMD_* remplacent les propriétés JMS_IBM_* correspondantes, tant que la propriété de destination WMQ_MQMD_WRITE_ENABLED est définie sur true.

Les différences d'effet entre la définition des propriétés de message à l'aide de JMS_IBM_MQMD_* et de JMS_IBM_* sont significatives:

1. Les propriétés JMS_IBM_MQMD_* sont spécifiques au fournisseur IBM MQ JMS .
2. Les propriétés JMS_IBM_MQMD_* ne sont définies que dans MQMD. Les propriétés JMS_IBM_* sont définies dans MQMD uniquement si le message ne comporte pas d'en-tête MQRFH2 JMS . Sinon, ils sont définis dans l'en-tête JMS RFH2 .
3. Les propriétés JMS_IBM_MQMD_* n'ont aucune incidence sur le codage du texte et des nombres écrits dans un JMSMessage.

   Une application de réception suppose probablement que les valeurs de MQMD.Encoding et MQMD.CodedCharSetId correspondent au codage et au jeu de caractères des nombres et du texte dans le corps du message. Si des propriétés JMS_IBM_MQMD_* sont utilisées, il incombe à l'application émettrice de le faire. Le codage et le jeu de caractères des nombres et du texte dans le corps du message sont définis par les propriétés JMS_IBM_* .

   Le fragment mal codé dans Figure 39, à la page 258 envoie un message codé dans le jeu de caractères 1208, avec MQMD.CodedCharSetId défini sur 37.

   a. Envoyer un message codé de manière incorrecte

   ```
   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. Réception du message, en fonction de la valeur de JMS_IBM_CHARACTER_SET définie par la valeur de MQMD.CodedCharSetId:

   ```
   TextMessage tmi = (TextMessage) cons.receive();
   System.out.println("Message is \"" + tmi.getText() + "\"");
   ```

   c. Résultat:

   ```
   Message is "éÈÊ'>...??>?"
   ```

Figure 39. Données de message et MQMD codées de manière incohérente

L'un des fragments de code dans Figure 40, à la page 259 entraîne l'insertion d'un message dans une file d'attente ou une rubrique, avec son corps contenant le contenu de l'application sans qu'un en-tête MQRFH2 généré automatiquement soit ajouté.

1. Définition de WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Définition de WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Figure 40. Envoyez un message avec un corps de message MQ .

Réception d'un message

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_JMS, le type et le corps du message JMS entrant sont déterminés par le contenu du message WebSphere MQ reçu. Le type et le corps du message sont déterminés par les zones de l'en-tête MQRFH2 , ou dans MQMD, s'il n'y a pas de MQRFH2.

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_MQ, le type de message JMS entrant est JMSBytesMessage. Le corps du message JMS correspond aux données de message renvoyées par l'appel API MQGET sous-jacent. La longueur du corps du message correspond à la longueur renvoyée par l'appel MQGET . Le jeu de caractères et le codage des données dans le corps du message sont déterminés par les zones CodedCharSetId et Encoding de MQMD. Le format des données dans le corps du message est déterminé par la zone Format du fichier MQMD

Si WMQ_MESSAGE_BODY est défini sur WMQ_MESSAGE_BODY_UNSPECIFIED, la valeur par défaut, IBM MQ classes for JMS , est définie sur WMQ_MESSAGE_BODY_JMS.

Lorsque vous recevez un JMSBytesMessage, vous pouvez le décoder par référence aux propriétés suivantes:

- JMS_IBM_Format ou JMS_IBM_MQMD_Format: cette propriété indique le format de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.
- JMS_IBM_Character_Set ou JMS_IBM_MQMD_CodedCharSetId: cette propriété indique le CCSID de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: cette propriété indique le codage de l'en-tête IBM MQ ou du contenu de l'application qui démarre le corps du message JMS s'il n'y a pas d'en-tête WebSphere MQ précédent.

Le fragment de code suivant génère un message reçu qui est un JMSBytesMessage. Quel que soit le contenu du message reçu et de la zone de format du MQMDreçu, le message est un JMSBytesMessage.

```
  ((MQDestination)destination).setMessageBodyStyle
                              (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Propriété de destination WMQ_MESSAGE_BODY
WMQ_MESSAGE_BODY détermine si une application JMS traite l'en-tête MQRFH2 d'un message IBM MQ dans le cadre de la charge utile du message (c'est-à-dire dans le cadre du corps du message JMS).

Tableau 39. Noms et descriptions des propriétés		
Propriété	**Forme abrégée**	**Description**
WMQ_MESSAGE_BODY	MBODY	Indique si une application JMS traite l'en-tête MQRFH2 d'un message IBM MQ dans le cadre de la charge de message (c'est-à-dire dans le cadre du corps du message JMS).

Tableau 40. Noms de propriété, valeurs et méthodes de définition			
Propriété	**Valeurs valides dans l'outil d'administration (valeurs par défaut en gras)**	**Valeurs admises dans les programmes**	**Méthode de définition**
WMQ_MESSAGE_BODY	• **NON SPECIFIE** Lors de l'envoi, IBM MQ classes for JMS génère ou non et inclut un en-tête MQRFH2 , en fonction de la valeur de WMQ_TARGET_CLIENT. Lors de la réception, agit en tant que valeur JMS. • JMS Lors de l'envoi, IBM MQ classes for JMS génère automatiquement un en-tête MQRFH2 et l'inclut dans le message IBM MQ . Lors de la réception, IBM MQ classes for JMS définit les propriétés de message JMS en fonction des valeurs dans MQRFH2 (le cas échéant) ; il ne présente pas MQRFH2 dans le corps du message JMS . • MQ Lors de l'envoi, IBM MQ classes for JMS ne génère pas de MQRFH2. Lors de la réception, IBM MQ classes for JMS présente MQRFH2 dans le corps du message JMS .	• **WMQ_MESSAGE_BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

JMS messages persistants
Les applications IBM MQ classes for JMS peuvent utiliser l'attribut de file d'attente **NonPersistentMessageClass** pour améliorer les performances des messages persistants JMS , au détriment d'une certaine fiabilité.

Une file d'attente IBM MQ possède un attribut appelé **NonPersistentMessageClass**. La valeur de cet attribut détermine si les messages non persistants de la file d'attente sont supprimés au redémarrage du gestionnaire de files d'attente.

Vous pouvez définir l'attribut d'une file d'attente locale à l'aide de la commande IBM MQ Script (MQSC), DEFINE QLOCAL, avec l'un des paramètres suivants:

NPMCLASS (NORMAL)
Les messages non persistants de la file d'attente sont supprimés lorsque le gestionnaire de files d'attente redémarre. Il s'agit de la valeur par défaut.

NPMCLASS (ELEVEE)
Les messages non persistants de la file d'attente ne sont pas supprimés lorsque le gestionnaire de files d'attente redémarre à la suite d'un arrêt au repos ou immédiat. Les messages non persistants peuvent être supprimés, toutefois, à la suite d'un arrêt préventif ou d'un échec.

Cette rubrique explique comment les applications IBM MQ classes for JMS peuvent utiliser cet attribut de file d'attente pour améliorer les performances des messages persistants JMS .

La propriété PERSISTENCE d'un objet Queue ou Topic peut avoir la valeur HIGH. Vous pouvez utiliser l'outil d'administration IBM MQ JMS pour définir cette valeur ou une application peut appeler la méthode Destination.setPersistence() en transmettant la valeur WMQConstants.WMQ_PER_NPHIGH en tant que paramètre.

Si une application envoie un message persistant JMS ou un message non persistant JMS à une destination où la propriété PERSISTENCE a la valeur HIGH et que la file d'attente IBM MQ sous-jacente est définie sur NPMCLASS (HIGH), le message est inséré dans la file d'attente en tant que message non persistant IBM MQ . Si la propriété PERSISTENCE de la destination n'a pas la valeur HIGH ou si la file d'attente sous-jacente est définie sur NPMCLASS (NORMAL), un message persistant JMS est inséré dans la file d'attente en tant que message persistant IBM MQ et un message non persistant JMS est inséré dans la file d'attente en tant que message non persistant IBM MQ .

Si un message persistant JMS est inséré dans une file d'attente en tant que message non persistant IBM MQ et que vous souhaitez vous assurer que le message n'est pas supprimé à la suite d'un arrêt au repos ou immédiat d'un gestionnaire de files d'attente, toutes les files d'attente par lesquelles le message peut être acheminé doivent être définies sur NPMCLASS (HIGH). Dans le domaine de publication / abonnement, ces files d'attente incluent les files d'attente d'abonné. Pour faciliter l'application de cette configuration, IBM MQ classes for JMS émet une exception InvalidDestinationsi une application tente de créer un destinataire de message pour une destination où la propriété PERSISTENCE a la valeur HIGH et la file d'attente IBM MQ sous-jacente a la valeur NPMCLASS (NORMAL).

La définition de la propriété PERSISTENCE d'une destination sur HIGH n'affecte pas la manière dont un message est reçu de cette destination. Un message envoyé en tant que message persistant JMS est reçu en tant que message persistant JMS et un message envoyé en tant que message non persistant JMS est reçu en tant que message non persistant JMS .

Lorsqu'une application envoie le premier message à une destination où la propriété PERSISTENCE a la valeur HIGH ou lorsqu'une application crée le premier consommateur de message pour une destination où la propriété PERSISTENCE a la valeur HIGH, IBM MQ classes for JMS émet un appel MQINQ pour déterminer si NPMCLASS (HIGH) est défini sur la file d'attente IBM MQ sous-jacente. L'application doit donc avoir le droit d'interroger la file d'attente. En outre, IBM MQ classes for JMS conserve le résultat de l'appel MQINQ jusqu'à ce que la destination soit supprimée et n'émet pas d'autres appels MQINQ. Par conséquent, si vous modifiez le paramètre NPMCLASS dans la file d'attente sous-jacente alors que l'application utilise toujours la destination, IBM MQ classes for JMS ne remarque pas le nouveau paramètre.

En autorisant les messages persistants JMS à être placés dans des files d'attente IBM MQ en tant que messages non persistants IBM MQ , vous gagnez en performances au détriment d'une certaine fiabilité. Si vous avez besoin d'une fiabilité maximale pour les messages persistants JMS , n'envoyez pas les messages à une destination où la propriété PERSISTENCE a la valeur HIGH.

La couche JMS peut utiliser SYSTEM.JMS.TEMPQ.MODEL, au lieu de SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEME SYSTEM.JMS.TEMPQ.MODEL crée des files d'attente dynamiques permanentes qui acceptent

les messages persistants, car SYSTEM.DEFAULT.MODEL.QUEUE ne peut pas accepter les messages persistants. Pour utiliser des files d'attente temporaires afin d'accepter des messages persistants, vous devez donc utiliser SYSTEM.JMS.TEMPQ.MODEL, ou remplacez la file d'attente modèle par une autre file d'attente de votre choix.

Utilisation de TLS avec IBM MQ classes for JMS
Les applications IBM MQ classes for JMS peuvent utiliser le chiffrement TLS (Transport Layer Security). Pour ce faire, ils ont besoin d'un fournisseur JSSE.

Les connexions IBM MQ classes for JMS utilisant TRANSPORT (CLIENT) prennent en charge le chiffrement TLS. TLS fournit le chiffrement des communications, l'authentification et l'intégrité des messages. Il est généralement utilisé pour sécuriser les communications entre deux homologues sur Internet ou dans un intranet.

IBM MQ classes for JMS utilise Java Secure Socket Extension (JSSE) pour gérer le chiffrement TLS et requiert par conséquent un fournisseur JSSE. Les machines virtuelles Java JSE v1.4 ont un fournisseur JSSE intégré. Les détails de la gestion et du stockage des certificats peuvent varier d'un fournisseur à l'autre. Pour plus d'informations à ce sujet, consultez la documentation de votre fournisseur JSSE.

Cette section suppose que votre fournisseur JSSE est correctement installé et configuré, et que des certificats appropriés ont été installés et mis à la disposition de votre fournisseur JSSE. Vous pouvez désormais utiliser JMSAdmin pour définir un certain nombre de propriétés d'administration.

Si votre application IBM MQ classes for JMS utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290.

propriété d'objet SSLCIPHERSUITE
Définissez SSLCIPHERSUITE pour activer le chiffrement TLS sur un objet ConnectionFactory .

Pour activer le chiffrement TLS sur un objet ConnectionFactory , utilisez JMSAdmin pour définir la propriété SSLCIPHERSUITE sur une CipherSuite prise en charge par votre fournisseur JSSE. Cette valeur doit correspondre à la valeur CipherSpec définie sur le canal cible. Toutefois, les CipherSuites sont distinctes des CipherSpecs et ont donc des noms différents. «CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for JMS», à la page 265 contient une table mappant les CipherSpecs pris en charge par IBM MQ à leurs CipherSuites équivalentes connues de JSSE. Pour plus d'informations sur les CipherSpecs et les CipherSuites avec IBM MQ, voir Sécurisation de IBM MQ.

Par exemple, pour configurer un objet ConnectionFactory pouvant être utilisé pour créer une connexion via un canal MQI activé par TLS avec une spécification CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA, exécutez la commande suivante pour JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Il peut également être défini à partir d'une application, à l'aide de la méthode setSSLCipherSuite () sur un objet MQConnectionFactory .

Par souci de commodité, si un CipherSpec est spécifié dans la propriété SSLCIPHERSUITE, JMSAdmin tente de mapper le CipherSpec à un CipherSuite approprié et émet un avertissement. Cette tentative de mappage n'est pas effectuée si la propriété est spécifiée par une application.

Vous pouvez également utiliser la table de définition de canal du client (CCDT). Pour plus d'informations, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290.

Propriété d'objet SSLFIPSREQUIRED
Si vous avez besoin d'une connexion pour utiliser une CipherSuite prise en charge par le fournisseur JSSE FIPS IBM Java (IBMJSSEFIPS), définissez la propriété SSLFIPSREQUIRED de la fabrique de connexions sur YES.

Remarque : Sous AIX, Linux, and Windows, IBM MQ fournit la conformité à la norme FIPS 140-2 via le module cryptographique IBM Crypto for C (ICC) . Le certificat de ce module a été déplacé vers le statut Historique. Les clients doivent afficher le certificatIBM Crypto for C (ICC) et prendre connaissance des

conseils fournis par le NIST. Un module FIPS 140-3 de remplacement est actuellement en cours et son statut peut être affiché en le recherchant dans la liste des modules NIST CMVP en cours de traitement.

IBM MQ Operator 3.2.0 et l'image de conteneur du gestionnaire de files d'attente à partir de la version 9.4.0.0 sont basés sur UBI 9. La conformité à la norme FIPS 140-3 est actuellement en attente et son statut peut être affiché en recherchant "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" dans la liste de processus des modules CMVP NIST.

La valeur par défaut de cette propriété est NO, ce qui signifie qu'une connexion peut utiliser n'importe quelle CipherSuite prise en charge par IBM MQ.

Si une application utilise plusieurs connexions, la valeur de SSLFIPSREQUIRED utilisée lorsque l'application crée la première connexion détermine la valeur utilisée lorsque l'application crée une connexion ultérieure. Cela signifie que la valeur de la propriété SSLFIPSREQUIRED de la fabrique de connexions utilisée pour créer une connexion ultérieure est ignorée. Vous devez redémarrer l'application si vous souhaitez utiliser une valeur différente de SSLFIPSREQUIRED.

Une application peut définir cette propriété en appelant la méthode setSSLFipsRequired () d'un objet ConnectionFactory . La propriété est ignorée si CipherSuite n'est pas défini.

Tâches associées

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client

Référence associée

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

propriété d'objet SSLPEERNAME
Utilisez SSLPEERNAME pour spécifier un modèle de nom distinctif afin de vous assurer que votre application JMS se connecte au gestionnaire de files d'attente approprié.

Une application JMS peut s'assurer qu'elle se connecte au gestionnaire de files d'attente approprié en spécifiant un modèle de nom distinctif (DN). La connexion aboutit uniquement si le gestionnaire de files d'attente présente un nom distinctif correspondant au modèle. Pour plus de détails sur le format de ce modèle, voir les rubriques connexes.

Le nom distinctif est défini à l'aide de la propriété SSLPEERNAME d'un objet ConnectionFactory . Par exemple, la commande JMSAdmin suivante définit un objet ConnectionFactory pour que le gestionnaire de files d'attente s'identifie avec un nom usuel commençant par les caractères `QMGR.` et avec au moins deux noms d'unité organisationnelle, dont le premier doit être IBM et le second WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

La vérification n'est pas sensible à la casse et les points-virgules peuvent être utilisés à la place des virgules. SSLPEERNAME peut également être défini à partir d'une application à l'aide de la méthode setSSLPeerName () sur un objet MQConnectionFactory . Si cette propriété n'est pas définie, aucune vérification n'est effectuée sur le nom distinctif fourni par le gestionnaire de files d'attente. Cette propriété est ignorée si CipherSuite n'est pas défini.

propriété d'objet SSLCERTSTORES
Utilisez SSLCERTSTORES pour spécifier une liste de serveurs LDAP à utiliser pour la vérification de la liste de révocation de certificat (CRL).

Il est courant d'utiliser une liste de révocation de certificat (CRL) pour identifier les certificats qui ne sont plus dignes de confiance. Les listes de révocation de certificat sont généralement hébergées sur des serveurs LDAP. JMS permet de spécifier un serveur LDAP pour la vérification de la liste de révocation de certificat sous Java 2 v1.4 ou ultérieure. L'exemple JMSAdmin suivant demande à JMS d'utiliser une CRL hébergée sur un serveur LDAP nommé crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Remarque : Pour utiliser un CertStore avec une CRL hébergée sur un serveur LDAP, assurez-vous que votre kit de développement de logiciels (SDK) Java est compatible avec la CRL. Certains logiciels SDK

exigent que la CRL soit conforme à la RFC 2587, qui définit un schéma pour LDAP v2. La plupart des serveurs LDAP v3 utilisent RFC 2256 à la place.

Si votre serveur LDAP n'est pas en cours d'exécution sur le port par défaut 389, vous pouvez spécifier le port en ajoutant un signe deux-points (:) et le numéro de port au nom d'hôte. Si le certificat présenté par le gestionnaire de files d'attente est présent dans la CRL hébergée sur crl1.ibm.com, la connexion n'est pas établie. Pour éviter un point de défaillance unique, JMS permet de fournir plusieurs serveurs LDAP en fournissant une liste de serveurs LDAP délimités par le caractère espace. Par exemple :

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

Lorsque plusieurs serveurs LDAP sont spécifiés, JMS tente chacun de leur côté jusqu'à ce qu'il trouve un serveur avec lequel il peut vérifier avec succès le certificat du gestionnaire de files d'attente. Chaque serveur doit contenir des informations identiques.

Une chaîne de ce format peut être fournie par une application dans la méthode MQConnectionFactory.setSSLCertStores (). L'application peut également créer un ou plusieurs objets java.security.cert.CertStore , les placer dans un objet Collection approprié et fournir cet objet Collection à la méthode setSSLCertStores (). De cette manière, l'application peut personnaliser la vérification de la liste de révocation de certificat. Consultez la documentation JSSE pour plus de détails sur la construction et l'utilisation des objets CertStore .

Le certificat présenté par le gestionnaire de files d'attente lors de la configuration d'une connexion est validé comme suit:

1. Le premier objet CertStore de la collection identifiée par les magasins sslCertest utilisé pour identifier un serveur CRL.
2. Une tentative est effectuée pour contacter le serveur CRL.
3. Si la tentative aboutit, le serveur recherche une correspondance pour le certificat.
   a. Si le certificat est révoqué, le processus de recherche est terminé et la demande de connexion échoue avec le code anomalie MQRC_SSL_CERTIFICATE_RÉVOQUÉ.
   b. Si le certificat est introuvable, le processus de recherche est terminé et la connexion est autorisée à continuer.
4. Si la tentative de contact du serveur échoue, l'objet CertStore suivant est utilisé pour identifier un serveur CRL et le processus se répète à l'étape 2.

   S'il s'agit du dernier CertStore de la collection ou si la collection ne contient aucun objet CertStore , le processus de recherche a échoué et la demande de connexion a échoué avec le code anomalie MQRC_SSL_CERT_STORE_ERROR.

L'objet Collection détermine l'ordre dans lequel les CertStores sont utilisés.

Si votre application utilise setSSLCertStores () pour définir une collection d'objets CertStore , MQConnectionFactory ne peut plus être lié à un espace de nom JNDI. Si vous tentez de le faire, une exception est générée. Si la propriété sslCertStores n'est pas définie, aucune vérification de révocation n'est effectuée sur le certificat fourni par le gestionnaire de files d'attente. Cette propriété est ignorée si CipherSuite n'est pas défini.

propriété d'objet SSLRESETCOUNT
Cette propriété représente le nombre total d'octets envoyés et reçus par une connexion avant que la clé secrète utilisée pour le chiffrement ne soit renégociée.

Le nombre d'octets envoyés est le nombre avant chiffrement et le nombre d'octets reçus est le nombre après déchiffrement. Le nombre d'octets inclut également les informations de contrôle envoyées et reçues par IBM MQ classes for JMS.

Par exemple, pour configurer un objet ConnectionFactory pouvant être utilisé pour créer une connexion via un canal MQI activé par TLS avec une clé secrète renégociée après la transmission de 4 Mo de données, exécutez la commande suivante à JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Une application peut définir cette propriété en appelant la méthode setSSLResetCount () d'un objet ConnectionFactory .

Si la valeur de cette propriété est zéro, qui est la valeur par défaut, la clé secrète n'est jamais renégociée. La propriété est ignorée si CipherSuite n'est pas défini.

Propriété d'objet SSLSocketFactory
Pour personnaliser d'autres aspects de la connexion TLS pour une application, créez une fabrique SSLSocketFactory et configurez JMS pour l'utiliser.

Vous pouvez personnaliser d'autres aspects de la connexion TLS pour une application. Par exemple, vous pouvez initialiser le matériel de cryptographie ou modifier le magasin de clés et le magasin de clés de confiance utilisés. Pour ce faire, l'application doit d'abord créer un objet javax.net.ssl.SSLSocketFactory personnalisé en conséquence. Consultez la documentation JSSE pour plus d'informations sur la procédure à suivre, car les fonctions personnalisables varient d'un fournisseur à l'autre. Une fois qu'un objet SSLSocketFactory approprié est obtenu, utilisez la méthode MQConnectionFactory.setSSLSocketFactory () pour configurer JMS afin d'utiliser l'objet SSLSocketFactory personnalisé.

Si votre application utilise la méthode setSSLSocketFactory () pour définir un objet SSLSocketFactory personnalisé, l'objet MQConnectionFactory ne peut plus être lié à un espace de nom JNDI. Si vous tentez de le faire, une exception est générée. Si cette propriété n'est pas définie, l'objet SSLSocketFactory par défaut est utilisé. Consultez la documentation JSSE pour plus de détails sur le comportement de l'objet SSLSocketFactory par défaut. Cette propriété est ignorée si CipherSuite n'est pas défini.

Important : Ne partez pas du principe que l'utilisation des propriétés SSL garantit la sécurité lorsqu'un objet ConnectionFactory est extrait d'un espace de nom JNDI qui n'est pas lui-même sécurisé. En particulier, l'implémentation LDAP standard de JNDI n'est pas sécurisée. Un attaquant peut imiter le serveur LDAP, induisant une application JMS à se connecter au mauvais serveur sans s'en apercevoir. Lorsque des dispositifs de sécurité appropriés sont en place, d'autres implémentations de JNDI (telles que l'implémentation fscontext) sont sécurisées.

Modification du magasin de clés ou du magasin de clés de confiance JSSE
Si vous apportez des modifications au magasin de clés ou au magasin de clés de confiance, vous devez effectuer certaines actions pour que les modifications soient prises en compte.

Si vous modifiez le contenu du magasin de clés ou du magasin de clés de confiance JSSE ou que vous modifiez l'emplacement du magasin de clés ou du magasin de clés de confiance, les applications IBM MQ classes for JMS qui s'exécutent à ce moment-là ne prennent pas automatiquement en compte les modifications. Pour que les modifications prennent effet, les actions suivantes doivent être effectuées:

- Les applications doivent fermer toutes leurs connexions et détruire toutes les connexions inutilisées dans les pools de connexions.
- Si votre fournisseur JSSE met en cache les informations du magasin de clés et du magasin de clés de confiance, ces informations doivent être actualisées.

Une fois ces actions effectuées, les applications peuvent recréer leurs connexions.

En fonction de la façon dont vous concevez vos applications et de la fonction fournie par votre fournisseur JSSE, il peut être possible d'effectuer ces actions sans arrêter ni redémarrer vos applications. Toutefois, l'arrêt et le redémarrage des applications peuvent être la solution la plus simple.

CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for JMS
La capacité des applications IBM MQ classes for JMS à établir des connexions à un gestionnaire de files d'attente dépend du CipherSpec spécifié à l'extrémité serveur du canal MQI et de la CipherSuite spécifiée à l'extrémité client.

Le tableau suivant répertorie les CipherSpecs pris en charge par IBM MQ et leurs CipherSuiteséquivalentes.

Deprecated Consultez la rubrique CipherSpecs CipherSpecs pour voir si des CipherSpecs, répertoriés dans le tableau suivant, ont été dépréciés par IBM MQ et, si tel est le cas, dans quelle mesure la mise à jour de CipherSpec a été dépréciée.

Important : Les CipherSuites répertoriées sont celles prises en charge par l'environnement d'exécution IBM Java (JRE) fourni avec IBM MQ. Les CipherSuites répertoriées incluent celles prises en charge par l'environnement d'exécution Java Oracle Java . Pour plus d'informations sur la configuration de votre application pour qu'elle utilise un environnement d'exécution Java Oracle Java , voir Configuration de votre application pour qu'elle utilise des mappages IBM Java ou Oracle Java CipherSuite.

Le tableau indique également le protocole utilisé pour la communication et indique si la CipherSuite est conforme à la norme FIPS 140-2.

Remarque : Sous AIX, Linux, and Windows, IBM MQ fournit la conformité à la norme FIPS 140-2 via le module cryptographique IBM Crypto for C (ICC) . Le certificat de ce module a été déplacé vers le statut Historique. Les clients doivent afficher le certificatIBM Crypto for C (ICC) et prendre connaissance des conseils fournis par le NIST. Un module FIPS 140-3 de remplacement est actuellement en cours et son statut peut être affiché en le recherchant dans la liste des modules NIST CMVP en cours de traitement.

IBM MQ Operator 3.2.0 et l'image de conteneur du gestionnaire de files d'attente à partir de la version 9.4.0.0 sont basés sur UBI 9. La conformité à la norme FIPS 140-3 est actuellement en attente et son statut peut être affiché en recherchant "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" dans la liste de processus des modules CMVP NIST.

Les suites de chiffrement désignées comme conformes à la norme FIPS 140-2 peuvent être utilisées si l'application n'a pas été configurée pour appliquer la conformité à la norme FIPS 140-2, mais si la conformité à la norme FIPS 140-2 a été configurée pour l'application (voir les remarques suivantes sur la configuration), seules les CipherSuites marquées comme compatibles à la norme FIPS 140-2 peuvent être configurées. La tentative d'utilisation d'autres CipherSuites génère une erreur.

Remarque : Chaque environnement d'exécution Java peut disposer de plusieurs fournisseurs de sécurité cryptographique, chacun d'eux pouvant fournir une implémentation de la même CipherSuite. Cependant, tous les fournisseurs de sécurité ne sont pas certifiés FIPS 140-2. Si la conformité à la norme FIPS 140-2 n'est pas appliquée pour une application, il est possible qu'une implémentation non certifiée de CipherSuite soit utilisée. Les implémentations non certifiées peuvent ne pas fonctionner conformément à la norme FIPS 140-2, même si CipherSuite répond théoriquement au niveau de sécurité minimal requis par la norme. Pour plus d'informations sur la configuration de la mise en application de la norme FIPS 140-2 dans les applications IBM MQ JMS , voir les remarques suivantes.

Pour plus d'informations sur la conformité FIPS 140-2 et Suite-B pour CipherSpecs et CipherSuites, voir Spécification de CipherSpecs. Vous devrez peut-être également prendre connaissance des informations concernant les Federal Information Processing Standardsaméricaines.

Pour utiliser l'ensemble complet des CipherSuites et pour utiliser la conformité FIPS 140-2 et/ou Suite-B certifiée, un environnement d'exécution Java approprié est requis. IBM Java 7 Service Refresh 4 Fix Pack 2 ou un niveau supérieur de IBM JRE fournit la prise en charge appropriée pour les suites de chiffrement TLS 1.2 CipherSuites répertoriées dans Tableau 41, à la page 267.

Pour pouvoir utiliser les chiffrements TLS 1.3 , l'environnement d'exécution Java qui exécute votre application doit prendre en charge TLS 1.3.

Remarque : Pour utiliser des CipherSuites, les fichiers de règles "sans restriction" doivent être configurés dans l'environnement d'exécution Java. Pour plus de détails sur la configuration des fichiers de règles dans un SDK ou un JRE, voir la rubrique *IBM SDK Policy files* dans le document *Security Reference for IBM SDK, Java Technology Edition* correspondant à la version que vous utilisez.

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	oui
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	non

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	non
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	oui

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	oui
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	non

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	non
TLS_RSA_WITH_3DES_EDE_CBC_SHA «2», à la page 285	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	non «4», à la page 285

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	non «4», à la page 285
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	non «4», à la page 285

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	non «4», à la page 285
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	non «4», à la page 285

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	non «4», à la page 285
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	non «4», à la page 285

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	non
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	non
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	non

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	oui
TLS_AES_128_GCM_SHA256 «3», à la page 285	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS v1.3	non
TLS_AES_256_GCM_SHA384 «3», à la page 285	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS v1.3	non

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_CHACHA20_POLY1305_SHA256 «3», à la page 285	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS v1.3	non
TLS_AES_128_CCM_SHA256 «3», à la page 285	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS v1.3	non

Tableau 41. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 285	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_AES_128_CCM_8_SHA256 «3», à la page 285	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS v1.3	non
Tous «3», à la page 285	*ANY	*ANY	Multiple	non
ANY_TLS13 «3», à la page 285	*TLS13	*TLS13	TLS V13	non
ANY_TLS12_OR_HIGHER «3», à la page 285	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 et versions ultérieures	non
ANY_TLS13_OR_HIGHER «3», à la page 285	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 et versions ultérieures	non

Remarques :

1. Il s'agit de la valeur configurée sur un canal dans IBM MQ, y compris dans une table de définition de canal du client (CCDT) (binaire ou JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA est obsolète. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.
3. Pour pouvoir utiliser les chiffrements TLS v1.3 , l'environnement d'exécution Java (Java runtime environment) exécutant votre application doit prendre en charge TLS v1.3.
4. V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, l'environnement d'exécution Java de IBM Java 8 supprime la prise en charge de l'échange de clés RSA en mode FIPS.

Configuration des suites de chiffrement et de la conformité FIPS dans une application IBM MQ classes for JMS

- Une application qui utilise IBM MQ classes for JMS peut utiliser l'une des deux méthodes suivantes pour définir CipherSuite pour une connexion:
  - Appelez la méthode setSSLCipherSuite d'un objet ConnectionFactory .
  - Utilisez l'outil d'administration IBM MQ JMS pour définir la propriété SSLCIPHERSUITE d'un objet ConnectionFactory .
- Une application qui utilise IBM MQ classes for JMS peut utiliser l'une des deux méthodes suivantes pour appliquer la conformité à la norme FIPS 140-2:
  - Appelez la méthode setSSLFipsRequired d'un objet ConnectionFactory .
  - Utilisez l'outil d'administration IBM MQ JMS pour définir la propriété SSLFIPSREQUIRED d'un objet ConnectionFactory .

Configuration de votre application pour l'utilisation des mappages IBM Java ou Oracle Java CipherSuite

V 9.4.0 Depuis la IBM MQ 9.4.0, un chiffrement peut être défini en tant que nom CipherSpec ou CipherSuite et est géré correctement par IBM MQ.

Remarque : Removed La Java propriété système `com.ibm.mq.cfg.useIBMCipherMappings`, qui contrôlait les mappages utilisés dans les versions antérieures de IBM MQ, n'est plus nécessaire et est supprimée du produit à l'adresse IBM MQ 9.4.0.

Limitations de l'interopérabilité

Certaines CipherSuites peuvent être compatibles avec plusieurs IBM MQ CipherSpec, selon le protocole utilisé. Toutefois, seule la combinaison CipherSuite/CipherSpec qui utilise la version TLS spécifiée dans le tableau 1 est prise en charge. La tentative d'utilisation des combinaisons non prises en charge de CipherSuites et CipherSpecs échouera avec une exception appropriée. Les installations utilisant l'une de ces combinaisons CipherSuite/CipherSpec doivent être déplacées vers une combinaison prise en charge.

Le tableau suivant présente les CipherSuites auxquelles cette limitation s'applique.

Tableau 42. CipherSuites et leurs CipherSpecs pris en charge et non pris en charge

CipherSuite	TLS CipherSpec pris en charge	CipherSpec SSL non pris en charge
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA «1», à la page 286	TRIPLE_DES_SHA_US

Tableau 42. CipherSuites et leurs CipherSpecs pris en charge et non pris en charge (suite)		
CipherSuite	**TLS CipherSpec pris en charge**	**CipherSpec SSL non pris en charge**
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Remarque :

1. Deprecated Ce CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA est obsolète. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.

Ecriture des exits de canal dans Java for IBM MQ classes for JMS
Vous créez des exits de canal en définissant des classes Java qui implémentent des interfaces spécifiées.

Pour une introduction aux exits de sécurité, commencez par la rubrique Programmes d'exit de sécurité de canal .

Trois interfaces sont définies dans le package com.ibm.mq.exits :

- WMQSendExit, pour un exit d'émission
- WMQReceiveExit, pour un exit de réception
- WMQSecurityExit, pour un exit de sécurité

L'exemple de code suivant définit une classe qui implémente les trois 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
  }
}
```

Chaque exit reçoit en tant que paramètres un objet MQCXP et un objet MQCD. Ces objets représentent les structures MQCXP et MQCD définies dans l'interface de procédure.

Lorsqu'un exit d'émission est appelé, le paramètre agentBuffer contient les données qui sont sur le point d'être envoyées au gestionnaire de files d'attente du serveur. Un paramètre de longueur n'est pas requis car l'expression agentBuffer.limit () fournit la longueur des données. L'exit d'émission renvoie comme valeur les données à envoyer au gestionnaire de files d'attente du serveur. Toutefois, si l'exit d'émission n'est pas le dernier exit d'émission d'une séquence d'exits d'émission, les données renvoyées

sont transmises à l'exit d'émission suivant de la séquence. Un exit d'émission peut renvoyer une version modifiée des données qu'il reçoit dans le paramètre agentBuffer ou renvoyer les données inchangées. Le corps de sortie le plus simple possible est donc:

```
{ return agentBuffer; }
```

Lorsqu'un exit de réception est appelé, le paramètre agentBuffer contient les données qui ont été reçues du gestionnaire de files d'attente du serveur. L'exit de réception renvoie comme valeur les données à transmettre à l'application par IBM MQ classes for JMS. Toutefois, si l'exit de réception n'est pas le dernier exit de réception dans une séquence d'exits de réception, les données renvoyées sont transmises à l'exit de réception suivant dans la séquence.

Lorsqu'un exit de sécurité est appelé, le paramètre agentBuffer contient les données qui ont été reçues dans un flux de sécurité de l'exit de sécurité à l'extrémité serveur de la connexion. L'exit de sécurité renvoie comme valeur les données à envoyer dans un flux de sécurité à l'exit de sécurité du serveur.

Les exits de canal sont appelés avec une mémoire tampon comportant un tableau de sauvegarde. Pour de meilleures performances, l'exit doit renvoyer une mémoire tampon avec un tableau de sauvegarde.

Jusqu'à 32 caractères de données utilisateur peuvent être transmis à un exit de canal lorsqu'il est appelé. L'exit accède aux données utilisateur en appelant la méthode getExitData () de l'objet MQCXP. Bien que l'exit puisse modifier les données utilisateur en appelant la méthode setExitData (), les données utilisateur sont actualisées chaque fois que l'exit est appelé. Toute modification apportée aux données utilisateur est donc perdue. Toutefois, l'exit peut transmettre des données d'un appel à l'autre à l'aide de la zone utilisateur de l'exit de l'objet MQCXP. L'exit accède à la zone utilisateur de l'exit par référence en appelant la méthode getExitUserArea().

Chaque classe d'exit doit avoir un constructeur. Le constructeur peut être soit le constructeur par défaut, comme illustré dans l'exemple précédent, soit un constructeur avec un paramètre de chaîne. Le constructeur est appelé pour créer une instance de la classe d'exit pour chaque exit défini dans la classe. Par conséquent, dans l'exemple précédent, une instance de la classe MyMQExits est créée pour l'exit d'émission, une autre instance est créée pour l'exit de réception et une troisième instance est créée pour l'exit de sécurité. Lorsqu'un constructeur avec un paramètre de chaîne est appelé, le paramètre contient les mêmes données utilisateur qui sont transmises à l'exit de canal pour lequel l'instance est créée. Si une classe d'exit possède à la fois un constructeur par défaut et un constructeur à un seul paramètre, le constructeur à un seul paramètre est prioritaire.

Ne fermez pas la connexion à partir d'un exit de canal.

Lorsque des données sont envoyées à l'extrémité serveur d'une connexion, le chiffrement TLS est effectué *après* l'appel des exits de canal. De même, lorsque des données sont reçues de l'extrémité serveur d'une connexion, le déchiffrement TLS est effectué *avant* l'appel des exits de canal.

Dans les versions de IBM MQ classes for JMS antérieures à IBM WebSphere MQ 7.0, les exits de canal étaient implémentés à l'aide des interfaces MQSendExit, MQReceiveExitet MQSecurityExit. Vous pouvez toujours utiliser ces interfaces, mais les nouvelles interfaces sont préférées pour améliorer les fonctions et les performances.

Configuration de IBM MQ classes for JMS pour l'utilisation des exits de canal
Une application IBM MQ classes for JMS peut utiliser des exits de sécurité de canal, d'envoi et de réception sur le canal MQI qui démarre lorsque l'application se connecte à un gestionnaire de files d'attente. L'application peut utiliser des exits écrits en Java, C ou C + +. L'application peut également utiliser une séquence d'exits d'émission ou de réception exécutés successivement.

Les propriétés suivantes sont utilisées pour spécifier un exit d'émission ou une séquence d'exits d'émission utilisés par une connexion JMS :

- Propriété **SENDEXIT** d'un objet MQConnectionFactory .
- La propriété **sendexit** sur une spécification d'activation utilisée par l'adaptateur de ressources IBM MQ pour les communications entrantes,

- La propriété **sendexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM MQ pour la communication de sortie.

La valeur de la propriété est une chaîne qui comprend un ou plusieurs éléments séparés par des virgules. Chaque élément identifie un exit d'émission de l'une des manières suivantes:

- Nom d'une classe qui implémente l'interface `WMQSendExit` pour un exit d'émission écrit en Java.
- Chaîne au format *libraryName* (*entryPoint*) pour un exit d'émission écrit en C ou C + +.

De la même manière, les propriétés suivantes spécifient l'exit de réception, ou la séquence d'exits de réception, utilisés par une connexion:

- Propriété **RECEXIT** d'un objet MQConnectionFactory .
- La propriété **receiveexit** sur une spécification d'activation utilisée par l'adaptateur de ressources IBM MQ pour les communications entrantes,
- La propriété **receiveexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM MQ pour la communication de sortie.

Les propriétés suivantes spécifient l'exit de sécurité utilisé par une connexion:

- Propriété **SECEXIT** d'un objet MQConnectionFactory .
- La propriété **securityexit** sur une spécification d'activation utilisée par l'adaptateur de ressources IBM MQ pour les communications entrantes,
- La propriété **securityexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM MQ pour la communication de sortie.

Pour MQConnectionFactories, vous pouvez définir les propriétés **SENDEXIT**, **RECEXIT** et **SECEXIT** à l'aide de l'outil d'administration IBM MQ JMS ou de IBM MQ Explorer. Une application peut également définir les propriétés en appelant les méthodes `setSendExit()`, `setReceiveExit()`et `setSecurityExit()` .

Les exits de canal sont chargés par leur propre chargeur de classe. Pour trouver un exit de canal, le chargeur de classe recherche les emplacements suivants dans l'ordre indiqué.

1. Chemin d'accès aux classes spécifié par la propriété **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** ou par l'attribut **JavaExitsClassPath** dans la strophe Channels du fichier de configuration du client IBM MQ .
2. Deprecated Chemin d'accès aux classes spécifié par la propriété système Java **com.ibm.mq.exitClasspath**. Notez que cette propriété est désormais obsolète.
3. Le répertoire d'exit IBM MQ , comme illustré dans la Tableau 43, à la page 288. Le chargeur de classe recherche d'abord dans le répertoire les fichiers de classe qui ne sont pas conditionnés dans des fichiers d'archive Java (JAR). Si l'exit de canal est introuvable, le chargeur de classe recherche ensuite les fichiers JAR dans le répertoire.

Tableau 43. Répertoire des exits IBM MQ

Plateforme	Répertoire
Linux AIX AIX and Linux	`/var/mqm/exits` (exits de canal 32 bits) `/var/mqm/exits64` (exits de canal 64 bits)
Windows Windows	*rép_données_installation*\exits où *rép_données_installation* est le répertoire que vous avez choisi pour les fichiers de données IBM MQ lors de l'installation. Le répertoire par défaut est C:\ProgramData\IBM\MQ.

Remarque : Si un exit de canal existe dans plusieurs emplacements, IBM MQ classes for JMS charge la première instance qu'il trouve.

Le parent du chargeur de classe est le chargeur de classe utilisé pour charger IBM MQ classes for JMS. Il est donc possible pour le chargeur de classe parent de charger un exit de canal s'il est introuvable dans l'un des emplacements précédents. Toutefois, lorsque vous utilisez IBM MQ classes for JMS dans un environnement tel qu'un serveur d'applications JEE , il est peu probable que vous puissiez influencer le choix du chargeur de classe parent. Par conséquent, le chargeur de classe doit être configuré en définissant la Java propriété système **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** sur le serveur d'applications.

Si votre application est exécutée avec Java security manager activé, le fichier de configuration de règles utilisé par l'environnement d'exécution Java dans lequel l'application s'exécute doit disposer des droits permettant de charger une classe d'exit de canal. Pour plus d'informations sur la procédure à suivre, voir Exécution des classes IBM MQ pour les applications JMS sous Java Security Manager.

Les interfaces `MQSendExit`, `MQReceiveExit`et `MQSecurityExit` fournies avec des versions antérieures à IBM WebSphere MQ 7.0 sont toujours prises en charge. Si vous utilisez des exits de canal qui implémentent ces interfaces, `com.ibm.mq.jar` doit être présent dans le chemin d'accès aux classes.

Pour plus d'informations sur l'écriture des exits de canal en C, voir «Programmes d'exit de canal pour les canaux de messagerie», à la page 988. Vous devez stocker les programmes d'exit de canal écrits en C ou C++ dans le répertoire indiqué dans Tableau 43, à la page 288.

Si votre application utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290.

Spécification des données utilisateur à transmettre aux exits de canal lors de l'utilisation de IBM MQ classes for JMS
Jusqu'à 32 caractères de données utilisateur peuvent être transmis à un exit de canal lorsqu'il est appelé.

La propriété SENDEXITINIT d'un objet MQConnectionFactory spécifie les données utilisateur qui sont transmises à chaque exit d'émission lorsqu'il est appelé. La valeur de la propriété est une chaîne qui comprend un ou plusieurs éléments de données utilisateur séparés par des virgules. La position de chaque élément de données utilisateur dans la chaîne détermine à quel exit d'émission, dans une séquence d'exits d'émission, les données utilisateur sont transmises. Par exemple, le premier élément de données utilisateur de la chaîne est transmis au premier exit d'émission dans une séquence d'exits d'émission.

Vous pouvez définir la propriété SENDEXITINIT à l'aide de l'outil d'administration IBM MQ JMS ou de IBM MQ Explorer. Une application peut également définir la propriété en appelant la méthode setSendExitInit().

De la même manière, la propriété RECEXITINIT d'un objet ConnectionFactory indique les données utilisateur qui sont transmises à chaque exit de réception et la propriété SECEXITINIT indique les données utilisateur transmises à un exit de sécurité. Vous pouvez définir ces propriétés à l'aide de l'outil d'administration IBM MQ JMS ou de IBM MQ Explorer. Une application peut également définir les propriétés en appelant les méthodes setReceiveExitInit() et setSecurityExitInit().

Notez les règles suivantes lors de la spécification des données utilisateur qui sont transmises aux exits de canal:

- Si le nombre d'éléments de données utilisateur dans une chaîne est supérieur au nombre d'exits dans une séquence, les éléments excédentaires de données utilisateur sont ignorés.
- Si le nombre d'éléments de données utilisateur dans une chaîne est inférieur au nombre d'exits dans une séquence, chaque élément non spécifié de données utilisateur est défini sur une chaîne vide. Deux virgules consécutives dans une chaîne, ou une virgule au début d'une chaîne, indiquent également un élément non spécifié de données utilisateur.

Si une application utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, toutes les données utilisateur spécifiées dans une définition de canal de connexion client sont transmises aux exits de canal lorsqu'ils sont appelés. Pour plus d'informations sur l'utilisation d'une table de définition de canal du client, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290.

Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS
Une application IBM MQ classes for JMS peut utiliser des définitions de canal de connexion client stockées dans une table de définition de canal du client (CCDT). Vous configurez un objet ConnectionFactory pour utiliser la table de définition de canal du client. Son utilisation est soumise à certaines restrictions.

Au lieu de créer une définition de canal de connexion client en définissant certaines propriétés d'un objet ConnectionFactory , une application IBM MQ classes for JMS peut utiliser des définitions de canal de connexion client stockées dans une table de définition de canal client. Ces définitions sont créées par des commandes IBM MQ Script (MQSC) ou des commandes IBM MQ Programmable Command Format (PCF). Lorsque l'application crée un objet Connection, IBM MQ classes for JMS recherche dans la table de définition de canal du client une définition de canal de connexion client appropriée et utilise la définition de canal pour démarrer un canal MQI. Pour plus d'informations sur les tables de définition de canal du client et sur la façon d'en créer une, voir Table de définition de canal du client.

Pour utiliser une table de définition de canal du client, la propriété CCDTURL d'un objet ConnectionFactory doit être définie sur un objet URL. IBM MQ classes for JMS ne lisez pas les informations relatives à la table de définition de canal du client à partir du fichier de configuration IBM MQ MQI client , bien que d'autres valeurs soient utilisées à partir de là (voir «Le fichier de configuration IBM MQ classes for JMS/Jakarta Messaging», à la page 102 pour laquelle la valeur s'applique). L'objet URL encapsule un URL qui identifie le nom et l'emplacement du fichier contenant la table de définition de canal du client et indique comment accéder au fichier. Vous pouvez définir la propriété CCDTURL à l'aide de l'outil d'administration IBM MQ JMS ou une application peut définir la propriété en créant un objet URL et en appelant la méthode setCCDTURL() de l'objet ConnectionFactory .

Par exemple, si le fichier ccdt1.tab contient une table de définition de canal du client et qu'il est stocké sur le même système que celui sur lequel l'application est en cours d'exécution, l'application peut définir la propriété CCDTURL de la manière suivante:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Comme autre exemple, supposons que le fichier ccdt2.tab contienne une table de définition de canal du client et qu'il soit stocké sur un système différent de celui sur lequel l'application s'exécute. Si le fichier est accessible à l'aide du protocole FTP, l'application peut définir la propriété CCDTURL de la manière suivante:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

Outre la définition de la propriété CCDTURL de l'objet ConnectionFactory , la propriété QMANAGER du même objet doit avoir l'une des valeurs suivantes:

- Nom d'un gestionnaire de files d'attente
- Un astérisque (*) suivi du nom d'un groupe de gestionnaires de files d'attente

Ces valeurs sont les mêmes que celles qui peuvent être utilisées pour le paramètre **QMgrName** sur un appel MQCONN émis par une application client qui utilise l'interface MQI (Message Queue Interface). Pour plus d'informations sur la signification de ces valeurs, voir MQCONN. Vous pouvez définir la propriété QMANAGER à l'aide de l'outil d'administration IBM MQ JMS ou de l'explorateur IBM MQ . Une application peut également définir la propriété en appelant la méthode setQueueManager () de l'objet ConnectionFactory .

Si une application crée ensuite un objet Connexion à partir de l'objet ConnectionFactory , IBM MQ classes for JMS accède à la table de définition de canal du client identifiée par la propriété CCDTURL, utilise la propriété QMANAGER pour rechercher dans la table une définition de canal de connexion client appropriée, puis utilise la définition de canal pour démarrer un canal MQI vers un gestionnaire de files d'attente.

Notez que les propriétés CCDTURL et CHANNEL d'un objet ConnectionFactory ne peuvent pas être définies toutes les deux lorsque l'application appelle la méthode createConnection(). Si les deux

propriétés sont définies, la méthode émet une exception. La propriété CCDTURL ou CHANNEL est considérée comme étant définie si sa valeur est autre que null, une chaîne vide ou une chaîne contenant tous les caractères blancs.

Lorsque IBM MQ classes for JMS trouve une définition de canal de connexion client appropriée dans la table de définition de canal du client, il utilise uniquement les informations extraites de la table pour démarrer un canal MQI. Toutes les propriétés liées aux canaux de l'objet ConnectionFactory sont ignorées.

En particulier, notez les points suivants si vous utilisez TLS:

- Un canal MQI utilise TLS uniquement si la définition de canal extraite de la table de définition de canal du client indique le nom d'un CipherSpec pris en charge par IBM MQ classes for JMS.
- Une table de définition de canal du client contient également des informations sur l'emplacement des serveurs LDAP (Lightweight Directory Access Protocol) qui contiennent des listes de révocation de certificats (CRL). IBM MQ classes for JMS utilise uniquement ces informations pour accéder aux serveurs LDAP qui contiennent des listes de révocation de certificat.
- Une table de définition de canal du client peut également contenir l'emplacement d'un répondeur OCSP. IBM MQ classes for JMS ne peut pas utiliser les informations OCSP dans un fichier de table de définition de canal du client. Toutefois, vous pouvez configurer OCSP comme décrit dans la section Online Certificate Status Protocol (OCSP) dans les applications client Java et JMS.

Pour plus d'informations sur l'utilisation de TLS avec une table de définition de canal du client, voir Utilisation du client transactionnel étendu avec des canaux TLS.

Notez également les points suivants si vous utilisez des exits de canal:

- Un canal MQI utilise uniquement les exits de canal et les données utilisateur associées spécifiées par la définition de canal extraite de la table de définition de canal du client.
- Une définition de canal extraite d'une table de définitions de canal du client peut spécifier des exits de canal écrits en Java. Cela signifie, par exemple, que le paramètre SCYEXIT de la commande DEFINE CHANNEL pour créer une définition de canal de connexion client peut spécifier le nom d'une classe qui implémente l'interface WMQSecurityExit . De même, le paramètre SENDEXIT peut indiquer le nom d'une classe qui implémente l'interface WMQSendExit et le paramètre RCVEXIT peut indiquer le nom d'une classe qui implémente l'interface WMQReceiveExit . Pour plus d'informations sur l'écriture d'un exit de canal dans Java, voir«Ecriture des exits de canal dans Java for IBM MQ classes for JMS», à la page 286.

  L'utilisation d'exits de canal écrits dans un langage autre que Java est également prise en charge. Pour plus d'informations sur la spécification des paramètres SCYEXIT, SENDEXIT et RCVEXIT dans la commande DEFINE CHANNEL pour les exits de canal écrits dans un autre langage, voir DEFINE CHANNEL.

Reconnexion automatique du client JMS
Configurez votre client JMS pour qu'il se reconnecte automatiquement à la suite d'une défaillance du réseau, du gestionnaire de files d'attente ou du serveur.

Normalement, si une application IBM MQ classes for JMS autonome est connectée à un gestionnaire de files d'attente à l'aide du transport client et que le gestionnaire de files d'attente devient indisponible pour une raison quelconque (en raison d'une indisponibilité du réseau, d'une défaillance du gestionnaire de files d'attente ou de l'arrêt du gestionnaire de files d'attente, par exemple), IBM MQ classes for JMS émet une exception JMSException la prochaine fois que l'application tente de communiquer avec le gestionnaire de files d'attente. L'application doit intercepter l'exception JMSException et tenter de se reconnecter au gestionnaire de files d'attente. Vous pouvez simplifier la conception de l'application en activant la reconnexion automatique du client. Lorsque le gestionnaire de files d'attente devient indisponible, IBM MQ classes for JMS tente de se reconnecter automatiquement au gestionnaire de files d'attente pour le compte de l'application. Cela signifie que l'application n'a pas besoin de contenir de logique pour se reconnecter.

L'utilisation de cette implémentation de la reconnexion automatique du client n'est pas prise en charge dans les serveurs d'applications Java Platform, Enterprise Edition . Pour une implémentation alternative,

voir «Utilisation de la reconnexion automatique du client dans les environnements Java EE», à la page 298 .

Utilisation de la reconnexion automatique du client JMS
Si une application IBM MQ classes for JMS autonome utilise une fabrique de connexions pour laquelle la propriété CONNECTIONNAMELIST ou CCDTURL est définie, l'application peut utiliser la reconnexion client automatique.

La reconnexion automatique du client peut être utilisée pour se reconnecter aux gestionnaires de files d'attente, y compris ceux qui font partie d'une configuration à haute disponibilité. Les configurations à haute disponibilité incluent des gestionnaires de files d'attente multi-instance, des gestionnaires de files d'attente de données répliquées (RDQM) ou des gestionnaires de files d'attente à haute disponibilité sur un dispositif IBM MQ .

Le comportement de la fonctionnalité de reconnexion automatique du client fournie par IBM MQ classes for JMS dépend des propriétés suivantes:

La propriété de fabrique de connexions JMS TRANSPORT (nom abrégé TRAN)

TRANSPORT indique comment les applications qui utilisent la fabrique de connexions se connectent à un gestionnaire de files d'attente. Cette propriété doit être définie sur la valeur CLIENT pour que la reconnexion automatique du client soit utilisée. La reconnexion automatique du client n'est pas disponible pour les applications qui se connectent à un gestionnaire de files d'attente qui utilise une fabrique de connexions dont la propriété TRANSPORT est définie sur BIND, DIRECT ou DIRECTIVE THTTP.

La propriété de fabrique de connexions JMS QMANAGER (nom abrégé QMGR)

La propriété QMANAGER indique le nom du gestionnaire de files d'attente auquel la fabrique de connexions se connecte.

La propriété de fabrique de connexions JMS CONNECTIONNAMELIST (nom abrégé CRHOSTS)

La propriété CONNECTIONNAMELIST est une liste séparée par des virgules, dans laquelle chaque entrée contient des informations sur le nom d'hôte et le port à utiliser pour se connecter au gestionnaire de files d'attente spécifié par la propriété QMANAGER lorsque vous utilisez le transport CLIENT. Le format de la liste est le suivant: nom d'hôte (port), nom d'hôte (port).

Propriété de fabrique de connexions JMS CCDTURL (Short name CCDT)

La propriété CCDTURL pointe vers la table de définition de canal du client que le IBM MQ classes for JMS utilise lorsqu'il se connecte à un gestionnaire de files d'attente à l'aide d'une table de définition de canal du client.

Propriété de la fabrique de connexions JMS CLIENTRECONNECTOPTIONS (Short name CROPT)

CLIENTRECONNECTOPTIONS contrôle si IBM MQ classes for JMS tente de se connecter automatiquement à un gestionnaire de files d'attente pour le compte d'une application si un gestionnaire de files d'attente devient disponible.

Attribut DefRecon dans la strophe Channels du fichier de configuration client

L'attribut DefRecon fournit une option d'administration permettant à toutes les applications de se reconnecter automatiquement ou de désactiver la reconnexion automatique pour les applications qui sont écrites pour se reconnecter automatiquement.

La reconnexion automatique du client n'est disponible que lorsqu'une application parvient à se connecter à un gestionnaire de files d'attente.

Lorsqu'une application se connecte à un gestionnaire de files d'attente qui utilise le transport CLIENT, IBM MQ classes for JMS utilise la valeur de la propriété de fabrique de connexions CLIENTRECONNECTOPTIONS pour déterminer s'il convient d'utiliser la reconnexion automatique du client si le gestionnaire de files d'attente auquel l'application est connectée devient indisponible.
Le tableau 1 présente les valeurs possibles pour la propriété CLIENTRECONNECTOPTIONS et le comportement de IBM MQ classes for JMS pour chacune de ces valeurs:

Tableau 44. Valeurs possibles de la propriété CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Comportement de IBM MQ classes for JMS**
ANY	Si CONNECTIONNAMELIST est défini, utilisez la valeur de la propriété CONNECTIONNAMELIST pour ouvrir une connexion à une combinaison de nom d'hôte et de port et pour vous connecter à un gestionnaire de files d'attente. Pour utiliser cette option de reconnexion automatique du client, la propriété QMANAGER doit être définie sur la valeur par défaut ou sur "*". Si CCDTURL est défini, ouvrez la table de définition de canal du client spécifiée par la propriété CCDTURL, sélectionnez une entrée dans la table, puis utilisez cette entrée pour démarrer un canal de connexion client vers un gestionnaire de files d'attente. Pour utiliser cette option de reconnexion automatique du client, la propriété QMANAGER doit être définie sur l'une des valeurs suivantes: • Un astérisque (*) • Un astérisque (*) suivi du nom d'un groupe de gestionnaires de files d'attente • Une chaîne vide ou une chaîne contenant tous les caractères vides
ADEF	Utilisez la valeur de DefRecon pour déterminer si la reconnexion automatique du client est disponible.
DESACTIVE	N'effectuez aucune reconnexion automatique du client et renvoyez une exception JMSException à l'application.
QMGR	Indique que le client doit se reconnecter au même gestionnaire de files d'attente. Cette option doit être utilisée pour les solutions à haute disponibilité, où la reconnexion à une autre instance du même gestionnaire de files d'attente est requise. Si CONNECTIONNAMELIST est défini, utilisez la valeur de la propriété CONNECTIONNAMELIST pour ouvrir une connexion à une combinaison de nom d'hôte et de port, et connectez-vous au gestionnaire de files d'attente spécifié par la propriété QMANAGER. Si CCDTURL est défini, ouvrez la table de définition de canal du client spécifiée par la propriété CCDTURL, recherchez les entrées de la table qui correspondent au nom de gestionnaire de files d'attente spécifié par la propriété QMANAGER, puis utilisez ces entrées pour démarrer un canal de connexion client vers ce gestionnaire de files d'attente.

Si CONNECTIONNAMELIST est défini, lorsque vous effectuez une reconnexion automatique du client, IBM MQ classes for JMS utilise les informations de la propriété de fabrique de connexions CONNNECTIONNAMELIST pour déterminer le système auquel se reconnecter.

IBM MQ classes for JMS tente initialement de se reconnecter à l'aide du nom d'hôte et du port spécifiés dans la première entrée de la commande CONNECTIONNAMELIST. Si une connexion est établie, IBM MQ classes for JMS tente de se connecter au gestionnaire de files d'attente dont le nom est spécifié dans la propriété QMANAGER. Si une connexion au gestionnaire de files d'attente peut être établie, le IBM MQ classes for JMS rouvre tous les objets IBM MQ que l'application avait ouverts avant la reconnexion automatique du client et continue de s'exécuter comme auparavant.

Si une connexion ne peut pas être établie au gestionnaire de files d'attente requis à l'aide de la première entrée de la commande CONNECTIONNAMELIST, IBM MQ classes for JMS tente la deuxième entrée de la commande CONNECTIONNAMELIST, etc.

Une fois que le IBM MQ classes for JMS a essayé toutes les entrées de la commande CONNECTIONNAMELIST, il attend un certain temps avant de tenter de se reconnecter. Pour effectuer la nouvelle tentative de reconnexion, IBM MQ classes for JMS commence par la première entrée de la commande CONNECTIONNAMELIST. Ils tentent ensuite à tour de rôle chaque entrée de la commande CONNECTIONNAMELIST jusqu'à ce qu'une reconnexion se produise ou que la fin de la commande CONNECTIONNAMELIST soit atteinte, ce qui permet à IBM MQ classes for JMS d'attendre un certain temps avant d'effectuer une nouvelle tentative.

Si CCDTURL est défini, lors de la reconnexion automatique du client, IBM MQ classes for JMS utilise la table de définition de canal du client spécifiée dans la propriété CCDTURL pour déterminer à quel système se reconnecter.

IBM MQ classes for JMS analyse initialement la table de définition de canal du client et trouve une entrée appropriée qui correspond à la valeur de la propriété QMANAGER. Lorsqu'une entrée est trouvée, IBM MQ classes for JMS tente de se reconnecter au gestionnaire de files d'attente requis à l'aide de cette entrée. Si une connexion au gestionnaire de files d'attente peut être établie, le IBM MQ classes for JMS rouvre tous les objets IBM MQ que l'application avait ouverts avant la reconnexion automatique du client et continue de s'exécuter comme auparavant.

Si une connexion ne peut pas être établie au gestionnaire de files d'attente requis, IBM MQ classes for JMS recherche une autre entrée appropriée dans la table de définition de canal du client et tente de l'utiliser, etc.

Une fois que le IBM MQ classes for JMS a essayé toutes les entrées appropriées dans la table de définition de canal du client, il attend un certain temps avant de tenter de se reconnecter à nouveau. Pour effectuer la nouvelle tentative de reconnexion, IBM MQ classes for JMS analyse à nouveau la table de définition de canal du client et tente la première entrée appropriée. Ils essaieront ensuite chaque entrée appropriée dans la table de définition de canal du client jusqu'à ce qu'une reconnexion se produise ou que la dernière entrée appropriée dans la table de définition de canal du client ait été tentée. IBM MQ classes for JMS attend alors un certain temps avant d'effectuer une nouvelle tentative.

Si vous utilisez CONNECTIONNAMELIST ou CCDTURL, le processus de reconnexion automatique du client se poursuit jusqu'à ce que le IBM MQ classes for JMS se reconnecte au gestionnaire de files d'attente spécifié par la propriété QMANAGER.

Par défaut, les tentatives de reconnexion se produisent aux intervalles suivants:

- La première tentative est effectuée après un délai initial de 1 seconde, plus un aléa jusqu'à 250 millisecondes.
- La seconde tentative est effectuée 2 secondes, plus un intervalle aléatoire pouvant atteindre 500 millisecondes, après l'échec de la première tentative.
- La troisième tentative est effectuée 4 secondes, plus un intervalle aléatoire pouvant atteindre 1 seconde, après l'échec de la deuxième tentative.
- La quatrième tentative est effectuée 8 secondes, plus un intervalle aléatoire pouvant atteindre 2 secondes, après l'échec de la troisième tentative.

- La cinquième tentative est effectuée 16 secondes, plus un intervalle aléatoire pouvant aller jusqu'à 4 secondes, après l'échec de la quatrième tentative.
- La sixième tentative et toutes les tentatives suivantes sont effectuées 25 secondes, plus un intervalle aléatoire pouvant atteindre 6 secondes et 250 millisecondes après l'échec de la tentative précédente.

Les tentatives de reconnexion sont retardées par des intervalles partiellement fixes et partiellement aléatoires. Cela permet d'empêcher toutes les applications IBM MQ classes for JMS qui étaient connectées à un gestionnaire de files d'attente qui n'est plus disponible de se reconnecter simultanément.

Si vous devez augmenter les valeurs par défaut afin de refléter plus précisément le temps nécessaire à la récupération d'un gestionnaire de files d'attente ou à l'activation d'un gestionnaire de files d'attente de secours, modifiez l'attribut ReconDelay dans la strophe Channel du fichier de configuration du client. Pour plus d'informations, voir Strophe CHANNELS du fichier de configuration du client.

Le fait qu'une application IBM MQ classes for JMS continue de fonctionner correctement après avoir été reconnectée automatiquement dépend de sa conception. Lisez les rubriques connexes pour savoir comment concevoir des applications pouvant utiliser la fonctionnalité de reconnexion automatique.

Codes anomalie indiquant qu'un gestionnaire de files d'attente n'est plus disponible
Les codes anomalie indiquent qu'un gestionnaire de files d'attente n'est plus disponible ou qu'il est inaccessible lors d'une tentative de reconnexion IBM MQ classes for JMS automatique.

«Reconnexion automatique du client JMS», à la page 291 fournit une présentation des exceptions JMSExceptions et de la façon dont vos applications peuvent redémarrer automatiquement, ainsi que des informations dans «Utilisation de la reconnexion automatique du client JMS», à la page 292 qui détaillent les exigences relatives à la reconnexion automatique du client.

Les informations suivantes répertorient les codes anomalie IBM MQ que votre application doit vérifier:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING

RC2162
MQRC_Q_MGR_STOPPING

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_STOPPING

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

La plupart des exceptions JMSExceptions renvoyées aux applications d'entreprise contiennent une exception MQException liée qui contient le code anomalie. Pour implémenter la logique de nouvelle tentative pour les codes anomalie de la liste précédente, vos applications d'entreprise doivent vérifier cette exception liée à l'aide d'un code similaire à l'exemple suivant:

```
} catch (JMSException ex) {
        Exception linkedEx = ex.getLinkedException();
        if (ex.getLinkedException() != null) {
```

```
            if (linkedEx instanceof MQException) {
                MQException mqException = (MQException) linkedEx;
                int reasonCode = mqException.reasonCode;
                // Handle the reason code accordingly
            }
        }
    }
```

Concepts associés

IBM MQ classes for JMS

Utilisation de la reconnexion automatique du client dans les environnements Java SE et Java EE

Vous pouvez utiliser la reconnexion automatique du client IBM MQ pour faciliter diverses solutions de haute disponibilité et de reprise après incident dans un environnement Java SE et Java EE .

Différentes solutions de haute disponibilité et de reprise après incident sont disponibles sur différentes plateformes:

- Multi Les gestionnaires de files d'attente multi-instance sont des instances du même gestionnaire de files d'attente configurées sur des serveurs différents (voir Gestionnaires de files d'attente multi-instance). Une instance du gestionnaire de files d'attente est définie en tant qu'instance active et une autre instance est définie en tant qu'instance de secours. En cas de défaillance de l'instance active, le gestionnaire de files d'attente multi-instance redémarre automatiquement sur le serveur de secours.

  Les gestionnaires de files d'attente actifs et de secours ont le même identificateur de gestionnaire de files d'attente (QMID). Les applications client IBM MQ qui se connectent à un gestionnaire de files d'attente multi-instance peuvent être configurées pour se reconnecter automatiquement à une instance de secours d'un gestionnaire de files d'attente à l'aide de la reconnexion automatique du client.
- Linux Le gestionnaire de files d'attente de données répliquées (RDQM) est une solution à haute disponibilité disponible sur les plateformes Linux (voir Haute disponibilité du gestionnaire de files d'attente de données répliquées). Une configuration de gestionnaire de files d'attente de données répliquées comprend trois serveurs configurés dans un groupe à haute disponibilité (HA), chacun contenant une instance du gestionnaire de files d'attente. Une instance est le gestionnaire de files d'attente en cours d'exécution, qui réplique de manière synchrone ses données sur les deux autres instances. Si le serveur qui exécute ce gestionnaire de files d'attente échoue, une autre instance du gestionnaire de files d'attente démarre et peut utiliser les données actuelles. Les trois instances du gestionnaire de files d'attente partageant une adresse IP flottante, les clients n'ont besoin d'être configurés qu'avec une seule adresse IP. Les applications client qui se connectent à un gestionnaire de files d'attente de données répliquées peuvent être configurées pour se reconnecter automatiquement à une instance de secours d'un gestionnaire de files d'attente à l'aide de la reconnexion automatique du client.
- MQ Appliance Une solution à haute disponibilité peut également être fournie par une paire de dispositifs IBM MQ (voir Haute disponibilité et Reprise après incident dans la documentation IBM MQ Appliance). Un gestionnaire de files d'attente à haute disponibilité s'exécute sur l'un des dispositifs, tout en répliquant de manière synchrone les données sur l'instance de secours du gestionnaire de files d'attente sur l'autre dispositif. Si le dispositif principal est défaillant, le gestionnaire de files d'attente démarre automatiquement et s'exécute sur l'autre dispositif. Les deux instances du gestionnaire de files d'attente peuvent être configurées pour partager une adresse IP flottante, de sorte que les clients n'ont besoin d'être configurés qu'avec une seule adresse IP. Les applications client qui se connectent à un gestionnaire de files d'attente à haute disponibilité sur un dispositif IBM MQ peuvent être configurées pour se reconnecter automatiquement à l'instance de secours d'un gestionnaire de files d'attente à l'aide de la reconnexion automatique du client.

Remarque : Dans les environnements Java EE , tels que WebSphere Application Server, la reconnexion automatique du client avec les spécifications d'activation à l'aide de la fonctionnalité fournie par IBM MQ classes for JMS n'est pas prise en charge. L'adaptateur de ressources IBM MQ fournit son propre mécanisme de reconnexion des spécifications d'activation si le gestionnaire de files d'attente auquel la spécification d'activation se connectait devient indisponible. Pour plus d'informations, voir «Prise en charge de la reconnexion automatique du client dans les environnements Java EE», à la page 298.

Concepts associés

Gestionnaires de files d'attente multi-instance

reconnexion client automatique

Référence associée

Haute disponibilité de RDQM

Utilisation de la reconnexion automatique du client dans les environnements Java SE

Les applications qui utilisent des IBM MQ classes for JMS exécutées dans des environnements Java SE peuvent faire appel à la fonctionnalité de reconnexion automatique du client par le biais de la propriété **CLIENTRECONNECTOPTIONS** de fabrique de connexion.

La propriété de fabrique de connexions **CLIENTRECONNECTOPTIONS** utilise deux propriétés de fabrique de connexions supplémentaires, **CONNECTIONNAMELIST** et **CCDTURL**, pour déterminer comment se connecter au serveur sur lequel s'exécute le gestionnaire de files d'attente.

CONNECTIONNAMELIST Propriété

La propriété **CONNECTIONNAMELIST** est une liste séparée par des virgules qui contient le nom d'hôte et le port utilisés pour se connecter à un gestionnaire de files d'attente en mode client. Elle est utilisée avec les valeurs **QMANAGER** et **CHANNEL**. Lorsqu'une application utilise la propriété **CONNECTIONNAMELIST** pour créer une connexion client, les IBM MQ classes for JMS tentent de se connecter à chaque hôte dans l'ordre la liste. Si le premier hôte du gestionnaire de files d'attente n'est pas disponible, les IBM MQ classes for JMS tentent de se connecter à l'hôte suivant dans la liste. Si la fin de la liste des noms de connexions est atteinte sans qu'une connexion soit créée, les IBM MQ classes for JMS émettent le code anomalie `MQRC_QMGR_NOT_AVAILABLE` IBM MQ.

En cas de défaillance du gestionnaire de files d'attente auquel l'application est connectée, les applications qui utilisaient **CONNECTIONNAMELIST** pour se connecter à ce gestionnaire de files d'attente reçoivent une exception indiquant que le gestionnaire de files d'attente n'est pas disponible. L'application doit intercepter l'exception et supprimer les ressources qu'elle utilisait. Pour créer une connexion, l'application doit utiliser la fabrique de connexions. Cette dernière tente de se connecter à chaque hôte dans l'ordre de la liste. Le gestionnaire de files d'attente défaillant n'est plus disponible. La fabrique de connexions tente de se connecter à un autre hôte dans la liste.

CCDTURL Propriété

La propriété **CCDTURL** contient une URL (Uniform Resource Locator) qui pointe vers une table de définition de canal du client. Elle est utilisée avec la propriété **QMANAGER**. La table de définition de canal du client contient une liste des canaux client qui sont utilisés pour se connecter à un gestionnaire de files d'attente défini sur un système IBM MQ. Pour plus d'informations sur la façon dont les tables de définition de canal du client sont utilisées par IBM MQ classes for JMS, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for JMS», à la page 290.

Utilisation de la propriété CLIENTRECONNECTOPTIONS pour activer la reconnexion automatique du client dans IBM MQ classes for JMS

La propriété **CLIENTRECONNECTOPTIONS** est utilisée pour activer la reconnexion automatique du client dans les IBM MQ classes for JMS. Les valeurs possibles pour cette propriété sont les suivantes :

ASDEF
: Le comportement de la reconnexion automatique du client est défini par la valeur par défaut qui est indiquée dans la strophe de canal du fichier de configuration du client IBM MQ (`mqclient.ini`).

Désactivé
: La reconnexion automatique du client est désactivée.

QMGR
: Les IBM MQ classes for JMS tentent de se connecter à un gestionnaire de files d'attente avec le même identificateur de gestionnaire de files d'attente que le gestionnaire de files d'attente auquel elles étaient connectées, à l'aide de l'une des options suivantes :

- La propriété **CONNECTIONNAMELIST** et le canal qui est défini dans la propriété **CHANNEL**.
- La table de définition de canal du client définie dans la propriété **CCDTURL**.

Tout
Les IBM MQ classes for JMS tentent de se connecter à un gestionnaire de files d'attente ayant le même nom à l'aide de la propriété **CONNECTIONNAMELIST** ou **CCDTURL**.

Information associée
Strophe CHANNELS du fichier de configuration client

Utilisation de la reconnexion automatique du client dans les environnements Java EE
L'adaptateur de ressources IBM MQ , qui peut être déployé dans des environnements Java EE (Java Platform, Enterprise Edition), et le fournisseur de messagerie WebSphere Application Server IBM MQ utilisent le IBM MQ classes for JMS pour communiquer avec les gestionnaires de files d'attente IBM MQ . L'adaptateur de ressources IBM MQ et le fournisseur de messagerie WebSphere Application Server IBM MQ fournissent un certain nombre de mécanismes permettant aux spécifications d'activation, aux ports d'écoute WebSphere Application Server et aux applications s'exécutant dans des conteneurs client de se reconnecter automatiquement à un gestionnaire de files d'attente. Les EJB (Enterprise JavaBeans) et les applications Web doivent implémenter leur propre logique de reconnexion.

Remarque : La reconnexion automatique du client avec les spécifications d'activation à l'aide de la fonctionnalité fournie par IBM MQ classes for JMS n'est pas prise en charge (voir «Reconnexion automatique du client JMS», à la page 291). L'adaptateur de ressources IBM MQ fournit son propre mécanisme de reconnexion des spécifications d'activation si le gestionnaire de files d'attente auquel la spécification d'activation se connectait devient indisponible.

Le mécanisme fourni par l'adaptateur de ressources est contrôlé par:

- La propriété **reconnectionRetryCount**de l'adaptateur de ressources IBM MQ .
- La propriété **reconnectionRetryInterval**de l'adaptateur de ressources IBM MQ .
- Propriété de spécification d'activation **connectionNameList**.

Pour plus d'informations sur ces propriétés, voir «Configuration des propriétés de l'objet ResourceAdapter», à la page 460.

L'utilisation de la reconnexion automatique du client dans la méthode `onMessage()` d'une application de bean géré par message ou dans toute autre application exécutée dans l'environnement Java Platform, Enterprise Edition n'est pas prise en charge. L'application doit implémenter sa propre logique de reconnexion si le gestionnaire de files d'attente auquel elle se connectait devient indisponible. Pour plus d'informations, voir «Implémentation de la logique de reconnexion dans une application Java EE», à la page 306.

Prise en charge de la reconnexion automatique du client dans les environnements Java EE
Dans les environnements Java EE , tels que WebSphere Application Server, l'adaptateur de ressources IBM MQ et le fournisseur de messagerie WebSphere Application Server IBM MQ fournissent un certain nombre de mécanismes qui permettent aux applications de se reconnecter automatiquement à un gestionnaire de files d'attente. Dans certains cas cependant des restrictions s'appliquent à cette prise en charge.

L'adaptateur de ressources IBM MQ qui peut être déployé dans des environnements Java EE et le fournisseur de messagerie WebSphere Application Server IBM MQ, utilisent les IBM MQ classes for JMS pour communiquer avec les gestionnaires de files d'attente IBM MQ.

Le tableau suivant décrit le support offert par l'adaptateur de ressources IBM MQ et le fournisseur de messagerie WebSphere Application Server IBM MQ pour la reconnexion automatique du client.

Tableau 45. Récapitulatif de la prise en charge des options de reconnexion automatique du client dans les environnements Java EE

Options de reconnexion automatique	**Propriété CONNECTIONNAMELIST**	**Propriété CCDTURL**	**Propriété CLIENTRECONNECTOPTIONS**	**Approche alternative à la reconnexion automatique du client**
Spécifications d'activation	Prise en charge avec des restrictions	Prise en charge avec des restrictions	Non pris en charge	L'environnement Java EE et les spécifications d'activation fournissent leur propre mécanisme de reconnexion
WebSphere Application Server ports d'écoute	Prise en charge avec des restrictions	Prise en charge avec des restrictions	Non pris en charge	WebSphere Application Server fournit son propre mécanisme de reconnexion
Applications Enterprise JavaBeans et Web	Prise en charge avec des restrictions	Prise en charge avec des restrictions	Non pris en charge	Les applications doivent implémenter leur propre logique de reconnexion
Applications exécutées dans des conteneurs client	Pris en charge	Pris en charge	Pris en charge	Non applicable

Les applications de bean géré par message installées dans un environnement Java EE , tel que IBM MQ classes for JMS, peuvent utiliser des spécifications d'activation pour traiter des messages sur un système IBM MQ . Les spécifications d'activation permettent de détecter les messages qui arrivent sur un système IBM MQ et de les distribuer aux beans gérés par message pour traitement. Les beans gérés par message peuvent également établir d'autres connexions aux systèmes IBM MQ à partir de leur méthode **onMessage()**. Pour plus d'informations sur la manière dont ces connexions peuvent utiliser la reconnexion automatique du client, voir Applications Enterprise JavaBeans et Web.

Spécifications d'activation
En ce qui concerne les spécifications d'activation, les propriétés **CONNECTIONNAMELIST** et **CCDTURL** sont prises en charge avec des restrictions et la propriété **CLIENTRECONNECTOPTIONS** n'est pas prise en charge.

Les applications de bean géré par message (MDB) installées dans un environnement Java EE , tel que WebSphere Application Server, peuvent utiliser des spécifications d'activation pour traiter des messages sur un système IBM MQ .

Les spécifications d'activation sont utilisées pour détecter les messages qui parviennent à un système IBM MQ et les transmettre ensuite aux beans gérés par message pour traitement. Cette section décrit comme la spécification d'activation surveille le système IBM MQ.

Les beans gérés par message peuvent également établir des connexions supplémentaires aux systèmes IBM MQ à partir de leur méthode `onMessage()`.

Des détails sur la manière dont ces connexions peuvent utiliser la reconnexion automatique du client sont fournis dans «Applications Enterprise JavaBeans et Web», à la page 304.

CONNECTIONNAMELIST Propriété

Lors du démarrage, la spécification d'activation tente de se connecter au gestionnaire de files d'attente à l'aide :

- du gestionnaire de files d'attente indiqué dans la propriété **QMANAGER**,
- du canal mentionné dans la propriété **CHANNEL**,
- des informations de nom d'hôte et port provenant de la première entrée dans la propriété **CONNECTIONNAMELIST**

Si la spécification d'activation ne peut pas se connecter au gestionnaire de files d'attente à l'aide de la première entrée de la liste, elle passe à la deuxième entrée, et ainsi de suite, jusqu'à ce qu'une connexion au gestionnaire de files d'attente soit établie ou que la fin de la liste soit atteinte.

Si la spécification d'activation ne parvient pas à se connecter au gestionnaire de files d'attente spécifié à l'aide de l'une des entrées du **CONNECTIONNAMELIST**, elle s'arrête et doit être redémarrée.

Une fois en cours d'exécution, la spécification d'activation obtient le message du système IBM MQ et le transmet à un bean géré par message pour traitement.

En cas de défaillance du gestionnaire de files d'attente au cours du traitement d'un message, l'environnement Java EE détecte l'erreur et tente de reconnecter la spécification d'activation.

La spécification d'activation utilise les informations fournies dans la propriété **CONNECTIONNAMELIST**, comme précédemment, lors de sa tentative de reconnexion.

Si la spécification d'activation tente toutes les entrées du **CONNECTIONNAMELIST** et qu'elle ne parvient toujours pas à se connecter au gestionnaire de files d'attente, la spécification d'activation attend pendant la période indiquée par la IBM MQ propriété d'adaptateur de ressources **reconnectionRetryInterval** avant de faire une nouvelle tentative.

La IBM MQ propriété d'adaptateur de ressources **reconnectionRetryCount** définit le nombre de tentatives de reconnexion consécutives effectuées avant l'arrêt d'une spécification d'activation et nécessite un redémarrage manuel

Une fois que la spécification d'activation s'est reconnectée à un système IBM MQ, l'environnement Java EE effectue le nettoyage transactionnel requis et recommence à transmettre les messages aux beans gérés par message pour traitement.

Pour que le nettoyage transactionnel fonctionne correctement, l'environnement Java EE doit pouvoir accéder aux journaux du gestionnaire de files d'attente défaillant.

Si les spécifications d'activation sont utilisées avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectées à un gestionnaire de files d'attente multi-instance, **CONNECTIONNAMELIST** doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les beans gérés par message transactionnels sont utilisés avec des gestionnaires de files d'attente autonomes, la propriété **CONNECTIONNAMELIST** doit contenir une seule entrée pour s'assurer que la spécification d'activation se reconnecte toujours au même gestionnaire de files d'attente exécuté sur le même système à la suite d'une défaillance.

CCDTURL Propriété

Lors du démarrage, la spécification d'activation tente de se connecter au gestionnaire de files d'attente indiqué dans la propriété **QMANAGER** à l'aide de la première entrée de la table de définition de canal du client (CCDT).

Si la spécification d'activation ne peut pas se connecter au gestionnaire de files d'attente à l'aide de la première entrée de la table, elle passe à la deuxième entrée, et ainsi de suite, jusqu'à ce qu'une connexion au gestionnaire de files d'attente soit établie ou que la fin de la table soit atteinte.

Si la spécification d'activation ne parvient pas à se connecter au gestionnaire de files d'attente spécifié à l'aide de l'une des entrées de la table de définition de canal du client, la spécification d'activation s'arrête et doit être redémarrée.

Une fois en cours d'exécution, la spécification d'activation obtient le message du système IBM MQ et le transmet à un bean géré par message pour traitement.

En cas de défaillance du gestionnaire de files d'attente au cours du traitement d'un message, l'environnement Java EE détecte l'erreur et tente de reconnecter la spécification d'activation.

La spécification d'activation utilise les informations fournies dans la propriété CCDT, comme précédemment, lors de sa tentative de reconnexion.

Si la spécification d'activation tente toutes les entrées de la table de définition de canal du client et qu'elle ne parvient toujours pas à se connecter au gestionnaire de files d'attente, la spécification d'activation attend la période spécifiée par la IBM MQ propriété d'adaptateur de ressources **reconnectionRetryInterval** avant de faire une nouvelle tentative.

La IBM MQ propriété d'adaptateur de ressources **reconnectionRetryCount** définit le nombre de tentatives de reconnexion consécutives effectuées avant l'arrêt d'une spécification d'activation et nécessite un redémarrage manuel

Une fois que la spécification d'activation s'est reconnectée à un système IBM MQ, l'environnement Java EE effectue le nettoyage transactionnel requis et recommence à transmettre les messages aux beans gérés par message pour traitement.

Pour que le nettoyage transactionnel fonctionne correctement, l'environnement Java EE doit pouvoir accéder aux journaux du gestionnaire de files d'attente défaillant.

Si les spécifications d'activation sont utilisées avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectées à un gestionnaire de files d'attente multi-instance, la table de définition de canal du client doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les beans gérés par message transactionnels sont utilisés avec des gestionnaires de files d'attente autonomes, la table de définition de canal du client doit contenir une seule entrée pour s'assurer que la spécification d'activation se reconnecte toujours au même gestionnaire de files d'attente exécuté sur le même système à la suite d'une défaillance.

Veillez à définir *PREFERRED* comme valeur par défaut pour la propriété **AFFINITY** dans les tables de définition de canal du client utilisées avec les spécifications d'activation afin que les connexions soient établies avec le même gestionnaire de files d'attente actif.

CLIENTRECONNECTOPTIONS Propriété

Les spécifications d'activation fournissent leur propre fonctionnalité de reconnexion. Cette dernière permet aux spécifications de se reconnecter automatiquement à un système IBM MQ en cas de défaillance du gestionnaire de files d'attente auquel ils étaient connectés.

C'est la raison pour laquelle la fonctionnalité de reconnexion automatique du client fournie par les IBM MQ classes for JMS n'est pas prise en charge.

Vous devez définir la propriété **CLIENTRECONNECTOPTIONS** sur *DISABLED* pour toutes les spécifications d'activation utilisées dans l'environnement Java EE.

WebSphere Application Server ports d'écoute
Les applications de bean géré par message qui sont installées dans WebSphere Application Server peuvent également utiliser les ports d'écoute pour traiter les messages sur un système IBM MQ.

Les ports d'écoute sont utilisés pour détecter les messages qui parviennent à un système IBM MQ et les transmettre ensuite aux beans gérés par message pour traitement. Cette rubrique explique comment le port d'écoute surveille le système IBM MQ.

Les beans gérés par message peuvent également établir des connexions supplémentaires aux systèmes IBM MQ à partir de leur méthode `onMessage()`.

Pour plus d'informations sur la manière dont ces connexions peuvent utiliser la reconnexion automatique du client, voir «Applications Enterprise JavaBeans et Web», à la page 304.

En ce qui concerne les ports d'écoute WebSphere Application Server :

- **CONNECTIONNAMELIST** et **CCDTURL** sont pris en charge avec des restrictions
- **CLIENTRECONNECTOPTIONS** n'est pas pris en charge

CONNECTIONNAMELIST Propriété

Les ports d'écoute utilisent des pools de connexions JMS lors de la connexion à IBM MQ et sont donc concernés par ce qu'implique l'utilisation des pools de connexions. Pour plus d'informations, voir «Spécifications d'activation», à la page 299.

S'il n'y a pas de connexions libres et que le nombre maximal de connexions n'a pas encore été créé à partir de cette fabrique de connexions, la propriété **CONNECTIONNAMELIST** est utilisée pour tenter de créer une nouvelle connexion à IBM MQ.

Si tous les systèmes IBM MQ contenus dans **CONNECTIONNAMELIST** ne sont pas accessibles, le port d'écoute s'arrête.

Le port d'écoute se met en attente pendant la durée indiquée par la propriété personnalisée **RECOVERY.RETRY.INTERVAL** du service d'écoute de messages et tente à nouveau de se reconnecter.

Cette tentative de reconnexion vérifie s'il existe des connexions libres dans le pool de connexions, au cas où une connexion y aurait été replacée entre deux tentatives de connexion. Si aucune connexion n'est disponible, le port d'écoute utilise **CONNECTIONNAMELIST** comme précédemment.

Une fois que le port d'écoute s'est reconnecté à un système IBM MQ, l'environnement Java EE effectue le nettoyage transactionnel requis et recommence à transmettre les messages aux beans gérés par message pour traitement.

Pour que le nettoyage transactionnel fonctionne correctement, l'environnement Java EE doit pouvoir accéder aux journaux du gestionnaire de files d'attente défaillant.

Si les ports d'écoute sont utilisés avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectés à un **gestionnaire de files d'attente multi-instance**, **CONNECTIONNAMELIST** doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les beans gérés par message transactionnels sont utilisés avec des gestionnaires de files d'attente autonomes, la propriété **CONNECTIONNAMELIST** doit contenir une seule entrée pour s'assurer que la spécification d'activation se reconnecte toujours au même gestionnaire de files d'attente exécuté sur le même système à la suite d'une défaillance.

CCDTURL Propriété

Lors du démarrage, le port d'écoute tente de se connecter au gestionnaire de files d'attente indiqué dans la propriété **QMANAGER** à l'aide de la première entrée dans la table de définition de canal du client.

Si le port d'écoute ne peut pas se connecter au gestionnaire de files d'attente à l'aide de la première entrée de la table, il passe à la deuxième entrée, et ainsi de suite, jusqu'à ce qu'une connexion au gestionnaire de files d'attente soit établie ou que la fin de la table soit atteinte.

Si le port d'écoute ne peut se connecter au gestionnaire de files d'attente indiqué à l'aide d'aucune des entrées de la table, il s'arrête.

Le port d'écoute se met en attente pendant la durée indiquée par la propriété personnalisée **RECOVERY.RETRY.INTERVAL** du service d'écoute de messages et tente à nouveau de se reconnecter.

Cette tentative de reconnexion s'effectue par le biais de toutes les entrées de la table, comme précédemment.

Lorsque le port d'écoute est en cours d'exécution, il obtient des messages du système IBM MQ et les transmet à un bean géré par message pour traitement.

En cas de défaillance du gestionnaire de files d'attente au cours du traitement d'un message, l'environnement Java EE détecte l'erreur et tente de reconnecter le port d'écoute. Ce dernier utilise les informations contenues dans la table de définition de canal du client pour effectuer les tentatives de reconnexion.

Si le port d'écoute fait un essai avec toutes les entrées de la table et ne peut toujours pas se connecter au gestionnaire de files d'attente, il attend la période de temps indiquée par la propriété **RECOVERY.RETRY.INTERVAL** avant d'effectuer une nouvelle tentative.

La propriété **MAX.RECOVERY.RETRIES** du service d'écoute de messages définit le nombre de tentatives de reconnexions consécutives qui sont effectuées avant qu'un port d'écoute s'arrête et doive être redémarré manuellement.

Une fois que le port d'écoute s'est reconnecté à un système IBM MQ, l'environnement Java EE effectue le nettoyage transactionnel requis et recommence à transmettre les messages aux beans gérés par message pour traitement.

Pour que le nettoyage transactionnel fonctionne correctement, l'environnement Java EE doit pouvoir accéder aux journaux du gestionnaire de files d'attente défaillant.

Si les ports d'écoute sont utilisés avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectés à un gestionnaire de files d'attente multi-instance, la table de définition de canal du client doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les beans gérés par message transactionnels sont utilisés avec des gestionnaires de files d'attente autonomes, la table de définition de canal du client doit contenir une seule entrée pour s'assurer que le port d'écoute se reconnecte toujours au même gestionnaire de files d'attente exécuté sur le même système à la suite d'une défaillance.

Veillez à définir *PREFERRED* comme valeur par défaut pour la propriété **AFFINITY** dans les tables de définition de canal du client utilisées avec les ports d'écoute afin que les connexions soient établies avec le même gestionnaire de files d'attente actif.

CLIENTRECONNECTOPTIONS Propriété

Les ports d'écoute fournissent leur propre fonctionnalité de reconnexion. Cette dernière permet aux ports d'écoute de se reconnecter automatiquement à un système IBM MQ en cas de défaillance du gestionnaire de files d'attente auquel ils étaient connectés.

C'est la raison pour laquelle la fonctionnalité de reconnexion automatique du client fournie par les IBM MQ classes for JMS n'est pas prise en charge.

Vous devez définir la propriété **CLIENTRECONNECTOPTIONS** sur *DISABLED* pour tous les ports d'écoute utilisés dans l'environnement Java EE.

Applications Enterprise JavaBeans et Web
Les applications Enterprise JavaBean (EJB) et les applications qui s'exécutent dans le conteneur web, comme les servlets, utilisent une fabrique de connexions JMS pour créer une connexion à un gestionnaire de files d'attente IBM MQ.

Les restrictions suivantes s'appliquent aux EJB et aux applications Web :

- **CONNECTIONNAMELIST** et **CCDTURL** sont pris en charge avec des restrictions
- **CLIENTRECONNECTOPTIONS** n'est pas pris en charge

CONNECTIONNAMELIST Propriété

Si l'environnement Java EE fournit un pool de connexions pour les connexions JMS, voir «Utilisation de CONNECTIONNAMELIST ou CCDT dans un pool de connexions», à la page 305 pour en connaître les effets sur le comportement de la propriété **CONNECTIONNAMELIST**.

Si l'environnement Java EE ne fournit pas un pool de connexions JMS, l'application utilise la propriété **CONNECTIONNAMELIST** de la même façon que les applications Java SE.

Si les applications sont utilisées avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectées à un gestionnaire de files d'attente multi-instance, **CONNECTIONNAMELIST** doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les applications sont utilisées avec des gestionnaires de files d'attente autonomes, la propriété **CONNECTIONNAMELIST** doit contenir une seule entrée, pour s'assurer que l'application se reconnecte toujours au même gestionnaire de files d'attente, s'exécutant sur le même système, à la suite d'un incident.

CCDTURL Propriété

Si l'environnement Java EE fournit un pool de connexions pour les connexions JMS, voir «Utilisation de CONNECTIONNAMELIST ou CCDT dans un pool de connexions», à la page 305 pour en connaître les effets sur le comportement de la propriété **CCDTURL**.

Si l'environnement Java EE ne fournit pas un pool de connexions JMS, l'application utilise la propriété **CCDTURL** de la même façon que les applications Java SE.

Si les applications sont utilisées avec des beans gérés par message transactionnels qui participent aux transactions XA et sont connectées à un gestionnaire de files d'attente multi-instance, la table de définition de canal du client doit contenir une entrée à la fois pour l'instance de gestionnaire de files d'attente active et de secours.

Cela signifie que l'environnement Java EE peut accéder aux journaux du gestionnaire de files d'attente si l'environnement doit effectuer une récupération de transaction, quel que soit le gestionnaire de files d'attente auquel se reconnecte l'environnement à la suite d'une défaillance.

Si les applications sont utilisées avec des gestionnaires de files d'attente autonomes, la table de définition de canal du client doit contenir une seule entrée pour s'assurer que les applications se reconnectent toujours au même gestionnaire de files d'attente exécuté sur le même système à la suite d'une défaillance.

CLIENTRECONNECTOPTIONS Propriété

Vous devez définir la propriété **CLIENTRECONNECTOPTIONS** sur *DISABLED* pour toutes les fabriques de connexions JMS utilisées par les EJB ou les applications qui s'exécutent dans le conteneur web.

Les applications qui doivent se reconnecter automatiquement à un nouveau gestionnaire de files d'attente en cas de défaillance du gestionnaire de files d'attente qu'elles utilisent doivent implémenter

leur propre logique de reconnexion. Pour plus d'informations, voir «Implémentation de la logique de reconnexion dans une application Java EE», à la page 306.

Scénarios: WebSphere Application Server avec IBM MQ

Scénarios: profil WebSphere Application Server Liberty avec IBM MQ

Applications exécutées dans des conteneurs client
Certains environnements Java EE EE, tels que WebSphere Application Server, fournissent un conteneur client qui peut être utilisé pour exécuter des applications Java SE.

Les applications qui s'exécutent dans ces environnements utilisent une fabrique de connexions JMS pour se connecter à un gestionnaire de files d'attente IBM MQ.

En ce qui concerne les applications exécutées dans des conteneurs client :

- **CONNECTIONNAMELIST** et **CCDTURL** sont entièrement prises en charge
- **CLIENTRECONNECTOPTIONS** est entièrement prise en charge

CONNECTIONNAMELIST Propriété

Si l'environnement Java EE fournit un pool de connexions pour les connexions JMS, voir «Utilisation de CONNECTIONNAMELIST ou CCDT dans un pool de connexions», à la page 305 pour en connaître les effets sur le comportement de la propriété **CONNECTIONNAMELIST**.

Si l'environnement Java EE ne fournit pas un pool de connexions JMS, l'application utilise la propriété **CONNECTIONNAMELIST** de la même façon que les applications Java SE.

CCDTURL Propriété

Si l'environnement Java EE fournit un pool de connexions pour les connexions JMS, voir «Utilisation de CONNECTIONNAMELIST ou CCDT dans un pool de connexions», à la page 305 pour en connaître les effets sur le comportement de la propriété **CCDTURL**.

Si l'environnement Java EE ne fournit pas un pool de connexions JMS, l'application utilise la propriété **CCDTURL** de la même façon que les applications Java SE.

Utilisation de CONNECTIONNAMELIST ou CCDT dans un pool de connexions
Certains environnements Java EE , par exemple WebSphere Application Server, fournissent un pool de connexions JMS . qui peut être utilisé pour exécuter des applications Java SE .

Les applications qui créent une connexion à l'aide d'une fabrique de connexions définie dans l'environnement Java EE obtiennent une connexion libre existante du pool de connexions pour cette fabrique de connexions ou une nouvelle connexions s'il n'existe pas de connexion appropriée dans le pool de connexions.

Cela peut avoir des implications si la fabrique de connexions est configurée avec la propriété **CONNECTIONNAMELIST** ou **CCDTURL** définie.

La première fois que la fabrique de connexions est utilisée pour créer une connexion, l'environnement Java EE utilise **CONNECTIONNAMELIST**. ou **CCDTURL** pour créer une nouvelle connexion au système IBM MQ . Lorsque cette connexion n'a plus d'utilité, elle est renvoyée dans le pool de connexions et elle redevient disponible pour être réutilisée.

Si un autre élément crée une connexion à partir de la fabrique de connexions, l'environnement Java EE renvoie la connexion du pool de connexions au lieu d'utiliser la propriété **CONNECTIONNAMELIST** ou **CCDTURL** pour créer une connexion.

Si une connexion est utilisée lorsqu'une instance de gestionnaire de files d'attente échoue, elle est annulée. Toutefois, le contenu du pool de connexions ne l'est pas, ce qui signifie que le pool peut encore potentiellement contenir des connexions à un gestionnaire de files d'attente qui n'est plus en cours d'exécution.

Dans cette situations, la prochaine fois qu'une demande de création d'une connexion à partir de la fabrique de connexions est émise, une connexion au gestionnaire de files d'attente défaillant est renvoyée. Toute tentative d'utilisation de cette connexion échoue étant donné que le gestionnaire de files d'attente n'est plus en cours d'exécution, ce qui entraîne l'annulation de la connexion.

C'est uniquement lorsque le pool de connexions est vide que l'environnement Java EE utilise la propriété **CONNECTIONNAMELIST** ou **CCDTURL** pour créer une connexion à IBM MQ.

En raison de la manière dont la propriété **CONNECTIONNAMELIST** et les tables de définition de canal du client sont utilisées pour créer des connexions JMS, il est également possible qu'un pool de connexions contienne des connexions à des systèmes IBM MQ différents.

Par exemple, supposons qu'une fabrique de connexions soit configurée avec la propriété **CONNECTIONNAMELIST** définie à la valeur suivante :

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Supposons que la première fois qu'une application tente de créer une connexion à un gestionnaire de files d'attente autonome à partir de cette fabrique de connexions, le gestionnaires de files d'attente qui s'exécute sur le système `hostname1(port1)` ne soit pas accessible. Cela signifie que l'application se termine avec une connexion au gestionnaire de files d'attente exécuté sur `hostname2(port2)`.

Une autre application intervient et crée une connexion JMS à partir de la même fabrique de connexions. Le gestionnaire de files d'attente sur `hostname1(port1)` est maintenant disponible alors qu'une connexion JMS est créée à ce système IBM MQ et est renvoyée à l'application.

Lorsque les deux applications s'arrêtent, elles ferment leurs connexions JMS qui sont alors renvoyées dans le pool de connexions.

Le pool de connexions pour notre fabrique de connexions comporte donc maintenant deux connexions JMS :

- Une connexion au gestionnaire de files d'attente exécuté sur `hostname1(port1)`
- Une connexion au gestionnaire de files d'attente exécuté sur `hostname2(port2)`

Cela peut entraîner des problèmes liés à la récupération des transactions. Si le systèmeJava EE doit annuler une transaction, il doit pouvoir se connecter à un gestionnaire de files d'attente qui a accès aux journaux des transactions.

Implémentation de la logique de reconnexion dans une application Java EE
Les JavaBeans d'entreprise et les applications Web qui souhaitent se reconnecter automatiquement en cas d'échec d'un gestionnaire de files d'attente doivent implémenter leur propre logique de reconnexion.

Les options suivantes fournissent plus d'informations sur la manière dont vous pouvez atteindre cet objectif.

Autoriser l'échec de l'application

Cette approche ne requiert pas de modification de l'application, mais exige une reconfiguration administrative de la définition de fabrique de connexions dans le but d'inclure la propriété **CONNECTIONNAMELIST**. Toutefois, cette approche exige que l'auteur de l'appel puisse traiter une défaillance de manière appropriée. Sachez que cela est également nécessaire pour les défaillances telles que MQRC_Q_FULL qui ne sont pas liées à un échec de connexion.

Exemple de code pour ce processus :

```
public class SimpleServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
         try {
 // get connection factory/ queue
 InitialContext ic = new InitialContext();
 ConnectionFactory cf =
```

```
           (ConnectionFactory)ic.lookup("java:comp/env/jms/WMQCF");
 Queue q = (Queue) ic.lookup("java:comp/env/jms/WMQQueue");

 // send a message
 Connection c = cf.createConnection();
 Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
 MessageProducer p = s.createProducer(q);
 Message m = s.createTextMessage();
 p.send(m);

 // done, release the connection
 c.close();
 }
 catch (JMSException je) {
 // process exception
   }
 }
}
```

Le code précédent considère que la propriété **CONNECTIONNAMELIST** est définie pour la fabrique de connexions utilisée par ce servlet.

Lors du premier traitement du servlet, une nouvelle connexion est créée à l'aide de la propriété **CONNECTIONNAMELIST** , en supposant qu'aucune connexion en pool n'est disponible à partir d'autres applications se connectant au même gestionnaire de files d'attente.

Lorsque la connexion est libérée à la suite d'un appel `close()`, cette connexion est renvoyée dans le pool et réutilisée lors de la prochaine exécution du servlet - sans faire référence à **CONNECTIONNAMELIST** - jusqu'à ce qu'un échec de connexion se produise, auquel cas un événement CONNECTION_ERROR_OCCURRED est généré. Cet événement invite le pool à détruite la connexion défaillante.

Lors de la prochaine exécution de l'application, aucune connexion regroupée n'est disponible et **CONNECTIONNAMELIST** est utilisé pour se connecter au premier gestionnaire de files d'attente disponible. Si la bascule du gestionnaire de files d'attente a été effectuée (dans le cas par exemple où l'échec n'était pas une défaillance réseau transitoire), le servlet se connecte à l'instance de secours dès qu'elle est disponible.

Si d'autres ressources, comme des bases de données, sont appelées dans l'application, il peut s'avérer nécessaire d'indiquer que le serveur d'applications doit annuler la transaction.

Traitement de la reconnexion dans l'application

Si l'auteur de l'appel ne peut pas traiter un échec du servlet, la reconnexion doit être traitée par l'application. Comme indiqué dans l'exemple suivant, la gestion d'une connexion dans l'application exige que cette dernière demande une nouvelle connexion afin qu'elle puisse mettre en cache la fabrique de connexions recherchée dans JNDI et traiter une `JMSException` comme JMSCMQ0001: `WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN')`.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

  // get connection factory/ queue
  InitialContext ic = new InitialContext();
  ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
  Destination destination = (Destination) ic.lookup("java:comp/env/jms/WMQQueue");

  setupResources();

  // loop sending messages
  while (!sendComplete) {
    try {
      // create the next message to send
      msg.setText("message sent at "+new Date());
      // and send it
      producer.send(msg);
    }
    catch (JMSException je) {
```

```
        // drive reconnection
        setupResources();
    }
  }
```

Dans l'exemple suivant, `setupResources()` crée les objets JMS et inclut une boucle de mise en veille et de relance pour traiter les connexions non instantanées. En pratique, cette méthode empêche un grand nombre de tentatives de reconnexion. Sachez que des conditions d'exit ont été omises dans l'exemple pour plus de clarté.

```
 private void setupResources() {

    boolean connected = false;
    while (!connected) {
      try {
        connection = cf.createConnection(); // cf cached from JNDI lookup
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        msg = session.createTextMessage();
        producer = session.createProducer(destination); // destination cached from JNDI lookup
        // no exception? then we connected ok
        connected = true;
      }
      catch (JMSException je) {
        // sleep and then have another attempt
        try {Thread.sleep(30*1000);} catch (InterruptedException ie) {}
      }
    }
```

Si l'application gère la reconnexion, il est important qu'elle libère les connexions qu'elle détenait à d'autres ressources, qu'il s'agisse d'autres gestionnaires de files d'attente IBM MQ ou d'autres services de back-en, comme des bases de données. Vous devez établir à nouveau ces connexions une fois terminée la reconnexion à une nouvelle instance de gestionnaire de files d'attente IBM MQ. Si vous ne rétablissez pas les connexions, les ressources du serveur d'applications sont détenues inutilement pendant la durée de la tentative de reconnexion et peuvent avoir dépassé le délai d'expiration au moment d'être réutilisées.

Utilisation de WorkManager

En ce qui concerne les applications à longue durée de vie (traitement par lots, par exemple) où le temps de traitement est supérieur à quelques dizaines de secondes, WebSphere Application Server WorkManager peut être utilisé. Exemple de fragment de code pour WebSphere Application Server :

```
public class BatchSenderServlet extends HttpServlet  {

  private WorkManager workManager = null;
  private MessageSender sender; // background sender WorkImpl

  public void init() throws ServletException {
    InitialContext ctx = new InitialContext();
    workManager = (WorkManager)ctx.lookup(java:comp/env/wm/default);
    sender = new MessageSender(5000);
    workManager.startWork(sender);
  }

  public void destroy() {
    sender.halt();
  }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    if (sender.isRunning()) {
      out.println(sender.getStatus());
    }
}
```

où `web.xml` contient :

```
<resource-ref>
      <description>WorkManager</description>
```

```
      <res-ref-name>wm/default</res-ref-name>
      <res-type>com.ibm.websphere.asynchbeans.WorkManager</res-type>
      <res-auth>Container</res-auth>
      <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
```

et le lot est maintenant implémenté via l'interface de travail :

```
import com.ibm.websphere.asynchbeans.Work;

public class MessageSender implements Work {

  public MessageSender(int messages) {numberOfMessages = messages;}

  public void run() {
    // get connection factory/ queue
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory)
              ic.lookup("java:comp/env/jms/WMQCF");
    Destination destination = (Destination) ic.lookup("jms/WMQQueue");

    setupResources();

    // loop sending messages
    while (!sendComplete) {
      try {
        // create the next message to send
        msg.setText("message sent at "+new Date());
        // and send it
        producer.send(msg);
        // are we finished?
        if (sendCount == numberOfMessages) {sendComplete = true);
      }
      catch (JMSException je) {
          // drive reconnection
          setupResources();
      }
  }

  public boolean isRunning() {return !sendComplete;}

  public void release() {sendComplete = true;}
```

Si le traitement par lots prend plus longtemps à s'exécuter, dans le cas des messages de grande taille, d'un réseau à faible débit ou d'un accès intensif à la base de données (surtout lorsque la bascule est lente), le serveur commence à émettre des avertissements d'unité d'exécution bloquée comme dans l'exemple suivant :

WSVR0605W : L'unité d'exécution "`WorkManager.DefaultWorkManager` : 0" (00000035) est active depuis 694061 millisecondes et il se peut qu'elle soit bloquée. Il existe une unité ou x unités d'exécution au total qui peuvent être bloquées dans le serveur.

Il est possible de limiter ces avertissements en réduisant la taille des lots ou en augmentant le délai d'unité d'exécution bloquée. Toutefois, il est généralement préférable d'implémenter ce traitement dans un EJB (pour l'envoi par lots) ou dans un bean géré par message (pour la consommation et la réponse).

Sachez que la reconnexion gérée par une application ne fournit pas une solution générale pour traiter les erreurs d'exécution et l'application doit encore traiter les erreurs qui ne sont pas liées à l'échec de la connexion.

Exemple : tentative de placement d'un message dans une file d'attente qui est pleine (2053 MQRC_Q_FULL) ou tentative de connexion à un gestionnaire de files d'attente à l'aide de données d'identification de sécurité qui ne sont pas valides (2035 MQRC_NOT_AUTHORIZED).

L'application doit également traiter les erreurs 2059 MQRC_Q_MGR_NOT_AVAILABLE lorsqu'aucune instance n'est immédiatement disponible lorsque la bascule est en cours. Pour cela, elle signale les exceptions JMS lorsqu'elles surviennent au lieu d'effectuer une tentative de reconnexion en mode silencieux.

Regroupement d'objets IBM MQ classes for JMS
L'utilisation d'une forme de regroupement de connexions en dehors de Java EE permet de réduire la charge globale résultant, par exemple, de certaines applications autonomes utilisant des infrastructures ou déployées dans des environnements de cloud, ainsi que d'un plus grand nombre de connexions client à `QueueManagers` , ce qui entraîne une augmentation de la consolidation des serveurs des applications et des gestionnaires de files d'attente.

Dans le modèle de programmation Java EE , il existe un cycle de vie bien défini des différents objets utilisés. Les beans gérés par message (MDB) sont les plus contraints, tandis que les servlets offrent plus de liberté. Par conséquent, les options de mise en pool disponibles dans les serveurs Java EE sont adaptées aux différents modèles de programmation utilisés.

Avec Java SE (ou avec une autre infrastructure telle que Spring), les modèles de programmation sont extrêmement flexibles. Par conséquent, une stratégie de mise en commun unique ne convient pas à tous. Vous devriez examiner s'il y a un cadre en place qui pourrait faire n'importe quelle forme de mise en commun, par exemple, Spring.

La stratégie de mise en pool à utiliser dépend de l'environnement dans lequel votre application s'exécute.

Mise en pool d'objets dans un environnement Java EE
Les serveurs d'applications Java EE fournissent une fonctionnalité de regroupement de connexions qui peut être utilisée par les applications de bean géré par message, les beans Enterprise Java et les servlets.

WebSphere Application Server gère un pool de connexions à un fournisseur JMS dans le but d'améliorer les performances. Lorsqu'une application crée une connexion JMS, le serveur d'applications détermine si une connexion existe déjà dans le pool de connexions libress. Si c'est le cas, la connexion est renvoyée à l'application ; sinon, une connexion est créée.

La Figure 41, à la page 310 présente les spécifications d'activation et les ports d'écoute qui établissent une connexion JMS et utilisent cette dernière afin de surveiller une destination pour des messages en mode normal.

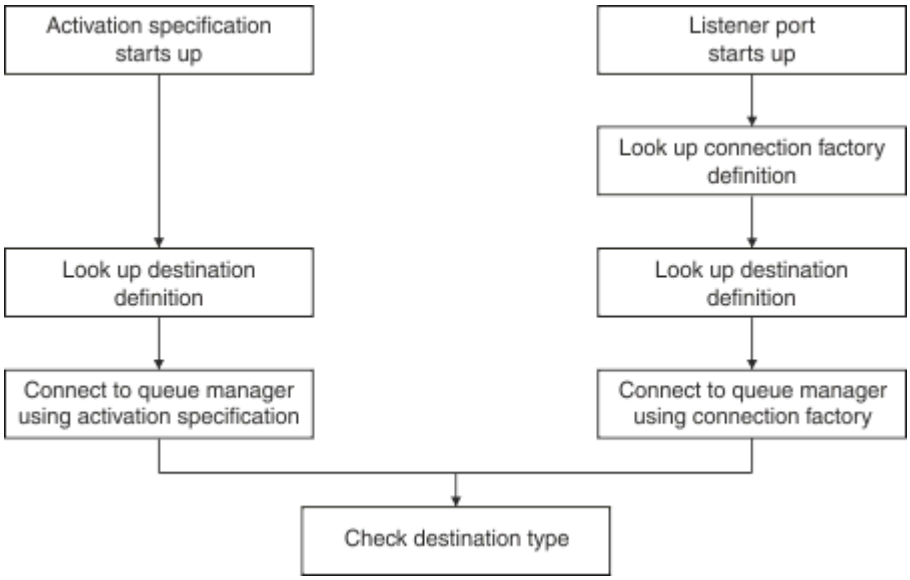


Figure 41. Mode normal

Lorsque vous utilisez le fournisseur de messagerie IBM MQ , les applications qui effectuent des opérations de messagerie sortante (telles que les beans Java d'entreprise et les servlets) et le composant de port d'écoute de bean géré par message peuvent utiliser ces pools de connexions.

Les spécifications d'activation du fournisseur de messagerie IBM MQ utilisent la fonctionnalité de regroupement de connexions fournie par l'adaptateur de ressources IBM MQ. Pour plus d'informations, voir Configuration des propriétés pour l'adaptateur de ressources WebSphere MQ.

Les «Exemples d'utilisation du pool de connexions», à la page 315 indiquent comment les applications qui exécutent des fonctions de messagerie sortante et les ports d'écoute utilisent le pool libre lors de la création de connexions JMS.

Les «Unités d'exécution de maintenance de pool de connexions libres», à la page 318 décrivent ce que deviennent ces connexions lorsqu'une application ou un port d'écoute a fini de les utiliser.

Les «Exemples d'unité d'exécution de maintenance de pool», à la page 319 décrivent comment le pool de connexions libress est nettoyé pour empêcher les connexions JMS de devenir périmées.

WebSphere Application Server impose une limite au nombre de connexions pouvant être créées à partir d'une fabrique. Ce nombre est indiqué par la propriété *maximum connections* de la fabrique de connexions. La valeur par défaut de cette propriété est 10, ce qui signifie que 10 connexions au maximum peuvent être créées à tout moment à partir d'une fabrique.

Un pool de connexions libress est associé à chaque fabrique. Lorsque le serveur d'applications démarre, les pools de connexions libres sont vides. Le nombre maximal de connexions pouvant exister dans le pool libre pour une fabrique est également indiqué par la propriété Maximum connections.

Conseil : Avec JMS 2.0, une fabrique de connexions peut être utilisée pour créer des connexions ainsi que des contextes. Ainsi, un pool de connexions peut être associé à une fabrique de connexions contenant un mélange de connexions et de contextes. Il est recommandé d'utiliser une fabrique de connexions pour la création de connexions ou de contextes seulement. Vous avez ainsi la garantie que le pool de connexions pour cette fabrique de connexions contient des objets d'un seul type seulement, ce qui le rend plus efficace.

Pour plus d'informations sur le fonctionnement du regroupement de connexions dans WebSphere Application Server, voir Configuration du regroupement de connexions pour les connexions JMS. Pour les autres serveurs d'applications, reportez-vous à la documentation appropriée du serveur d'applications.

Mode d'utilisation du pool de connexions

Chaque fabrique de connexions JMS est associé à un pool de connexions, lequel contient zéro ou plusieurs connexions JMS. Chaque connexion JMS est associée à un pool de sessions JMS et chaque pool de sessions JMS contient zéro ou plusieurs sessions JMS.

La Figure 42, à la page 312 décrit la relation entre ces objets.

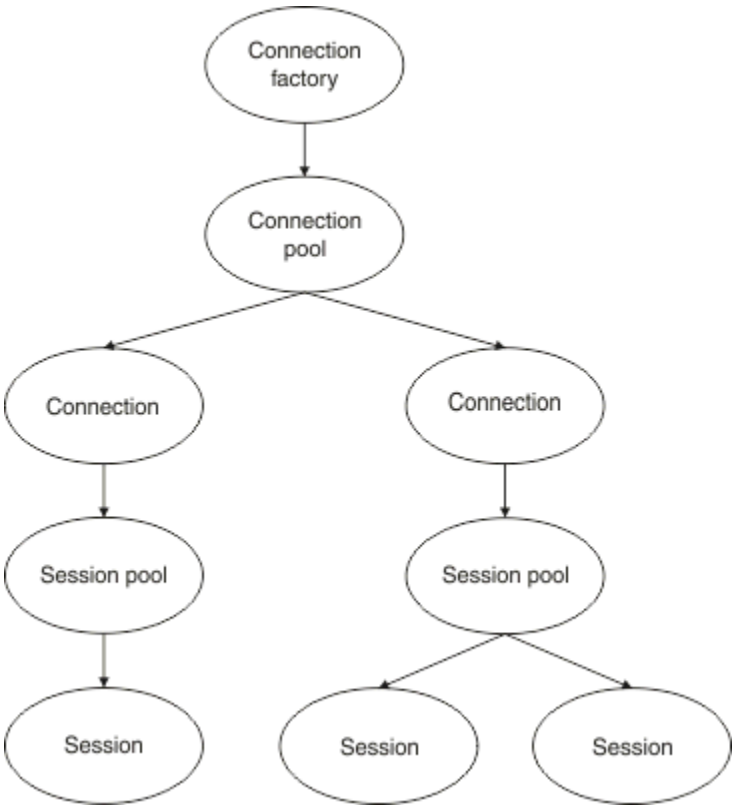


Figure 42. Pools de connexions et pools de sessions

Lorsqu'un port d'écoute démarre ou qu'une application souhaitant faire appel à la messagerie sortante utilise la fabrique pour créer une connexion, le port ou l'application appelle l'une des méthodes suivantes :

- **connectionFactory.createConnection()**
- **ConnectionFactory.createConnection(String, String)**
- **QueueConnectionFactory.createQueueConnection()**
- **QueueConnectionFactory.createQueueConnection(String, String)**
- **TopicConnectionFactory.createTopicConnection()**
- **TopicConnectionFactory.createTopicConnection(String, String)**

Le gestionnaire de connexions WebSphere Application Server tente d'obtenir une connexion depuis le pool libre pour cette fabrique et la renvoie à l'application.

S'il n'existe pas de connexion libre dans le pool, et que le nombre de connexions créées à partir de cette fabrique n'a pas atteint la limite indiquée dans la propriété *maximum connections* de cette fabrique, le gestionnaire de connexions crée une connexion destinée à l'application.

Toutefois, si une application tente de créer une connexion, mais que le nombre de connexions créées à partir de cette fabrique est déjà égal à la valeur de la propriété *maximum connections* de la fabrique de connexions, l'application attend qu'une connexion devienne disponible (replacée dans le pool libre).

Le délai d'attente de l'application est spécifié dans la propriété *connection timeout* du pool de connexions. Il est de 180 secondes par défaut. Si une connexion est replacée dans le pool libre dans cet intervalle de 180 secondes, le gestionnaire de connexions la retire à nouveau immédiatement du pool et la transmet à l'application. Toutefois, si le délai d'attente est dépassé, l'exception *ConnectionWaitTimeoutException* est émise.

Lorsqu'une application a fini d'utiliser la connexion et la ferme en appelant :

- **Connection.close()**

- **QueueConnection.close()**
- **TopicConnection.close()**

la connexion est en fait maintenue ouverte et elle est renvoyée au pool libre pour pouvoir être réutilisée par une autre application. Vous pouvez ainsi disposer de connexions ouvertes entre WebSphere Application Server et le fournisseur JMS, même si aucune application JMS ne s'exécute sur le serveur d'applications.

Propriétés avancées du pool de connexions
Un certain nombre de propriétés avancées peuvent être utilisées pour contrôler le comportement des pools de connexions JMS.

Protection contre les surtensions

La rubrique «Comment les applications qui font appel à la fonction de messagerie sortante utilisent le pool de connexions», à la page 317 décrit l'utilisation de la méthode `sendMessage()` qui incorpore `connectionFactory.createConnection()`.

Examinons le cas où 50 EJB créent des connexions JMS à partir de la même fabrique connexions dans le cadre de leur méthode `ejbCreate()`.

Si tous ces beans sont créés simultanément et s'il n'y a pas de connexion dans le pool de connexions libres de la fabrique, le serveur d'applications tente de créer simultanément 50 connexions JMS au même fournisseur JMS. Cela génère une charge très importante à la fois sur WebSphere Application Server et le fournisseur JMS.

Les propriétés de protection contre les surtensions peuvent éviter cette situation en limitant le nombre de connexions JMS pouvant être créées simultanément à partir d'une fabrique de connexions et en échelonnant la création de connexions supplémentaires.

La limitation du nombre de connexions JMS à un instant donné s'effectue par le biais de deux propriétés :

- Seuil de hausse
- Intervalle de création en situation de hausse

Lorsque les applications EJB tentent de créer une connexion JMS à partir d'une fabrique de connexions, le gestionnaire de connexions détermine le nombre de connexions qui sont créées. Si ce nombre est inférieur ou égal à la valeur de la propriété `surge threshold` , le gestionnaire de connexions continue d'ouvrir de nouvelles connexions.

Toutefois, si le nombre de connexions créées dépasse la valeur de la propriété `surge threshold` , le gestionnaire de connexions attend le délai spécifié par la propriété `surge creation interval` avant de créer et d'ouvrir une nouvelle connexion.

Connexions inactives

Une connexion JMS est considérée comme `stuck`si une application JMS utilise cette connexion pour envoyer une demande au fournisseur JMS et que le fournisseur ne répond pas dans un certain délai.

WebSphere Application Server permet de détecter les connexions `stuck` JMS . Pour utiliser cette fonction, vous devez définir trois propriétés:

- Temporisateur d'inactivité
- Durée d'inactivité
- Nombre maximal de connexions inactives

La rubrique «Exemples d'unité d'exécution de maintenance de pool», à la page 319 explique comment l'unité d'exécution de maintenance de pool s'exécute périodiquement et vérifie le contenu du pool libre d'une fabrique de connexions en recherchant les connexions qui sont inutilisées pendant un certain temps ou qui existent depuis trop longtemps.

Pour détecter les connexions inactives, le serveur d'applications gère également une unité d'exécution de connexion inactive qui vérifie l'état de toutes les connexions actives créées à partir d'une fabrique de connexions pour déterminer si l'une d'elles est en attente de réponse du fournisseur JMS.

Lorsque l'unité d'exécution de connexion bloquée est exécutée, elle est déterminée par la propriété `Stuck time timer` . La valeur par défaut de cette propriété est zéro, ce qui signifie que la détection des connexions inactives n'est jamais lancée.

Si l'unité d'exécution en trouve une qui attend une réponse, elle détermine la durée d'attente et compare cette durée à la valeur de la propriété `Stuck time` .

Si le délai de réponse du fournisseur JMS dépasse le délai spécifié par la propriété `Stuck time` , le serveur d'applications marque la connexion JMS comme étant bloquée.

Par exemple, supposons que la fabrique de connexions `jms/CF1` ait la propriété `Stuck time timer` définie sur 10 et la propriété `Stuck time` définie sur 15.

L'unité d'exécution de détection des connexions inactives s'active toutes les 10 secondes et elle vérifie si une connexion créée à partir de `jms/CF1` attend une réponse d'IBM MQ depuis plus de 15 secondes.

Supposons qu'un EJB crée une connexion JMS à IBM MQ à l'aide de `jms/CF1` et tente ensuite de créer une session JMS à l'aide de cette connexion en appelant `Connection.createSession()`.

Cependant, quelque chose empêche le fournisseur JMS de répondre à la demande. La machine est peut-être gelée ou un processus exécuté sur le fournisseur JMS est inactif, ce qui empêche le traitement de tout nouveau travail :

Dix secondes après que l'EJB a appelé `Connection.createSession()`, le temporisateur des connexions inactives s'active et recherche les connexions actives qui ont été créées à partir de `jms/CF1`.

Supposons qu'il n'existe qu'une seule connexion active, appelée par exemple `c1`. Le premier EJB a attendu 10 secondes une réponse à une demande qu'il a envoyée à `c1`, ce qui est inférieur à la valeur de `Stuck time`. Par conséquent, le temporisateur de connexion bloquée ignore cette connexion et devient inactif.

10 secondes après, l'unité d'exécution de détection des connexions inactives redevient active et recherche les connexions actives pour `jms/CF1`. Comme précédemment, supposons qu'il n'existe qu'une seule connexion, `c1`.

Cela fait maintenant 20 secondes que le premier EJB a appelé `createSession()` et ce dernier attend toujours une réponse. 20 secondes étant supérieures à la durée spécifiée dans la propriété `Stuck time` , l'unité d'exécution de connexion bloquée marque `c1` comme étant bloquée.

Si, cinq secondes après, IBM MQ finit par répondre et permet au premier EJB de créer une session JMS, la connexion est réutilisée.

Le serveur d'applications compte le nombre de connexions JMS créées à partir d'une fabrique de connexions qui sont inactives. Lorsqu'une application utilise cette fabrique de connexions pour créer une connexion JMS et qu'aucune connexion n'est disponible dans le pool de connexion de cette fabrique, le gestionnaire de connexions compare le nombre de connexions inactives à la valeur de la propriété `Stuck threshold`.

Si le nombre de connexions inactives est inférieur à la valeur définie pour la propriété `Stuck threshold`, le gestionnaire de connexions crée une nouvelle connexion et la transmet à l'application.

Toutefois, si le nombre de connexions inactives est égal à la valeur de la propriété `Stuck threshold` , l'application reçoit une exception de ressource.

Partitions de pool

WebSphere Application Server fournit deux propriétés qui permettent de partitionner le pool de connexions libres pour une fabrique de connexions :

- `Number of free pool partitions` indique au serveur d'applications le nombre de partitions dans lesquelles vous souhaitez diviser le pool de connexions libres.

• `Free pool distribution table size` détermine comment les partitions sont indexées.

Laissez ces propriété définies sur leurs valeurs par défaut (zéro) à moins que le centre de support IBM vous demande de les modifier.

Notez que WebSphere Application Server possède une propriété de pool de connexions avancée supplémentaire appelée `Number of shared partitions`. Cette propriété indique le nombre de propriétés utilisées pour stocker les connexions partagées. Cependant, comme les connexions JMS sont toujours partagées, cette propriété ne s'applique pas.

Exemples d'utilisation du pool de connexions
Le composant de port d'écoute de bean géré par message et les applications qui font appel aux fonctions de messagerie sortante utilisent un pool de connexions JMS.

La Figure 43, à la page 315 décrit comment le pool de connexions fonctionne pour WebSphere Application Server versions 7.5 et 8.0.

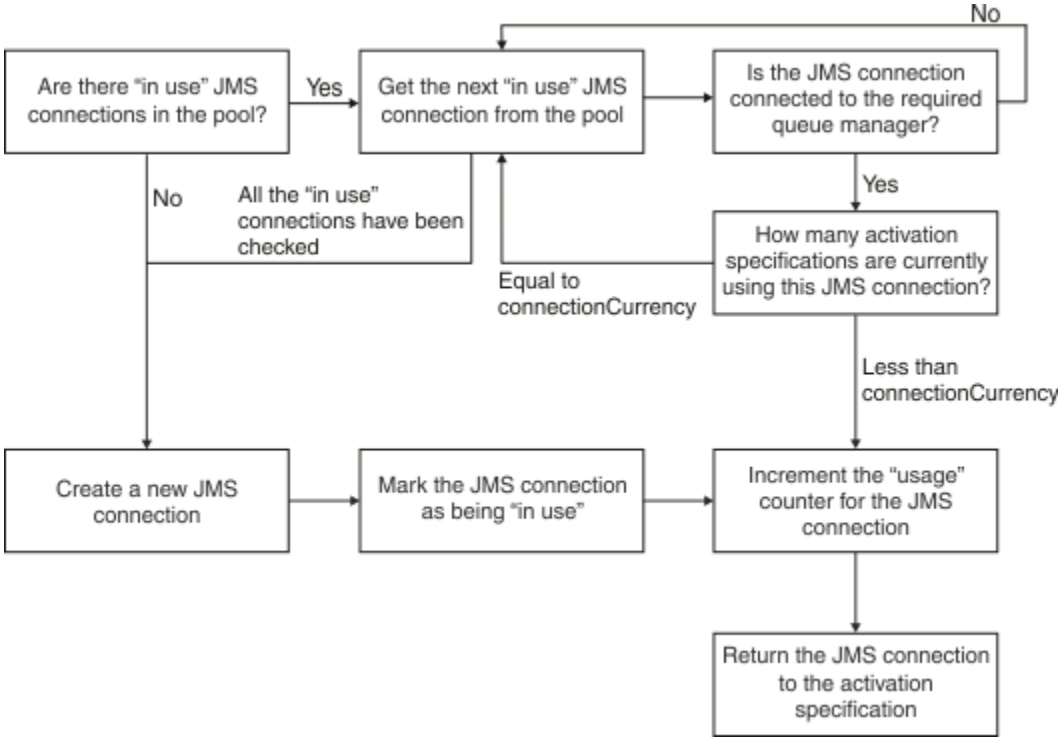


Figure 43. WebSphere Application Server versions 7.5 et 8.0 - comment fonctionne le pool de connexions

La Figure 44, à la page 316 décrit comment le pool de connexions fonctionne pour WebSphere Application Server version 8.5.

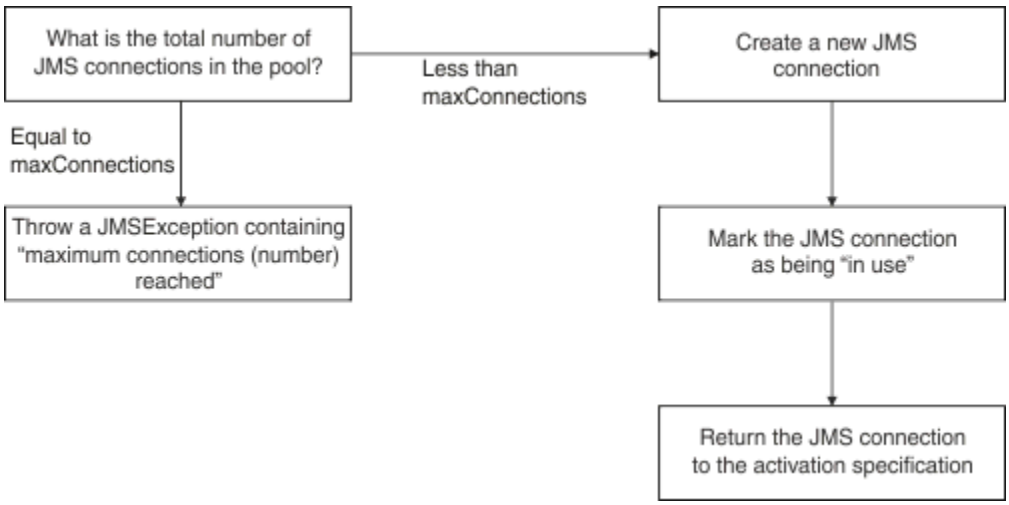


Figure 44. WebSphere Application Server versions 8.5 - comment fonctionne le pool de connexions

Comment les ports d'écoute de bean géré par message utilisent le pool de connexions

Supposons qu'un bean géré par message déployé sur un système WebSphere Application Server Network Deployment utilise IBM MQ comme fournisseur JMS. Le bean géré par message est déployé sur un port d'écoute qui utilise une fabrique de connexions appelée par exemple `jms/CF1` dont la propriété *maximum connections* a la valeur 2, ce qui signifie que deux connexions seulement peuvent être créées à tout moment à partir de cette fabrique.

Lorsque le port d'écoute démarre, il tente de créer une connexion à IBM MQ à l'aide de la fabrique de connexions `jms/CF1`.

Pour cela, le port demande une connexion au gestionnaire de connexions. Comme c'est la première fois que la fabrique de connexions `jms/CF1` est utilisée, il n'existe pas de connexion dans le pool de connexions libres `jms/CF1`. Le gestionnaire de connexions en crée une autre, par exemple `c1`. Sachez que cette connexion existe pendant toute la durée de vie du port d'écoute.

Etudions maintenant la situation dans laquelle vous arrêtez le port d'écoute à l'aide de la console d'administration WebSphere Application Server. Dans ce cas, le gestionnaire de connexions prend la connexion et la replace dans le pool libre. Toutefois, la connexion à IBM MQ reste ouverte.

Si vous redémarrez le port d'écoute, ce dernier demande à nouveau au gestionnaire de connexions une connexion au gestionnaire de files d'attente. Comme il existe maintenant une connexion (`c1`) dans le pool libre, le gestionnaire de connexions retire cette connexion du pool et la rend disponible sur le port d'écoute.

Considérons maintenant qu'il existe un deuxième bean géré par message déployé sur le serveur d'applications qui utilise un port d'écoute différent.

Supposons que vous essayez de démarrer un troisième port d'écoute qui est également configuré pour utiliser la fabrique de connexions `jms/CF1`. Le troisième port d'écoute demande une connexion auprès du gestionnaire de connexions qui recherche `jms/CF1` dans le pool libre et constate que ce dernier est vide. Il détermine ensuite le nombre de connexions qui ont déjà été créées à partir de la fabrique `jms/CF1`.

Etant donné que la propriété Maximum connections pour `jms/CF1` est définie sur 2 et que vous avez déjà créé deux connexions à partir de cette fabrique, le gestionnaire de connexions attend 180 secondes (la valeur par défaut de la propriété Connection timeout) pour mettre à disposition une connexion.

Toutefois, si vous arrêtez le premier port d'écoute, sa connexion `c1` est placée dans le pool libre pour `jms/CF1`. Le gestionnaire de connexions extrait cette connexion et l'affecte au troisième port d'écoute.

Si vous tentez maintenant de redémarrer le premier port d'écoute, ce dernier doit attendre que l'un des autres ports s'arrête avant de redémarrer. Si aucun des ports d'écoute n'est arrêté dans un délai de 180 secondes, le premier port d'écoute reçoit une erreur `ConnectionWaitTimeoutException` et s'arrête.

Comment les applications qui font appel à la fonction de messagerie sortante utilisent le pool de connexions

Pour cette option, supposons qu'un seul EJB, `EJB1`, installé sur le serveur d'applications, est appelé. Le bean implémente une méthode appelée `sendMessage()` en :

- Création d'une connexion JMS à IBM MQ à partir d'une fabrique `jms/CF1`à l'aide de `connectionFactory.createConnection()`.
- créant une session JMS à partir de la connexion,
- créant un expéditeur de message à partir de la session,
- envoyant un message,
- fermant l'expéditeur,
- fermant la session,
- fermant la connexion en appelant `connection.close()`.

Supposons que le pool libre pour la fabrique `jms/CF1` est vide. Lorsque l'EJB est appelé pour la première fois, le bean tente de créer une connexion à IBM MQ à partir de la fabrique `jms/CF1`. Comme le pool libre pour la fabrique est vide, le gestionnaire de connexions crée une connexion à l'`EJB1`.

Juste avant sa fermeture, la méthode appelle `connection.close()`. Au lieu de fermer `c1`, le gestionnaire de connexions prend la connexion et la place dans le pool libre pour `jms/CF1`.

La prochaine fois que `sendMessage()` est appelé, la méthode `connectionFactory.createConnection()` renvoie `c1` à l'application.

Supposons qu'une deuxième instance de l'EJB s'exécute en même temps que la première. Lorsque les deux instances appellent `sendMessage()`, deux connexions sont créées à partir de la fabrique de connexions `jms/CF1`.

Supposons maintenant qu'une troisième instance du bean est créée. Lorsque le troisième bean appelle `sendMessage()`, la méthode appelle `connectionFactory.createConnection()` pour créer une connexion à partir de `jms/CF1`.

Cependant, deux connexions sont créées à partir de `jms/CF1`, ce qui est égal à la valeur de la propriété Maximum connections pour cette fabrique. Ainsi, la méthode `createConnection()` attend 180 secondes (valeur par défaut de la propriété Connection timeout) qu'une connexion soit disponible.

Cependant, si la méthode `sendMessage()` pour le premier EJB appelle `connection.close()` et se ferme, la connexion qu'il utilisait, `c1`, est replacée dans le pool de connexions libres. Le gestionnaire de connexions extrait la connexion du pool libre et l'affecte au troisième EJB. L'appel de ce bean à `connectionFactory.createConnection()` génère un retour, ce qui permet à la méthode `sendMessage()` de se terminer.

Ports d'écoute de bean géré par message et EJB utilisant le même pool de connexions

Les deux exemples précédents indiquent comment les ports d'écoute et les EJB peuvent utiliser le pool de connexions de manière isolée. Cependant, un port d'écoute et un EJB peuvent s'exécuter sur le même serveur d'applications et créer des connexions JMS à l'aide de la même fabrique de connexions.

Vous devez prendre en compte les implications de cette situation

L'élément clé à retenir est que la fabrique de connexions est partagée entre le port d'écoute et l'EJB.

Par exemple, supposons qu'un port d'écoute et un EJB s'exécutent simultanément. Ils utilisent tous les deux la fabrique ce connexions `jms/CF1`, ce qui signifie que le nombre maximal de connexions défini par la propriété Maximum connections pour cette fabrique est atteint.

Si vous tentez de démarrer un autre port d'écoute ou une autre instante d'un EJB, il faut attendre qu'une connexion soit replacée dans le pool de connexions libres pour `jms/CF1`.

Unités d'exécution de maintenance de pool de connexions libres
Une unité d'exécution de maintenance de pool est associée à chaque pool de connexions libres. Elle surveille le pool libre pour s'assurer que les connexions qu'il contient sont toujours valides.

Si l'unité d'exécution de maintenance de pool décide qu'une connexion du pool libre doit être supprimée, elle ferme physiquement la connexion JMS à IBM MQ.

Comment fonctionne l'unité d'exécution de maintenance de pool ?

Le comportement de l'unité d'exécution de maintenance de pool dépend de la valeur des quatre propriétés du pool de connexions :

Aged timeout (délai d'expiration inutilisé)
Durée pendant laquelle une connexion reste ouverte.

Minimum connections (nombre minimal de connexions)
Nombre minimal de connexions qui sont laissées dans le pool libre d'une fabrique de connexions par le gestionnaire de connexions.

Reap time (intervalle de régulation)
Fréquence d'exécution de l'unité d'exécution de maintenance de pool.

Unused timeout (dépassement du délai d'attente d'inutilisation)
Durée pendant laquelle une connexion reste dans le pool libre avant d'être fermée.

Par défaut, l'unité d'exécution de maintenance de pool s'exécute toutes les 180 secondes, bien que cette valeur puisse être modifiée en définissant la propriété **Reap time** du pool de connexions.

L'unité d'exécution de maintenance examine chaque connexion du pool, détermine depuis combien de temps elle se trouve dans le pool et la durée qui s'est écoulée depuis qu'elle a été créée et utilisée pour la dernière fois.

Si la connexion n'a pas été utilisée pendant une période supérieure à la valeur de la propriété **Unused timeout** pour le pool de connexions, l'unité d'exécution de maintenance vérifie le nombre de connexions actuellement dans le pool libre. Si ce nombre est :

- supérieur à la valeur de **Minimum connections**, le gestionnaire de connexions ferme la connexion ;
- égal à la valeur de **Minimum connections**, la connexion n'est pas fermée et reste dans le pool libre.

La valeur par défaut de la propriété **Minimum connections** est *1*, ce qui signifie que, pour des raisons de performance, le gestionnaire de connexions tente toujours de conserver au moins une connexion dans le pool libre.

La propriété **Unused timeout** a par défaut la valeur 1800 secondes. Par défaut, si une connexion est replacée dans le pool libre et n'est pas utilisée à nouveau pendant au moins 1800 secondes, elle est fermée à condition qu'il reste au moins une connexion dans le pool libre.

Cette procédure empêche les connexions inutilisées de devenir périmées. Pour désactiver cette fonction, définissez la propriété **Unused timeout** sur zéro.

Si une connexion se trouve dans le pool libre et si la durée écoulée depuis sa création est supérieure à la durée indiquée dans la propriété **Aged timeout** pour le pool de connexions, elle est fermée quelle que soit la durée écoulée depuis sa dernière utilisation.

Par défaut, la propriété **Aged timeout** a la valeur zéro, ce qui signifie que l'unité d'exécution de maintenance n'effectue jamais cette vérification. Les connexions qui existent depuis plus longtemps que la durée indiquée dans la propriété **Aged timeout** sont supprimées quel que soit le nombre de connexions qui restent dans le pool libre. Sachez que la propriété **Minimum connections** n'a pas d'effet dans cette situation.

Désactivation de l'unité d'exécution de maintenance de pool

D'après la description précédente, vous pouvez constater que l'unité d'exécution de maintenance de pool effectue un travail important lorsqu'elle est active, en particulier s'il existe un grand nombre de connexions dans le pool libre de la fabrique de connexions.

Par exemple, supposons qu'il existe trois fabriques de connexions JMS, ayant chacune une propriété **Maximum connections** définie à la valeur 10. Toutes les 180 secondes, les trois unités d'exécution de maintenance de pool deviennent actives et examinent les pools libres pour chaque fabrique de connexions respectivement. Si les pools libres comportent un grand nombre connexions, les unités d'exécution de maintenance ont beaucoup de travail à effectuer, ce qui a un impact important sur les performances.

Vous pouvez désactiver l'unité d'exécution de maintenance de pool pour un pool de connexions libres donné en définissant sa propriété **Reap time** sur la valeur zéro.

La désactivation de l'unité d'exécution de maintenance signifie que les connexions ne sont jamais fermées, même si le délai indiqué dans **Unused timeout** est écoulé. Toutefois, les connexions peuvent être fermées si le délai indiqué dans **Aged timeout** est dépassé.

Lorsqu'une application a terminé une connexion, le gestionnaire de connexions vérifie la durée d'existence de la connexion et, si cette période est supérieure à la valeur de la propriété **Aged timeout** , le gestionnaire de connexions ferme la connexion au lieu de la renvoyer au pool libre.

Implications transactionnelles de la propriété Aged timeout (délai d'expiration inutilisé)

Comme indiqué dans la section précédente, la propriété **Aged timeout** indique la durée pendant laquelle une connexion au fournisseur JMS reste ouverte avant que le gestionnaire de connexions ne la ferme.

La valeur par défaut de la propriété **Aged timeout** est zéro, ce qui signifie que la connexion ne sera jamais fermée parce que trop ancienne. Vous devez laisser la propriété **Aged timeout** à cette valeur, car l'activation de **Aged timeout** peut avoir des implications transactionnelles lors de l'utilisation de JMS dans les EJB.

Dans JMS, l'unité d'une transaction est une JMS session , créée à partir d'une JMS connexion . Il s'agit de la JMS session qui est inscrite dans les transactions et non de la JMS connexion .

En raison de la conception du serveur d'applications, les connexions JMS peuvent être fermées car la durée indiquée dans **Aged timeout** est écoulée, même si les sessions JMS créées à partir de cette connexion sont impliquées dans une transaction.

La fermeture d'une connexion JMS entraîne l'annulation du travail transactionnel restant sur les sessions JMS, comme indiqué dans la spécification JMS. Toutefois, le serveur d'applications n'est pas informé que les sessions JMS créées à partir de la connexion ne sont plus valides. Lorsque le serveur tente d'utiliser la session pour valider ou annuler une transaction, une erreur `IllegalStateException` se produit.

Important : Si vous voulez utiliser **Aged timeout** avec des connexions JMS à partir d'EJB, vérifiez que le travail JMS est validé explicitement dans la session JMS avant que la méthode d'EJB qui effectue les opérations JMS ne se ferme.

Exemples d'unité d'exécution de maintenance de pool
Utilisation de l'exemple EJB (Enterprise JavaBean) pour comprendre le fonctionnement de l'unité d'exécution de maintenance de pool. Sachez que vous pouvez également utiliser des beans gérés par message et des ports d'écoute car tout ce dont vous avez besoin c'est d'obtenir des connexions dans le pool libre.

Voir «Comment les applications qui font appel à la fonction de messagerie sortante utilisent le pool de connexions», à la page 317 pour plus de détails sur la méthode `sendMessage()` .

Vous avez configuré la fabrique de connexions avec les valeurs suivantes :

- **Reap time** (intervalle de régulation) à sa valeur par défaut de 180 secondes

- **Aged timeout** (délai d'expiration inutilisé) à sa valeur par défaut de zéro seconde
- **Unused timeout** (dépassement du délai d'attente d'inutilisation) défini sur 300 secondes

Après le démarrage du serveur d'applications, la méthode `sendMessage()` est appelée.

La méthode crée une connexion appelée `c1` à l'aide de la fabrique `jms/CF1`, elle utilise cette fabrique pour envoyer un message et appelle ensuite `connection.close()`, ce qui place `c1` dans le pool libre.

Au bout de 180 secondes, l'unité d'exécution de maintenance de pool démarre et recherche le pool de connexions libres `jms/CF1`. Comme il existe une connexion libre `c1` dans le pool, l'unité d'exécution de maintenance examine l'heure à laquelle la connexion a été replacée et la compare à l'heure actuelle.

180 secondes se sont écoulées depuis que la connexion a été placée dans le pool libre, durée inférieure à la valeur de la propriété **Unused timeout** pour `jms/CF1`. L'unité d'exécution de maintenance ne touche donc pas à la connexion.

180 secondes après, l'unité d'exécution de maintenance de pool s'exécute à nouveau. L'unité d'exécution de maintenance localise la connexion `c1` et détermine qu'elle se trouve dans le pool depuis 360 secondes, durée plus longue que celle définie par **Unused timeout**. Le gestionnaire de connexions ferme donc la connexion.

Si vous exécutez à nouveau la méthode `sendMessage()` lorsque l'application appelle `connectionFactory.createConnection()`, le gestionnaire de connexions crée une connexion à IBM MQ car le pool de connexions libres pour la fabrique de connexions est vide.

L'exemple précédent indique comment l'unité d'exécution de maintenance utilise les propriétés **Reap time** et **Unused timeout** pour éviter les connexions périmées lorsque la propriété **Aged timeout** a la valeur zéro.

Comment fonctionne la propriété **Aged timeout** ?

Dans l'exemple suivant, les propriétés ont les valeurs suivantes :

- **Aged timeout** définie sur 300 secondes
- **Unused timeout** définie sur zéro.

Vous appelez la méthode `sendMessage()` qui tente de créer une connexion à partir de la fabrique de connexions `jms/CF1`.

Comme le pool libre de cette fabrique est vide, le gestionnaire de files d'attente crée une connexion, `c1`, et la renvoie à l'application. Lorsque `sendMessage()` appelle `connection.close()`, `c1` est replacée dans le pool de connexions libres.

180 secondes plus tard, l'unité d'exécution de maintenance de pool s'exécute. L'unité d'exécution localise `c1` dans le pool de connexions libres et vérifie depuis combien de temps elle a été créée. La connexion existe depuis 180 secondes, durée inférieure à celle indiquée dans **Aged timeout**. L'unité d'exécution de maintenance de pool ne touche pas à la connexion et se remet au repos.

60 secondes après, `sendMessage()` est appelée à nouveau. Cette fois-ci lorsque la méthode appelle `connectionFactory.createConnection()`, le gestionnaire de connexions découvre que la connexion `c1` est disponible dans le pool libre pour `jms/CF1`. Le gestionnaire de connexions extrait la connexion `c1` du pool libre et l'affecte à l'application.

La connexion est replacée dans le pool libre lorsque la méthode `sendMessage()` se ferme. 120 secondes après, l'unité d'exécution de maintenance de pool se réveille à nouveau, examine le contenu du pool libre pour `jms/CF1` et détecte `c1`.

Bien que la connexion n'ait été utilisée qu'il y a 120 secondes, l'unité d'exécution de maintenance de pool ferme la connexion car cette dernière existe depuis 360 secondes au total, durée supérieure à la valeur de 300 secondes que vous avez attribuée à la propriété **Aged timeout**.

Comment la propriété Minimum connections affecte l'unité d'exécution de maintenance de pool ?

En reprenant l'exemple «Comment les ports d'écoute de bean géré par message utilisent le pool de connexions», à la page 316, deux beans gérés par message sont déployés sur le serveur d'applications, chacun d'eux utilisant un port d'écoute différent.

Chaque port d'écoute est configuré pour utiliser la fabrique de connexions `jms/CF1` que vous avez configurée avec les propriétés suivantes :

- **`Unused timeout`** définie sur 120 secondes
- **`Reap time`** définie sur 180 secondes
- **`Minimum connections`** définie sur 1

Supposons que le premier port d'écoute est arrêté et que sa connexion `c1` est placée dans le pool libre. 180 secondes après, l'unité d'exécution de maintenance de pool se réveille, examine le contenu du pool libre pour `jms/CF1` et détecte que `c1` se trouve dans le pool libre depuis une durée plus longue que la valeur attribuée à la propriété **`Unused timeout`** pour la fabrique de connexions.

Toutefois, avant de fermer `c1`, l'unité d'exécution de maintenance de pool calcule le nombre de connexions qui resteraient dans le pool si cette connexion était éliminée. Etant donné que `c1` est la seule connexion du pool de connexions libres, le gestionnaire de connexions ne la ferme pas car sinon le nombre de connexions restant dans le pool libre serait inférieur à la valeur attribuée à la propriété **`Minimum connections`**.

Supposons maintenant que le deuxième port d'écoute est arrêté. Le pool de connexions libress contient maintenant deux connexions libres : `c1` et `c2`.

180 secondes après, l'unité d'exécution de maintenance de pool s'exécute à nouveau. A cet instant, `c1` se trouve dans le pool de connexion libre depuis 360 secondes et `c2` depuis 180 secondes.

L'unité d'exécution de maintenance de pool vérifie `c1` et découvre qu'elle figure dans le pool depuis plus longtemps que la durée indiquée dans la propriété **`Unused timeout`**.

L'unité d'exécution vérifie ensuite le nombre de connexions figurant dans le pool libre et le compare à la valeur attribuée à la propriété **`Minimum connections`**. Etant donné que le pool contient deux connexions, et que la propriété **`Minimum connections`** est définie sur 1, le gestionnaire de connexions ferme `c1`.

L'unité d'exécution de maintenance examine maintenant `c2`. Il se trouve également dans le pool de connexions libres depuis plus longtemps que la valeur de la propriété **`Unused timeout`** . Toutefois, si `c2` était fermée, le nombre de connexions dans le pool de libre serait inférieur à la valeur indiquée dans la propriété Minimum connections. Le gestionnaire de connexions ne touche donc pas à `c2`.

Connexions JMS et IBM MQ
Informations relatives à l'utilisation d'IBM MQ en tant que fournisseur JMS.

Utilisation du transport de liaisons

Si une fabrique de connexions a été configurée pour utiliser le transport de liaisons, chaque connexion JMS établit une conversation (également appelée **hconn**) avec IBM MQ. La conversation utilise la communication interprocessus (ou la mémoire partagée) pour communiquer avec le gestionnaire de files d'attente.

Utilisation du transport client

Lorsqu'une fabrique de connexions du fournisseur de messagerie IBM MQ a été configurée pour utiliser le transport client, chaque connexion créée à partir de cette fabrique établira une nouvelle conversation (également appelée **hconn**) avec IBM MQ.

Pour les fabriques de connexions qui se connectent à un gestionnaire de files d'attente à l'aide du fournisseur de messagerie IBM MQ en mode normal, plusieurs connexions JMS créées à partir de la

fabrique de connexions peuvent partagent une connexion TCP/IP à IBM MQ. Pour plus d'informations, voir «Partage d'une connexion TCP/IP dans IBM MQ classes for JMS», à la page 324.

Pour déterminer le nombre maximal de canaux client utilisés par les connexions JMS à un instant donné, ajoutez la valeur de la propriété *Nbre maximal de connexions* pour toutes les fabriques de connexions qui pointent vers le même gestionnaire de files d'attente.

Par exemple, supposons que deux fabriques de connexions, `jms/CF1` et `jms/CF2`, ont été configurées pour se connecter au même gestionnaire de files d'attente IBM MQ à l'aide du même canal IBM MQ.

Ces fabriques utilisent les propriétés de pool de connexions par défaut ce qui signifie que la propriété *Nbre maximal de connexions* a la valeur 10. Si toutes les connexions sont utilisées simultanément à partir de jms/CF1 et jms/CF2, 20 conversations seront établies entre le serveur d'applications et IBM MQ.

Si la fabrique de connexions se connecte au gestionnaire de files d'attente à l'aide du fournisseur de messagerie IBM MQ en mode normal, le nombre maximal de connexions TCP/IP qui peuvent exister entre le serveur d'applications et le gestionnaire de files d'attente pour ces fabriques de connexions est le suivant :

```
20/the value of SHARECNV for the IBM MQ channel
```

Si la fabrique de connexions est configurée pour se connecter à l'aide du fournisseur de messagerie IBM MQ en mode normal, le nombre maximal de connexions TCP/IP entre le serveur d'applications et IBM MQ pour ces fabriques de connexions est de 20 (une pour chaque connexion JMS dans les pools de connexions pour les deux fabriques).

Concepts associés

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85
IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

Mise en pool d'objets dans un environnement Java SE

Avec Java SE (ou avec une autre infrastructure telle que Spring), les modèles de programmation sont extrêmement flexibles. Par conséquent, une stratégie de mise en commun unique ne convient pas à tous. Vous devriez examiner s'il existe un cadre en place qui pourrait faire n'importe quelle forme de regroupement, par exemple, Spring.

Dans le cas contraire, la logique de l'application pourrait prendre en compte cette situation. Demandez-vous quelle est la complexité de l'application elle-même? Il est préférable de comprendre l'application et ce qu'elle exige de la connectivité au système de messagerie. Les applications sont souvent écrites également dans leur propre code encapsuleur autour de l'API JMS de base.

Bien que cela puisse être une approche très sensée et cacher la complexité, il convient de garder à l'esprit qu'elle peut poser des problèmes. Par exemple, une méthode `getMessage()` générique, souvent appelée, ne doit pas se contenter d'ouvrir et de fermer des consommateurs.

Points à prendre en considération:

- Combien de temps l'application aura-t-elle besoin d'accéder à IBM MQ? Tout le temps, ou juste de temps en temps.
- À quelle fréquence les messages seront-ils envoyés? Moins la fréquence est élevée, plus une connexion unique à IBM MQ peut être partagée.
- Une exception de connexion interrompue est généralement un signe de la nécessité de recréer une connexion en pool. Qu'en est-il de:
  - Exceptions de sécurité ou hôte non disponible
  - Exceptions de file d'attente saturée
- Si une exception de connexion interrompue se produit, que se passe-t-il pour les autres connexions libres dans le pool? Devraient-ils être fermés et recréés?

- Si TLS est utilisé, par exemple, combien de temps souhaitez-vous qu'une seule connexion reste ouverte?
- Comment une connexion en pool s'identifiera-t-elle de sorte qu'un administrateur de gestionnaire de files d'attente puisse repérer la connexion et la suivre.

Vous devez prendre en compte tous les objets JMS pour la mise en pool et mettre en pool cet objet chaque fois qu'il est possible de le faire. Les objets sont les suivants:

- Connexions JMS
- Session
- Contextes
- Producteurs et consommateurs de tous types

Lors de l'utilisation du transport client, les connexions, les sessions et les contextes JMS utilisent des sockets lorsqu'ils communiquent avec le gestionnaire de files d'attente IBM MQ . La mise en pool de ces objets permet d'économiser sur le nombre de connexions IBM MQ entrantes (`hConns`) au gestionnaire de files d'attente et de réduire le nombre d'instances de canal.

L'utilisation du transport de liaisons pour le gestionnaire de files d'attente supprime entièrement la couche réseau. Cependant, de nombreuses applications utilisent le transport client pour fournir une configuration à haute disponibilité et équilibrée de la charge de travail.

Les producteurs et les consommateurs JMS ouvrent des destinations sur le gestionnaire de files d'attente. Si moins de files d'attente ou de rubriques sont ouvertes et que plusieurs parties de l'application utilisent ces objets, il peut être utile de les mettre en pool.

Du point de vue de IBM MQ , ce processus sauvegarde une séquence d'opérations MQOPEN et MQCLOSE.

Connexions, sessions et contextes

Ces objets encapsulent tous les descripteurs de connexion IBM MQ au gestionnaire de files d'attente et sont générés à partir d'un `ConnectionFactory`. Vous pouvez ajouter une logique à une application pour limiter le nombre de connexions et d'autres objets créés à partir d'une seule fabrique de connexions à un nombre spécifique.

Vous pouvez utiliser une structure de données simple dans l'application pour contenir les connexions créées. Le code d'application qui doit utiliser l'une de ces structures de données peut *extraire* un objet à utiliser.

Prenez en compte les facteurs suivants:

- Quand les connexions devraient-elles être supprimées du pool? En règle générale, créez un programme d'écoute des exceptions sur la connexion. Lorsque ce programme d'écoute est appelé pour traiter une exception, vous devez recréer la connexion et les sessions créées à partir de cette connexion.
- Si une table de définition de canal du client est utilisée pour l'équilibrage de charge, les connexions peuvent être destinées à des gestionnaires de files d'attente différents. Cela peut s'appliquer aux exigences de mise en pool.

N'oubliez pas que la spécification JMS indique qu'il s'agit d'une erreur de programmation pour que plusieurs unités d'exécution accèdent simultanément à une session ou à un contexte. Le code IBM MQ JMS tente d'être rigoureux dans sa gestion des unités d'exécution. Toutefois, vous devez ajouter une logique à l'application pour vous assurer qu'un objet de session ou de contexte n'est utilisé que par une seule unité d'exécution à la fois.

Producteurs et consommateurs

Chaque fournisseur et destinataire créé ouvre une destination sur le gestionnaire de files d'attente. Si la même destination doit être utilisée pour une variété de tâches, il est logique de conserver les objets de consommateur ou de producteur ouverts. Ne fermez l'objet que lorsque tout le travail est terminé.

Bien que l'ouverture et la fermeture d'une destination soient de courtes opérations, si elles sont effectuées fréquemment, le temps peut s'ajouter.

La portée de ces objets se trouve dans la session ou le contexte à partir duquel ils sont créés. Par conséquent, ils doivent se trouver dans cette portée. En général, les applications sont écrites de telle sorte que cela est assez simple à faire.

Surveillance

Comment les applications surveillent-elles leurs pools d'objets? La réponse à cette question est largement déterminée par la complexité de la solution de mise en commun mise en oeuvre.

Si vous envisagez une implémentation de mise en pool JavaEE, il existe un grand nombre d'options, notamment:

- Taille actuelle des pools
- Temps que les objets y ont passé
- Nettoyage des piscines
- Actualisation des connexions

Vous devez également prendre en compte la façon dont une session unique réutilisée apparaît sur le gestionnaire de files d'attente. Il existe des propriétés de fabrique de connexions permettant d'identifier l'application (telle que `appName`) qui peut être utile.

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85
IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

Partage d'une connexion TCP/IP dans IBM MQ classes for JMS
Plusieurs instances d'un canal MQI peuvent être créées pour partager une connexion TCP/IP unique.

Les applications qui s'exécutent dans le même environnement d'exécution Java et qui utilisent IBM MQ classes for JMS ou l'adaptateur de ressources IBM MQ pour se connecter à un gestionnaire de files d'attente à l'aide du transport CLIENT peuvent être utilisées pour partager une instance de canal.

Si un canal est défini avec le paramètre **SHARECNV** défini sur une valeur supérieure à 1, ce nombre de conversations peut partager une instance de canal. Pour activer une fabrique de connexions ou une spécification d'activation afin d'utiliser cette fonction, définissez la propriété **SHARECONVALLOWED** sur YES.

Chaque connexion JMS et session JMS créée par une application JMS crée sa propre conversation avec le gestionnaire de files d'attente.

Lorsqu'une spécification d'activation démarre, l'adaptateur de ressources IBM MQ démarre une conversation avec le gestionnaire de files d'attente pour la spécification d'activation à utiliser. Chaque session de serveur du pool de sessions de serveur associé à la spécification d'activation démarre également une conversation avec le gestionnaire de files d'attente.

L'attribut **SHARECNV** est une approche optimale pour le partage des connexions. Par conséquent, lorsqu'une valeur **SHARECNV** supérieure à 0 est utilisée avec IBM MQ classes for JMS, il n'est pas garanti qu'une nouvelle demande de connexion partage toujours une connexion déjà établie.

Mode de partage des connexions TCP/IP

Deux stratégies sont disponibles pour le partage des connexions TCP/IP:

La stratégie globale
Cette stratégie est la stratégie par défaut pour le partage des connexions TCP/IP. Toute connexion ou session JMS peut utiliser une conversation sur n'importe quelle connexion TCP/IP appropriée.

La pertinence est déterminée par des facteurs tels que l'adresse de l'hôte, le numéro de port, l'ID utilisateur et le mot de passe et les paramètres TLS/SSL.

Cette approche de partage des connexions TCP/IP réduit le nombre d'instances de canal utilisées, mais au prix de conflits d'accès à un pool global de connexions TCP/IP.

La stratégie CONNECTION

Avec cette stratégie, les instances de canal sont uniquement partagées entre les objets JMS associés. En particulier, lorsqu'une connexion JMS est créée, une instance de canal est créée pour cette dernière et des conversations supplémentaires sur cette instance de canal ne sont disponibles que pour les sessions JMS créées par cette connexion JMS .

Si le nombre de conversations créées est supérieur à celui spécifié par l'attribut SHARECNV, une nouvelle instance de canal est créée et ne peut être utilisée que par les sessions JMS créées par la connexion JMS d'origine.

Cette approche de partage des instances de canal réduit les conflits pour les conversations, au détriment de la nécessité potentielle d'un plus grand nombre d'instances de canal.

Spécification explicite d'une stratégie de partage d'instance de canal

V 9.4.0

Par défaut, la stratégie GLOBAL est utilisée si les applications ne sont pas reconnectables. Les applications reconnectables utilisent toujours la stratégie CONNECTION.

Pour les applications qui utilisent IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, la stratégie CONNECTION peut être activée pour les applications non reconnectables à l'échelle de l'application. Vous pouvez activer la stratégie CONNECTION en définissant la propriété système `com.ibm.mq.jms.channel.sharing` sur la valeur `CONNECTION`. Cette valeur n'est pas sensible à la casse et toute valeur autre que `CONNECTION` est ignorée.

Vous pouvez définir la propriété système `com.ibm.mq.jms.channel.sharing` de l'une des manières suivantes:

- Définissez la propriété dans le cadre de l'initialisation de la machine virtuelle Java à l'aide de l'option de ligne de commande "-D":

```
-Dcom.ibm.mq.jms.channel.sharing=CONNECTION
```

- Définissez la propriété avant toute utilisation de IBM MQ classes for JMS ou de IBM MQ classes for Jakarta Messaging à l'aide de `System.setProperty()`

Calcul du nombre d'instances de canal pour la stratégie de partage GLOBAL

Utilisez les formules suivantes pour déterminer le nombre maximal d'instances de canal créées par une application:

Spécifications d'activation

Nombre d'instances de canal = (*maxPoolDepth_value* + 1) / *SHARECNV_value*

Où *maxPoolDepth_value* est la valeur de la propriété **maxPoolDepth** et *SHARECNV_value* est la valeur de la propriété **SHARECNV** sur le canal utilisé par la spécification d'activation.

Autres applications JMS

Nombre d'instances de canal = (*jms_connections* + *jms_sessions*) / *SHARECNV_value*

Où *jms_connections* est le nombre de connexions créées par l'application, *jms_sessions* est le nombre de sessions JMS créées par l'application et *SHARECNV_value* est la valeur de la propriété **SHARECNV** sur le canal utilisé par la spécification d'activation.

Calcul du nombre d'instances de canal pour la stratégie de partage CONNECTION

Le nombre d'instances de canal dépend de la répartition des sessions JMS entre les connexions JMS de l'application.

Autorisez une conversation pour la connexion JMS et une conversation pour chaque session JMS sous cette connexion JMS , puis divisez par la valeur **SHARECNV** , en arrondissant vers le haut. Ce calcul indique les instances de canal requises par cette connexion JMS .

Le même principe peut être appliqué aux spécifications d'activation. La spécification d'activation doit être considérée comme une connexion JMS et la propriété **maxPoolDepth** comme le nombre de sessions JMS .

Exemples

Les exemples suivants montrent comment utiliser les formules pour calculer le nombre d'instances de canal créées sur un gestionnaire de files d'attente par des applications à l'aide de l'adaptateur de ressources IBM MQ classes for JMS ou IBM MQ .

JMS exemple d'application
Une connexion d'application JMS se connecte à un gestionnaire de files d'attente à l'aide du transport CLIENT et crée une connexion JMS et trois sessions JMS . Le canal utilisé par l'application pour se connecter au gestionnaire de files d'attente a la propriété **SHARECNV** définie sur la valeur 10. Lorsque l'application est en cours d'exécution, il y a quatre conversations entre l'application et le gestionnaire de files d'attente et une instance de canal. Les quatre conversations partagent toutes l'instance de canal.

Exemple de spécification d'activation
Une spécification d'activation se connecte à un gestionnaire de files d'attente à l'aide du transport CLIENT. La spécification d'activation est configurée avec la propriété **maxPoolDepth** définie sur 10. Le canal que la spécification d'activation est configurée pour utiliser a la propriété **SHARECNV** définie sur 10. Lorsque la spécification d'activation est en cours d'exécution et que 10 messages sont traités simultanément, le nombre de conversations entre la spécification d'activation et le gestionnaire de files d'attente est de 11 (10 conversations pour les sessions de serveur et une pour la spécification d'activation). Le nombre d'instances de canal utilisées par la spécification d'activation est 2.

Exemple de spécification d'activation
Une spécification d'activation se connecte à un gestionnaire de files d'attente à l'aide du transport CLIENT. La spécification d'activation est configurée avec la propriété **maxPoolDepth** définie sur 5. Le canal que la spécification d'activation est configurée pour utiliser a la propriété **SHARECNV** définie sur 0. Lorsque la spécification d'activation est en cours d'exécution et que 5 messages sont traités simultanément, le nombre de conversations entre la spécification d'activation et le gestionnaire de files d'attente est de 6 (cinq conversations pour les sessions de serveur et une pour la spécification d'activation). Le nombre d'instances de canal utilisées par la spécification d'activation est 6 car la propriété **SHARECNV** sur le canal est définie sur 0, chaque conversation utilise sa propre instance de canal.

Tâches associées

«Détermination du nombre de connexions TCP/IP créées entre WebSphere Application Server et IBM MQ», à la page 515
A l'aide de la fonction de partage de conversations, plusieurs conversations peuvent partager des instances de canal MQI, également appelée connexion TCP/IP.

Spécification d'une plage de ports pour les connexions client dans IBM MQ classes for JMS
Utilisez la propriété LOCALADDRESS pour spécifier une plage de ports auxquels votre application peut être liée.

Lorsqu'une application IBM MQ classes for JMS tente de se connecter à un gestionnaire de files d'attente IBM MQ en mode client, un pare-feu peut autoriser uniquement les connexions provenant de ports spécifiés ou d'une plage de ports. Dans cette situation, vous pouvez utiliser la propriété LOCALADDRESS d'un objet ConnectionFactory, QueueConnectionFactory ou TopicConnectionFactory pour spécifier un port ou une plage de ports auxquels l'application peut se lier.

Vous pouvez définir la propriété LOCALADDRESS à l'aide de l'outil d'administration IBM MQ JMS ou en appelant la méthode setLocalAddress () dans une application JMS . Voici un exemple de définition de la propriété à partir d'une application:

```
mqConnectionFactory.setLocalAddress("192.0.2.0(2000,3000)");
```

Lorsque l'application se connecte à un gestionnaire de files d'attente par la suite, elle se lie à une adresse IP locale et à un numéro de port compris entre 192.0.2.0(2000) et 192.0.2.0(3000).

Dans un système comportant plusieurs interfaces réseau, vous pouvez également utiliser la propriété LOCALADDRESS pour spécifier l'interface réseau à utiliser pour une connexion.

Pour une connexion en temps réel à un courtier, la propriété LOCALADDRESS est pertinente uniquement lorsque la multidiffusion est utilisée. Dans ce cas, vous pouvez utiliser la propriété pour spécifier l'interface réseau locale à utiliser pour une connexion, mais la valeur de la propriété ne doit pas contenir de numéro de port ou de plage de numéros de port.

Des erreurs de connexion peuvent se produire si vous limitez la plage de ports. Si une erreur se produit, une exception JMSException est émise avec une exception MQException imbriquée qui contient le code anomalie IBM MQ MQRC_Q_MGR_NOT_AVAILABLE et le message suivant:

```
Tentative de connexion socket refusée en raison des restrictions LOCAL_ADDRESS_PROPERTY
```

Une erreur peut se produire si tous les ports de la plage spécifiée sont utilisés ou si l'adresse IP, le nom d'hôte ou le numéro de port spécifié n'est pas valide (un numéro de port négatif, par exemple).

Etant donné que IBM MQ classes for JMS peut créer des connexions autres que celles requises par une application, vous devez toujours spécifier une plage de ports. En général, chaque session créée par une application requiert un port et IBM MQ classes for JMS peut nécessiter trois ou quatre ports supplémentaires. Si une erreur de connexion se produit, augmentez la plage de ports.

Le regroupement de connexions, qui est utilisé par défaut dans IBM MQ classes for JMS, peut avoir un effet sur la vitesse à laquelle les ports peuvent être réutilisés. Par conséquent, une erreur de connexion peut se produire lors de la libération des ports.

Compression de canal dans IBM MQ classes for JMS
Une application IBM MQ classes for JMS peut utiliser les fonctions IBM MQ pour compresser un en-tête de message ou des données.

La compression des données qui circulent sur un canal IBM MQ peut améliorer les performances du canal et réduire le trafic réseau. A l'aide de la fonction fournie avec IBM MQ, vous pouvez compresser les données qui circulent sur les canaux de transmission de messages et les canaux MQI. Quel que soit le type de canal, vous pouvez compresser les données d'en-tête et les données de message indépendamment les unes des autres. Par défaut, aucune donnée n'est compressée sur un canal.

Une application IBM MQ classes for JMS spécifie les techniques qui peuvent être utilisées pour compresser les données d'en-tête ou de message sur une connexion en créant un objet java.util.Collection . Chaque technique de compression est un objet Integer dans la collection, et l'ordre dans lequel l'application ajoute les techniques de compression à la collection est l'ordre dans lequel les techniques de compression sont négociées avec le gestionnaire de files d'attente lorsque l'application crée la connexion. L'application peut ensuite transmettre la collection à un objet ConnectionFactory en appelant la méthode setHdrCompList(), pour les données d'en-tête, ou la méthode setMsgCompList(), pour les données de message. Lorsque l'application est prête, elle peut créer la connexion.

Les fragments de code suivants illustrent l'approche décrite. Le premier fragment de code vous montre comment implémenter la compression des données d'en-tête:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(WMQConstants.WMQ_COMPHDR_SYSTEM));
.
.
.
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
```

```
.
.
connection = cf.createConnection();
```

Le deuxième fragment de code vous montre comment implémenter la compression des données de message:

```
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();
```

Dans le second example, les techniques de compression sont négociées dans l'ordre RLE, puis ZLIBHIGH, lors de la création de la connexion. La technique de compression sélectionnée ne peut pas être modifiée pendant la durée de vie de l'objet de connexion. Pour utiliser la compression sur une connexion, les méthodes setHdrCompList() et setMsgCompList() doivent être appelées avant la création de l'objet Connection.

Insertion de messages de manière asynchrone dans IBM MQ classes for JMS
Normalement, lorsqu'une application envoie des messages à une destination, elle doit attendre que le gestionnaire de files d'attente confirme qu'elle a traité la demande. Vous pouvez améliorer les performances de la messagerie dans certaines circonstances en choisissant plutôt d'insérer des messages de manière asynchrone. Lorsqu'une application insère un message de manière asynchrone, le gestionnaire de files d'attente ne renvoie pas la réussite ou l'échec de chaque appel, mais vous pouvez rechercher les erreurs régulièrement.

Le fait qu'une destination renvoie le contrôle à l'application, sans déterminer si le gestionnaire de files d'attente a reçu le message en toute sécurité, dépend des propriétés suivantes:

La propriété de destination JMS PUTASYNCALLOWED (nom abrégé-PAALD).
PUTASYNCALLOWED contrôle si les applications JMS peuvent insérer des messages de manière asynchrone, si cette option est autorisée par la file d'attente ou la rubrique sous-jacente représentée par la destination JMS .

La propriété de file d'attente ou de rubrique IBM MQ DEFPRESP (type de réponse d'insertion par défaut).
DEFPRESP indique si les applications qui placent des messages dans la file d'attente ou qui publient des messages dans la rubrique peuvent utiliser la fonctionnalité d'insertion asynchrone.

Le tableau suivant présente les valeurs possibles pour les propriétés PUTASYNCALLOWED et DEFPRESP, ainsi que les combinaisons de valeurs utilisées pour activer la fonctionnalité d'insertion asynchrone:

Tableau 46. Comment les propriétés PUTASYNCALLOWED et DEFPRESP se combinent pour déterminer si les messages sont placés dans une destination de manière asynchrone.

Propriété de file d'attente IBM MQ	**PUTASYNCALLOWED = NON**	**PUTASYNCALLOWED = OUI**	**Fonctionnalité d'insertion asynchrone activée**
DEFPRESP=SYNC	La fonctionnalité d'insertion asynchrone n'est pas activée	Fonctionnalité d'insertion asynchrone activée	PUTASYNCALLOWED = AS_DEST, AS_Q_DEF ou AS_T_DEF
DEFPRESP=ASYNC	La fonctionnalité d'insertion asynchrone n'est pas activée	Fonctionnalité d'insertion asynchrone activée	PUTASYNCALLOWED = AS_DEST, AS_Q_DEF ou AS_T_DEF

Vous pouvez modifier le comportement en spécifiant la propriété IBM MQ-JMS Destination pour indiquer "NO" ou "YES", comme indiqué dans le tableau, mais elle peut également être remplacée pour l'ensemble de la machine virtuelle Java à l'aide de la valeur **SystemProperty** de la machine virtuelle Java:

```
com.ibm.mq.cfg.Channels.Put1DefaultAlwaysSync=Y
```

Pour les messages envoyés dans une session transactionnelle, l'application détermine finalement si le gestionnaire de files d'attente a reçu les messages en toute sécurité lorsqu'il appelle `commit()`.

Si une application envoie des messages persistants dans une session transactionnelle et qu'un ou plusieurs des messages ne sont pas reçus en toute sécurité, la validation de la transaction échoue et génère une exception. Toutefois, si une application envoie des messages non persistants dans une session transactionnelle et qu'un ou plusieurs des messages ne sont pas reçus en toute sécurité, la transaction est validée avec succès. L'application ne reçoit aucun commentaire indiquant que les messages non persistants ne sont pas arrivés en toute sécurité.

Pour les messages non persistants envoyés dans une session qui n'est pas transactionnelle, la propriété SENDCHECKCOUNT de l'objet *ConnectionFactory* indique le nombre de messages à envoyer, avant que IBM MQ classes for JMS ne vérifie que le gestionnaire de files d'attente a reçu les messages en toute sécurité.

Si une vérification détecte qu'un ou plusieurs messages n'ont pas été reçus en toute sécurité et que l'application a enregistré un programme d'écoute des exceptions avec la connexion, IBM MQ classes for JMS appelle la méthode `onException()` du programme d'écoute des exceptions pour transmettre une exception JMS à l'application.

L'exception JMS a le code d'erreur JMSWMQ0028 et ce code affiche le message suivant:

```
At least one asynchronous put message failed or gave a warning.
```

L'exception JMS comporte également une exception liée qui fournit plus de détails. La valeur par défaut de la propriété SENDCHECKCOUNT est zéro, ce qui signifie qu'aucune vérification de ce type n'est effectuée.

Cette optimisation est particulièrement avantageuse pour une application qui se connecte à un gestionnaire de files d'attente en mode client et qui doit envoyer une séquence de messages en succession rapide, mais qui ne nécessite pas de retour immédiat de la part du gestionnaire de files d'attente pour chaque message envoyé. Toutefois, une application peut toujours utiliser cette optimisation même si elle se connecte à un gestionnaire de files d'attente en mode liaisons, mais l'avantage attendu en termes de performances n'est pas aussi grand.

Remarque : Si vous utilisez un **MessageProducer** non identifié pour envoyer un message sous une transaction, les messages sont insérés par défaut dans la file d'attente à l'aide du mécanisme d'insertion asynchrone.

Cela peut être dû au fait que l'API JMS permet de créer le **MessageProducer** sans spécifier de destination, à l'aide de la syntaxe suivante:

```
javax.jms.MessageProducer messageProducer =  javax.jms.Session.createProducer(null);
  messageProducer.send(Destination destination, Message message, int deliveryMode, int priority, long
timeToLive);
```

Dans ce scénario, la destination JMS est fournie lorsque le message est envoyé et non pas avant l'heure à laquelle le **MessageProducer** est construit. En termes d'API IBM MQ , un MQPUT1 est émis pour insérer le message dans la file d'attente.

Si vous effectuez cette opération sous un point de synchronisation IBM MQ , ce qui signifie (dans la terminologie JMS) que le message est placé sous une transaction, soit à l'aide d'une session JMS transactionnelle, soit à l'aide d'une `XASession` , l'API IBM MQ classes for JMS passe à l'utilisation d'une insertion asynchrone.

Utilisation de la lecture anticipée avec IBM MQ classes for JMS
La fonctionnalité de lecture anticipée fournie par IBM MQ permet aux messages non persistants reçus en dehors d'une transaction d'être envoyés à IBM MQ classes for JMS avant qu'une application ne les demande. Le IBM MQ classes for JMS stocke les messages dans une mémoire tampon interne et les transmet à l'application lorsque celle-ci les demande.

Les applications IBM MQ classes for JMS qui utilisent `MessageConsumers` ou `MessageListeners` pour recevoir des messages d'une destination en dehors d'une transaction peuvent utiliser la fonctionnalité de lecture anticipée. L'utilisation de la lecture anticipée permet aux applications qui utilisent ces objets de bénéficier de meilleures performances lorsqu'elles reçoivent des messages.

Le fait qu'une application qui utilise `MessageConsumers` ou `MessageListeners` puisse utiliser la lecture anticipée dépend des propriétés suivantes:

Propriété de destination JMS READAHEADALLOWED (nom abrégé-RAALD).
READAHEADALLOWED détermine si les applications JMS peuvent utiliser la lecture anticipée lors de l'obtention ou de la consultation de messages non persistants en dehors d'une transaction, si la file d'attente ou la rubrique sous-jacente représentée par la destination JMS autorise cette option.

Propriété de file d'attente ou de rubrique IBM MQ DEFREADA (lecture anticipée par défaut).
DEFREADA indique si les applications qui reçoivent ou consultent des messages non persistants en dehors d'une transaction peuvent utiliser la lecture anticipée.

Le tableau suivant présente les valeurs possibles pour les propriétés READAHEADALLOWED et DEFREADA, ainsi que les combinaisons de valeurs utilisées pour activer la fonctionnalité de lecture anticipée:

Tableau 47. Comment les propriétés READAHEADALLOWED et DEFREADA se combinent pour déterminer si la lecture anticipée est utilisée lors de la réception ou de la consultation de messages non persistants en dehors d'une transaction.

Propriété de file d'attente IBM MQ	**READAHEADALLOWED = OUI**	**READAHEADALLOWED = NON**	**AS_DEST ou AS_Q_DEF ou AS_T_DEF**
DEFREADA = NON	Fonctionnalité de lecture anticipée activée	La fonctionnalité de lecture anticipée n'est pas activée	La fonctionnalité de lecture anticipée n'est pas activée
DEFREADA = OUI	Fonctionnalité de lecture anticipée activée	La fonctionnalité de lecture anticipée n'est pas activée	Fonctionnalité de lecture anticipée activée
DEFREADA = DESACTIVE	La fonctionnalité de lecture anticipée n'est pas activée	La fonctionnalité de lecture anticipée n'est pas activée	La fonctionnalité de lecture anticipée n'est pas activée

Si la fonctionnalité de lecture anticipée est activée, lorsqu'un `MessageConsumer` ou un `MessageListener` est créé par une application, IBM MQ classes for JMS crée une mémoire tampon interne pour la destination surveillée par `MessageConsumer` ou `MessageListener` . Il existe une mémoire tampon interne pour chaque `MessageConsumer` ou `MessageListener`. Le gestionnaire de files d'attente commence à envoyer des messages non persistants à IBM MQ classes for JMS lorsque l'application appelle l'une des méthodes suivantes:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

IBM MQ classes for JMS renvoie automatiquement le premier message à l'application, par l'appel de méthode effectué par l'application. Les autres messages non persistants sont stockés par IBM MQ classes for JMS dans la mémoire tampon interne créée pour la destination. Lorsque l'application

demande le traitement du message suivant, IBM MQ classes for JMS renvoie le message suivant dans la mémoire tampon interne.

IBM MQ classes for JMS demande davantage de messages non persistants au gestionnaire de files d'attente lorsque la mémoire tampon interne est vide.

La mémoire tampon interne utilisée par IBM MQ classes for JMS est supprimée lorsqu'une application ferme un `MessageConsumer`ou la session JMS à laquelle un `MessageListener` est associé.

Pour `MessageConsumers`, tous les messages non traités dans la mémoire tampon interne sont perdus.

Lors de l'utilisation de `MessageListeners`, ce qui arrive aux messages dans la mémoire tampon interne dépend de la propriété de destination JMS READAHEADCLOSEPOLICY (nom abrégé-RACP). La valeur par défaut de la propriété est DELIVER_ALL, ce qui signifie que la session JMS utilisée pour créer le `MessageListener` n'est pas fermée tant que tous les messages de la mémoire tampon interne n'ont pas été distribués à l'application. Si la propriété est définie sur DELIVER_CURRENT, la session JMS est fermée une fois que le message en cours a été traité par l'application et que tous les messages restants dans la mémoire tampon interne sont supprimés.

Publications conservées dans IBM MQ classes for JMS
Un client IBM MQ classes for JMS peut être configuré pour utiliser des publications conservées.

Un diffuseur de publications peut spécifier qu'une copie d'une publication doit être conservée pour pouvoir être envoyée aux futurs abonnés qui s'intéressent à la rubrique. Pour ce faire, dans IBM MQ classes for JMS , définissez la propriété d'entier JMS_IBM_RETAIN sur la valeur 1. Des constantes ont été définies pour ces valeurs dans l'interface com.ibm.msg.client.jms.JmsConstants . Par exemple, si vous avez créé un message *msg*, pour le définir en tant que publication conservée, utilisez le code suivant:

```
// set as a retained publication
msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION);
```

Vous pouvez maintenant envoyer le message normalement. JMS_IBM_RETAIN peut également être interrogé dans un message reçu. Il est donc possible de demander si un message reçu est une publication conservée.

Prise en charge de XA dans IBM MQ classes for JMS
JMS prend en charge les transactions compatibles XA dans les liaisons et les modes client avec un gestionnaire de transactions pris en charge dans un conteneur JEE .

Si vous avez besoin de la fonctionnalité XA dans un environnement de serveur d'applications, vous devez configurer votre application de manière appropriée. Reportez-vous à la documentation de votre serveur d'applications pour plus d'informations sur la configuration des applications en vue de l'utilisation de transactions réparties.

Un gestionnaire de files d'attente IBM MQ ne peut pas agir en tant que gestionnaire de transactions pour JMS.

Délai de distribution des messages JMS
Pour JMS 2.0 ou version ultérieure, vous pouvez spécifier un délai de distribution lors de l'envoi d'un message. Le gestionnaire de files d'attente ne distribue pas le message tant que le délai de distribution spécifié n'est pas écoulé.

Une application peut spécifier un délai de distribution en millisecondes, lorsqu'elle envoie un message, à l'aide de `MessageProducer.setDeliveryDelay(long deliveryDelay)` ou de `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Cette valeur est ajoutée à l'heure à laquelle le message est envoyé et indique l'heure au plus tôt à laquelle toute autre application peut obtenir ce message.

Le délai de distribution est implémenté à l'aide d'une seule file d'attente de transfert interne. Les messages dont le délai de distribution est différent de zéro sont placés dans cette file d'attente avec un en-tête qui indique le délai de distribution et des informations sur la file d'attente cible. Un composant du gestionnaire de files d'attente appelé processeur de délai de distribution surveille les messages de la

file d'attente de transfert. Lorsque le délai de distribution d'un message est écoulé, le message est retiré de la file d'attente de transfert et placé dans la file d'attente cible.

Clients de messagerie

L'implémentation IBM MQ du délai de distribution est disponible uniquement lorsque vous utilisez le client JMS . Les restrictions suivantes s'appliquent si vous utilisez le délai de livraison avec IBM MQ. Ces restrictions s'appliquent également à `MessageProducers` et `JMSProducers`, mais `JMSRuntimeExceptions` est émis dans le cas de `JMSProducers`.

- Toute tentative d'appel de `MessageProducer.setDeliveryDelay` avec une valeur différente de zéro lors de la connexion à un gestionnaire de files d'attente antérieur à IBM MQ 8.0génère un message `JMSException` avec un message MQRC_FUNCTION_NOT_SUPPORTED.
- Le délai de distribution n'est pas pris en charge pour les destinations en cluster dont la valeur **DEFBIND** est autre que MQBND_BIND_NOT_FIXED. Si un `MessageProducer` a un délai de distribution différent de zéro défini et qu'une tentative d'envoi à une destination qui ne répond pas à cette exigence est effectuée, l'appel génère un message `JMSException` avec un message MQRC_OPTIONS_ERROR.
- Toute tentative de définition d'une valeur de durée de vie inférieure à un délai de distribution différent de zéro précédemment spécifié, ou inversement, génère un message `JMSException` avec un message MQRC_EXPIRY_ERROR. Cette vérification est effectuée lors de l'appel des méthodes `setTimeToLive` , `setDeliveryDelay` ou `send` , en fonction de l'ensemble exact des opérations choisies.
- L'utilisation des publications conservées et le délai de distribution ne sont pas pris en charge. La tentative de publication d'un message avec un délai de distribution si ce message a été marqué comme conservé à l'aide de `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` génère un message `JMSException` avec un message MQRC_OPTIONS_ERROR.
- Le délai de distribution et le regroupement de messages ne sont pas pris en charge et toute tentative d'utilisation de cette combinaison génère un message `JMSException` avec un message MQRC_OPTIONS_ERROR.

Tout échec de l'envoi d'un message avec un délai de distribution entraîne l'émission par le client d'un `JMSException` avec un message d'erreur approprié, par exemple une file d'attente saturée. Dans certains cas, le message d'erreur peut s'appliquer à la destination cible et / ou à la file d'attente de transfert.

Remarque : IBM MQ permet aux applications qui placent un message dans une unité de travail d'obtenir à nouveau le même message même si l'unité de travail n'a pas été validée. Cette technique ne fonctionne pas avec le délai de distribution car le message n'est pas placé dans la file d'attente de transfert tant que l'unité de travail n'est pas validée et, par conséquent, n'a pas été envoyée à la destination cible.

Autorisation

IBM MQ effectue des vérifications d'autorisation sur la destination cible d'origine lorsque l'application envoie un message avec un délai de distribution différent de zéro. Si l'application n'est pas autorisée, l'envoi échoue. Lorsque le gestionnaire de files d'attente détecte que le délai de distribution d'un message est terminé, il ouvre la file d'attente cible. Aucun contrôle d'autorisation n'est effectué à ce stade.

SYSTEM.DDELAY.LOCAL.QUEUE

Une file d'attente système, SYSTEM.DDELAY.LOCAL.QUEUEest utilisé pour implémenter le délai de distribution.

- Multi Sous Multiplateformes, SYSTEM.DDELAY.LOCAL.QUEUE existe par défaut. La file d'attente système doit être modifiée de sorte que ses attributs MAXMSGL et MAXDEPTH soient suffisants pour la charge attendue.
- z/OS Sous IBM MQ for z/OS, SYSTEM.DDELAY.LOCAL.QUEUE est utilisée comme file d'attente de transfert pour les messages envoyés avec un délai de distribution aux files d'attente locales et

partagées. Sous z/OS, la file d'attente doit être créée et définie de sorte que ses attributs MAXMSGL et MAXDEPTH soient suffisants pour la charge attendue.

Lorsque cette file d'attente est créée, elle doit être sécurisée afin que le plus petit nombre possible d'utilisateurs y aient accès. L'accès à la file d'attente doit être réservé à des fins de maintenance et de surveillance.

Lorsqu'un message est envoyé par une application JMS avec un délai de distribution différent de zéro, il est inséré dans cette file d'attente avec un nouvel ID message. L'ID de message d'origine est placé dans l'ID de corrélation du message. Cet ID de corrélation permet à une application d'extraire un message de la file d'attente de transfert lorsque cela est nécessaire, par exemple si un délai de distribution important a été utilisé par erreur.

Considérations relatives à z/OS

Si votre système s'exécute sous z/OS, des considérations supplémentaires doivent être prises en compte si vous souhaitez utiliser le délai de livraison.

Si le délai de distribution doit être utilisé, la file d'attente système SYSTEM.DDELAY.LOCAL.QUEUE doit être défini. Il doit être défini avec une classe de stockage suffisante pour sa charge attendue et avec INDXTYPE (NONE) et MSGDLVSQ (FIFO) indiqués. Un exemple de définition de la file d'attente système est fourni, mis en commentaire, dans le JCL CSQ4INSG .

Files d'attente partagées

Le délai de distribution est pris en charge pour l'envoi de messages aux files d'attente partagées. Toutefois, une seule file d'attente de transfert privée est utilisée, que la file d'attente cible soit partagée ou non. Le gestionnaire de files d'attente propriétaire de cette file d'attente privée doit être en cours d'exécution pour envoyer le message différé à sa file d'attente partagée cible une fois le délai écoulé.

Remarque : Si un message non persistant est inséré avec un délai de distribution dans une file d'attente partagée et que le gestionnaire de files d'attente propriétaire de la file d'attente de transfert s'arrête, le message d'origine est perdu. Par conséquent, les messages non persistants envoyés avec un délai de distribution à une file d'attente partagée sont plus susceptibles d'être perdus que les messages non persistants envoyés sans délai de distribution à une file d'attente partagée.

Résolution de la destination cible

Si le message est envoyé à une file d'attente, la résolution est gérée deux fois: une fois par l'application JMS et une fois par le gestionnaire de files d'attente lorsqu'il extrait le message de la file d'attente de transfert et l'envoie à la file d'attente cible.

Les abonnements cible pour les publications sont mis en correspondance lorsque l'application JMS appelle la méthode d'envoi.

Si un message est envoyé avec une persistance ou une priorité en fonction de la définition de la file d'attente, la valeur est définie sur la première résolution et non sur la seconde.

Intervalle d'expiration

Le délai de distribution préserve le comportement de la propriété d'expiration, **MQMD.Expiry**. Par exemple, si un message a été inséré à partir d'une application JMS avec un intervalle d'expiration de 20 000 ms et un délai de distribution de 5 000 ms, et obtenu après un temps écoulé de 10 000 ms, la valeur de la zone MQMD.expiry peut être d'environ 50 dixièmes de seconde. Cette valeur indique que 15 secondes se sont écoulées entre le moment où le message a été inséré et le moment où il a été obtenu.

Si un message arrive à expiration alors qu'il se trouve dans la file d'attente de transfert et que l'une des options MQRO_EXPIRATION_ * est définie, le rapport généré concerne le message d'origine envoyé par l'application et l'en-tête utilisé pour contenir les informations de délai de distribution est supprimé.

Arrêt et démarrage du processeur de délai de distribution

z/OS Sous z/OS, le processeur de délai de distribution est intégré à l'espace adresse MSTR du gestionnaire de files d'attente. Lorsque le gestionnaire de files d'attente démarre, le processeur de délai de distribution démarre également. Si la file d'attente de transfert est disponible, elle ouvre la file d'attente et attend que les messages qu'elle contient soient traités. Si la file d'attente de transfert n'a pas été définie ou est désactivée pour les extractions, ou si une autre erreur se produit, le processeur de délai de distribution s'arrête. Si la file d'attente de transfert est définie ultérieurement ou modifiée pour être activée, le processeur de délai de distribution redémarre. Si le processeur de délai de distribution s'arrête pour une autre raison, il peut être redémarré en modifiant l'attribut **PUT** de la file d'attente de transfert de ENABLED à DISABLED , puis de nouveau à ENABLED . Si vous devez arrêter le processeur de délai de distribution pour une raison quelconque, définissez l'attribut PUT de la file d'attente de transfert sur DISABLED.

Multi Sous Multiplateformes, le processeur de délai démarre avec le gestionnaire de files d'attente et est automatiquement redémarré en cas d'échec récupérable.

Echec de l'insertion dans la file d'attente cible

Si un message retardé ne peut pas être inséré dans la file d'attente cible une fois son retard terminé, le message est traité comme indiqué dans ses options de rapport: il est soit supprimé, soit envoyé à la file d'attente de rebut. Si cette action échoue, une tentative d'insertion ultérieure du message est effectuée. Si l'action aboutit, un rapport d'exception est généré et envoyé à la file d'attente spécifiée, si le rapport est demandé. Si le message de rapport n'a pas pu être envoyé, il est envoyé à la file d'attente des messages non livrés. Si l'envoi du rapport à la file d'attente de rebut échoue et que le message est persistant, toutes les modifications sont supprimées et le message d'origine est annulé et redistribué ultérieurement. Si le message n'est pas persistant, le message de rapport est supprimé, mais d'autres modifications sont validées. Si une publication différée ne peut pas être distribuée parce qu'un abonné s'est désabonné ou, dans le cas d'un abonné non durable, parce qu'il s'est déconnecté, le message est supprimé en mode silencieux. Les messages de rapport sont toujours générés comme décrit précédemment.

Si une publication différée ne peut pas être distribuée à un abonné et qu'elle est placée dans la file d'attente des messages non livrés et que l'insertion dans la file d'attente des messages non livrés échoue, le message est supprimé.

Pour réduire la probabilité d'échec de l'insertion dans la file d'attente cible une fois le délai de distribution terminé, le gestionnaire de files d'attente effectue des vérifications de base lorsque le client JMS envoie un message avec un délai de distribution différent de zéro. Ces vérifications consistent à déterminer si la file d'attente est désactivée, si le message est supérieur à la longueur maximale autorisée et si la file d'attente est saturée.

Publication/abonnement

La mise en correspondance d'une publication avec les abonnements disponibles se produit lorsque l'application JMS envoie un message avec un délai de distribution différent de zéro. Un message est envoyé à SYSTEM.DDELAY.LOCAL.QUEUE , où elle est conservée jusqu'à la fin du délai de distribution. Si l'un de ces abonnés est un abonnement proxy pour un autre gestionnaire de files d'attente, la distribution sur ce gestionnaire de files d'attente se produit une fois le délai de distribution terminé. Ainsi, les abonnés de l'autre gestionnaire de files d'attente peuvent recevoir des publications qui ont été publiées à l'origine avant leur abonnement. Il s'agit d'un écart par rapport à la spécification JMS 2.0 ou ultérieure.

Le délai de distribution avec publication / abonnement n'est pris en charge que si la rubrique cible est configurée avec (N) PMSGDLV = ALLAVAIL. Une tentative d'utilisation d'autres valeurs entraîne une erreur MQRC_PUBLICATION_FAILURE. Si le processeur de délai de distribution échoue lors de l'insertion du message dans la file d'attente cible, le résultat est décrit dans la section "Echec de l'insertion dans la file d'attente cible".

Messages de rapport

Toutes les options de rapport sont prises en charge et activées par le processeur de distribution, à l'exception des options suivantes qui sont ignorées, mais qui sont transmises au message lorsqu'il est envoyé à la file d'attente cible:

- MQRO_COA*
- COD_MQRO*
- PAN_MQRO/NAN MQRO/MQRO_
- ACTIVITE MQRO

Abonnements clonés et partagés

Il existe deux méthodes pour permettre à plusieurs consommateurs d'accéder au même abonnement. Ces deux méthodes utilisent des abonnements clonés ou des abonnements partagés.

Abonnements clonés

L'abonnement cloné est une extension IBM MQ . Les abonnements clonés permettent à plusieurs consommateurs de différentes machines virtuelles Java (JVM) d'accéder simultanément à l'abonnement. Ce comportement peut être utilisé en définissant la propriété **CLONESUPP** sur `Activé` sur un objet ConnectionFactory . Par défaut, **CLONESUPP** est `Désactivé`. Les abonnements clonés ne peuvent être activés que sur les abonnements durables. Si **CLONESUPP** est activé, chaque connexion ultérieure établie à l'aide de cette ConnectionFactory est clonée.

Un abonnement durable peut être considéré comme cloné si un ou plusieurs destinataires sont créés pour recevoir des messages de cet abonnement, c'est-à-dire qu'ils ont été créés en spécifiant le même nom d'abonnement. Cette opération ne peut être effectuée que si la connexion sous laquelle les consommateurs ont été créés a **CLONESUPP** défini sur `Activé` dans MQConnectionFactory. Lorsqu'un message est publié sur la rubrique de l'abonnement, une copie de ce message est envoyée à l'abonnement. Le message est disponible pour tous les consommateurs, mais un seul le reçoit.

Remarque : L'activation des abonnements clonés étend la spécification JMS .

Abonnements partagés

Les abonnements partagés, qui ont été introduits par la spécification JMS 2.0 , permettent aux messages d'un abonnement de rubrique d'être partagés entre plusieurs destinataires. Chaque message de l'abonnement est distribué à un seul consommateur de cet abonnement. Les abonnements partagés sont activés par l'appel approprié à l'API JMS 2.0 ou ultérieure.

Les API peuvent être appelées de l'une des manières suivantes:

- A partir d'une application Java SE (ou d'un conteneur client Java EE).
- A partir d'un servlet ou de l'implémentation d'un bean géré par message.

La spécification JMS 2.0 ou ultérieure ne définit pas de méthode standard pour la gestion d'un bean géré par message à partir d'un abonnement partagé. Par conséquent, IBM MQ fournit la propriété de spécification d'activation **sharedSubscription** à cette fin. Pour plus d'informations sur cette propriété, voir «Configuration de l'adaptateur de ressources pour les communications entrantes», à la page 463 et «Exemples de définition de la propriété sharedSubscription», à la page 481.

Si un abonnement partagé est activé, il ne peut pas être dissocié.

Les abonnements partagés peuvent être créés en tant qu'abonnements durables ou non durables. Il n'est pas nécessaire de créer séparément des objets côté gestionnaire de files d'attente au-delà de la configuration JMS normale. Tous les objets requis sont créés de manière dynamique.

Choix entre des abonnements partagés ou clonés

Lorsque vous décidez d'utiliser des abonnements partagés ou clonés, tenez compte des avantages des deux. Dans la mesure du possible, utilisez des abonnements partagés car il s'agit d'un comportement défini par la spécification et non d'une extension spécifique à IBM MQ .

Le tableau suivant contient certains des points à prendre en compte lorsque vous décidez entre des abonnements partagés et clonés:

Tableau 48. Comparaison des considérations relatives aux abonnements partagés et aux abonnements clonés

Abonnements partagés	**Abonnements clonés**
Les abonnements partagés sont une partie standard de la spécification JMS 2.0 ou ultérieure.	Les abonnements clonés sont une extension spécifique à IBM MQ .
Les abonnements partagés sont créés à l'aide d'appels de méthode d'API explicites.	Les abonnements clonés sont contrôlés administrativement au niveau de la ConnectionFactory .
Les abonnements partagés peuvent être durables ou non durables.	Les abonnements clonés peuvent uniquement être durables.
Les abonnements partagés sont explicitement créés sur la base d'un abonnement individuel.	Les abonnements clonés sont utilisés pour tout abonnement durable sous une connexion pour laquelle la fonction est activée.
Si un abonnement est créé en tant qu'abonnement partagé, il ne peut pas être changé ultérieurement en abonnement non partagé, ou inversement.	Un abonnement peut être modifié de cloné à non cloné chaque fois qu'il est rouvert si la propriété **CLONESUPP** de la connexion propriétaire a été modifiée.

Concepts associés

Abonnés et abonnements

Durabilité des abonnements

Tâches associées

Utilisation d'abonnements partagés JMS 2.0

Référence associée

«Exemples de définition de la propriété sharedSubscription», à la page 481

Vous pouvez définir la propriété sharedSubscription d'une spécification d'activation dans un fichier WebSphere Liberty `server.xml` . Vous pouvez également définir la propriété dans un bean géré par message (MDB) à l'aide d'annotations.

CLONESUPP

Propriété SupportMQExtensions

La spécification JMS 2.0 a introduit des modifications dans le mode de fonctionnement de certains comportements. IBM MQ 8.0 et les versions ultérieures incluent la propriété **`com.ibm.mq.jms.SupportMQExtensions`**, qui peut être définie sur TRUE pour rétablir ces comportements modifiés dans les implémentations précédentes.

JM 3.0 La propriété **`com.ibm.mq.jakarta.jms.SupportMQExtensions`** (Jakarta Messaging 3.0) est prise en charge par le IBM MQ classes for Jakarta Messaging, qui sont disponibles dans `com.ibm.mq.jakarta.client.jar`.

JMS 2.0 La propriété **`com.ibm.mq.jms.SupportMQExtensions`** (JMS 2.0) est prise en charge par IBM MQ classes for JMS, qui sont disponibles dans `com.ibm.mq.allclient.jar` ou `com.ibm.mqjms.jar`.

Trois zones de fonctionnalité sont rétablies en définissant **`SupportMQExtensions`** sur `True`:

Priorité de message
Les messages peuvent avoir une priorité comprise entre 0 et 9. Avant JMS 2.0, les messages pouvaient également utiliser la valeur -1, indiquant que la priorité par défaut d'une file d'attente était utilisée. JMS 2.0 et les versions ultérieures n'autorisent pas la définition d'une priorité de message de -1. L'activation de **SupportMQExtensions** permet d'utiliser la valeur -1.

ID de client
La spécification JMS 2.0 ou ultérieure requiert que les ID client non nuls soient vérifiés pour leur unicité lors de l'établissement d'une connexion. L'activation de **SupportMQExtensions** signifie que cette exigence n'est pas prise en compte et qu'un ID client peut être réutilisé.

NoLocal
La spécification JMS 2.0 ou ultérieure requiert que, lorsque cette constante est activée, un consommateur ne puisse pas recevoir de messages publiés par le même ID client. Avant JMS 2.0, cet attribut était défini sur un abonné pour l'empêcher de recevoir des messages publiés par sa propre connexion. L'activation de **SupportMQExtensions** rétablit ce comportement dans son implémentation précédente.

Cette propriété peut être définie comme suit:

```
java -Dcom.ibm.mq.jms.SupportMQExtensions=true
```

Cette propriété peut être définie en tant que propriété système JVM standard dans la commande **java** ou contenue dans le fichier de configuration IBM MQ classes for JMS .

Concepts associés
«Le fichier de configuration IBM MQ classes for JMS/Jakarta Messaging», à la page 102
Les fichiers de configuration IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging spécifient les propriétés utilisées pour configurer IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging.

Référence associée
«Propriétés utilisées pour configurer le comportement du client JMS», à la page 109
Utilisez ces propriétés pour configurer le comportement du client JMS .

Utilisation d'abonnements partagés dans des applications JMS
Avec les abonnements partagés, un seul abonnement est partagé entre plusieurs consommateurs, un seul des consommateurs recevant une publication à un moment donné.

Les abonnements partagés sont disponibles à partir de JMS 2.0 . Lorsque vous développez une application JMS pour IBM MQ 8.0 ou version ultérieure, vous devrez peut-être prendre en compte l'impact de cette fonctionnalité sur votre gestionnaire de files d'attente.

L'idée derrière les abonnements partagés est de partager la charge entre plusieurs consommateurs. Un abonnement durable peut également être partagé entre plusieurs consommateurs.

Par exemple, supposons qu'il existe:

- Abonnement SUB, abonnement à un sujet FIFA2014/UPDATES pour recevoir les mises à jour des matchs de football, partagé par trois consommateurs C1, C2et C3
- Publication P1 du producteur dans la rubrique FIFA2014/UPDATES

Lorsqu'une publication est effectuée sur FIFA2014/UPDATES, elle est reçue par un seul des trois destinataires (C1, C2ou C3), mais pas par tous.

L'exemple suivant illustre l'utilisation des abonnements partagés, ainsi que l'utilisation de l'API supplémentaire dans JMS 2.0, `Message.receiveBody()`, pour extraire uniquement le corps du message.

L'exemple crée trois unités d'exécution d'abonné, qui créent un abonnement partagé à la rubrique FIFA2014/UPDATES , et une unité d'exécution de diffuseur de publications.

JM 3.0

```
package mqv91Samples;
```

```
import jakarta.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import jakarta.jms.JMSContext;
import jakarta.jms.Topic;
import jakarta.jms.Queue;
import jakarta.jms.JMSConsumer;
import jakarta.jms.Message;
import jakarta.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

JMS 2.0

```
package mqv91Samples;

import javax.jms.JMSException;

import com.ibm.msg.client.jms.JmsConnectionFactory;
import com.ibm.msg.client.jms.JmsFactoryFactory;
import com.ibm.msg.client.wmq.WMQConstants;

import javax.jms.JMSContext;
import javax.jms.Topic;
import javax.jms.Queue;
import javax.jms.JMSConsumer;
import javax.jms.Message;
```

```
import javax.jms.JMSProducer;

/*
* Implements both Subscriber and Publisher
*/
class SharedNonDurableSubscriberAndPublisher implements Runnable {
   private Thread t;
   private String threadName;

   SharedNonDurableSubscriberAndPublisher( String name){
       threadName = name;
       System.out.println("Creating Thread:" +  threadName );
   }

   /*
    * Demonstrates shared non-durable subscription in JMS 2.0 and later
    */
    private void sharedNonDurableSubscriptionDemo(){
        JmsConnectionFactory cf = null;
        JMSContext msgContext = null;

        try {
            // Create Factory for WMQ JMS provider
            JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
            // Create connection factory
            cf = ff.createConnectionFactory();
            // Set MQ properties
            cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
            cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);
            // Create message context
            msgContext = cf.createContext();

            // Create a topic destination
            Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

            // Create a consumer. Subscription name specified, required for sharing of subscription.
            JMSConsumer msgCons = msgContext.createSharedConsumer(fifaScores, "FIFA2014SUBID");

            // Loop around to receive publications
            while(true){

                String msgBody=null;

                // Use JMS 2.0 and later receiveBody method as we are interested in message body only.
                msgBody = msgCons.receiveBody(String.class);

                if(msgBody != null){
                    System.out.println(threadName + " : " + msgBody);
                }
            }
        }catch(JMSException jmsEx){
            System.out.println(jmsEx);
        }
    }
```

```
 /*
        * Publisher publishes match updates like current attendance in the stadium, goal score and ball
possession by teams.
        */
        private void matchUpdatePublisher(){
            JmsConnectionFactory cf = null;
            JMSContext msgContext = null;
            int nederlandsGoals = 0;
            int chileGoals = 0;
            int stadiumAttendence = 23231;
            int switchIndex = 0;
            String msgBody = "";
            int nederlandsHolding = 60;
            int chileHolding = 40;

            try {
                // Create Factory for WMQ JMS provider
                JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);

                // Create connection factory
                cf = ff.createConnectionFactory();
                // Set MQ properties
                cf.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "QM3");
                cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_BINDINGS);

                // Create message context
```

```
                msgContext = cf.createContext();

                // Create a topic destination
                Topic fifaScores = msgContext.createTopic("/FIFA2014/UPDATES");

                // Create publisher to publish updates from stadium
                JMSProducer msgProducer = msgContext.createProducer();

                while(true){
                    // Send match updates
                    switch(switchIndex){
                        // Attendance
                    case 0:
                        msgBody ="Stadium Attendence " + stadiumAttendence;
                        stadiumAttendence += 314;
                        break;

                        // Goals
                    case 1:
                        msgBody ="SCORE: The Netherlands: " + nederlandsGoals + " - Chile:" + chileGoals;
                        break;

                        // Ball possession percentage
                    case 2:
                        msgBody ="Ball possession: The Netherlands: " + nederlandsHolding + "% - Chile:
" + chileHolding + "%";
                        if((nederlandsHolding > 60) && (nederlandsHolding < 70)){
                            nederlandsHolding -= 2;
                            chileHolding += 2;
                        }else{
                            nederlandsHolding += 2;
                            chileHolding -= 2;
                        }
                        break;
                    }

                    // Publish and wait for two seconds to publish next update
                    msgProducer.send (fifaScores, msgBody);
                    try{
                        Thread.sleep(2000);
                    }catch(InterruptedException iex){

                    }

                    // Increment and reset the index if greater than 2
                    switchIndex++;
                    if(switchIndex > 2)
                        switchIndex = 0;
                }
            }catch(JMSException jmsEx){
                System.out.println(jmsEx);
            }
        }

    /*
     * (non-Javadoc)
     * @see java.lang.Runnable#run()
     */
    public void run() {
        // If this is a publisher thread
        if(threadName == "PUBLISHER"){
            matchUpdatePublisher();
        }else{
            // Create subscription and start receiving publications
            sharedNonDurableSubscriptionDemo();
        }
    }

       // Start thread
   public void start (){
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

```
/*
```

```
* Demonstrate JMS 2.0 and later simplified API using IBM MQ 91 JMS Implementation
*/
public class Mqv91jms2Sample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        // Create first subscriber and start
        SharedNonDurableSubscriberAndPublisher subOne = new
SharedNonDurableSubscriberAndPublisher( "SUB1");
        subOne.start();

        // Create second subscriber and start
        SharedNonDurableSubscriberAndPublisher subTwo = new
SharedNonDurableSubscriberAndPublisher( "SUB2");
        subTwo.start();

        // Create third subscriber and start
        SharedNonDurableSubscriberAndPublisher subThree = new
SharedNonDurableSubscriberAndPublisher( "SUB3");
        subThree.start();

        // Create publisher and start
        SharedNonDurableSubscriberAndPublisher publisher = new
SharedNonDurableSubscriberAndPublisher( "PUBLISHER");
        publisher.start();
    }
}
```

Concepts associés

Interfaces de langage IBM MQ Java

V 9.4.0 Configuration de votre application modulaire pour utiliser IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging

V 9.4.0 Vous pouvez utiliser IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging de manière modulaire en nécessitant le module approprié dans votre application et en incluant le répertoire approprié dans le chemin du module.

L'emballage modulaire

Les fichiers JAR unifiés pour IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging fournissent des noms de module automatiques, qui remplacent les noms par défaut dérivés des noms de fichier JAR.

- IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) est fourni avec le nom de module `com.ibm.mq.javax`.
- IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) est fourni avec le nom de module `com.ibm.mq.jakarta`.

Le répertoire `MQ_HOME/java/lib` par défaut ne convient pas à une utilisation modulaire car les modules ne peuvent pas contenir le même package et le répertoire par défaut contient les mêmes packages dans plusieurs fichiers JAR. Par conséquent, de nouveaux répertoires contenant uniquement les fichiers JAR nécessaires sont disponibles, sans duplication des packages entre les fichiers JAR. Ces répertoires peuvent être inclus sur un `module-path`.

Remarque : Si vous disposez d'applications qui utilisent les fichiers JAR disponibles dans un contexte modulaire en s'appuyant sur les noms de module par défaut, vous devez mettre à jour vos applications pour exiger les nouveaux noms de module. Les noms de module par défaut sont dérivés des noms de fichier JAR.

Configuration de votre application modulaire pour utiliser IBM MQ classes for JMS

Vous pouvez configurer votre application modulaire pour utiliser IBM MQ classes for JMS (`com.ibm.mq.allclient.jar`) en procédant comme suit:

- Configurez l'application pour exiger le module `com.ibm.mq.javax` .

- Configurez l'application pour inclure le répertoire `MQ_HOME/java/lib/modules/javax` dans le chemin de module.

Configuration de votre application modulaire pour utiliser IBM MQ classes for Jakarta Messaging

Vous pouvez configurer votre application modulaire pour utiliser IBM MQ classes for Jakarta Messaging (`com.ibm.mq.jakarta.client.jar`) en procédant comme suit:

- Configurez l'application pour exiger le module `com.ibm.mq.jakarta` .
- Configurez l'application pour inclure le répertoire `MQ_HOME/java/lib/modules/jakarta` dans le chemin de module.

Configuration de votre application modulaire pour utiliser IBM MQ classes for Java

Pour utiliser IBM MQ classes for Java à partir d'une application modulaire, vous pouvez utiliser la configuration pour IBM MQ classes for JMS ou la configuration pour IBM MQ classes for Jakarta Messaging, car les deux fichiers JAR du client prennent en charge IBM MQ classes for Java. Toutefois, votre application ne doit utiliser qu'une seule de ces configurations, et non les deux.

Fonctions de IBM MQ classes for JMS Application Server

Cette rubrique décrit comment IBM MQ classes for JMS implémente la classe ConnectionConsumer et la fonctionnalité avancée dans la classe Session. Il récapitule également la fonction d'un pool de sessions de serveur.

Important : Ces informations sont fournies pour référence uniquement. Une application ne doit pas être écrite pour utiliser cette interface: elle est utilisée dans l'adaptateur de ressources IBM MQ pour se connecter aux serveurs Java EE. Pour des informations de connexion pratiques, voir «Utilisation de l'adaptateur de ressources IBM MQ», à la page 445.

IBM MQ classes for JMS prend en charge les fonctions ASF (Application Server Facilities) qui sont spécifiées dans la *spécification Java Message Service* (voir Oracle Technology Network for Java Developers). Cette spécification identifie trois rôles dans ce modèle de programmation:

- **Le JMS fournisseur** fournit la fonctionnalité ConnectionConsumer et la fonctionnalité de session avancée.
- **Le serveur d'applications** fournit la fonctionnalité ServerSessionPool et ServerSession .
- **L'application client** utilise la fonctionnalité fournie par le fournisseur JMS et le serveur d'applications.

Les informations de cette rubrique ne s'appliquent pas si une application utilise une connexion en temps réel à un courtier.

JMS ConnectionConsumer

L'interface ConnectionConsumer fournit une méthode hautes performances permettant de distribuer des messages simultanément à un pool d'unités d'exécution.

La spécification JMS permet à un serveur d'applications de s'intégrer étroitement à une implémentation JMS à l'aide de l'interface `ConnectionConsumer` . Cette fonction permet le traitement simultané des messages. Généralement, un serveur d'applications crée un pool d'unités d'exécution et l'implémentation JMS met les messages à la disposition de ces unités d'exécution. Un serveur d'applications JMS(tel que WebSphere Application Server) peut utiliser cette fonction pour fournir des fonctionnalités de messagerie de haut niveau, telles que des beans gérés par message.

Les applications normales n'utilisent pas ConnectionConsumer, mais les clients JMS experts peuvent l'utiliser. Pour de tels clients, ConnectionConsumer fournit une méthode hautes performances permettant de distribuer des messages simultanément à un pool d'unités d'exécution. Lorsqu'un message arrive dans une file d'attente ou une rubrique, JMS sélectionne une unité d'exécution dans le pool et lui distribue un lot de messages. Pour ce faire, JMS exécute une méthode `onMessage()` de MessageListenerassociée.

Vous pouvez obtenir le même effet en construisant plusieurs objets Session et MessageConsumer , chacun avec un MessageListenerenregistré. Cependant, ConnectionConsumer offre de meilleures performances, moins d'utilisation des ressources et une plus grande flexibilité. En particulier, moins d'objets Session sont requis.

Planification d'une application avec ASF

Cette section explique comment planifier une application, notamment:

- «Principes généraux de la messagerie point-à-point à l'aide d'ASF», à la page 343
- «Principes généraux de la messagerie de publication / abonnement à l'aide d'ASF», à la page 344
- «Suppression de messages de la file d'attente dans ASF», à la page 345
- Traitement des messages incohérents dans ASF. Voir «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.

Principes généraux de la messagerie point-à-point à l'aide d'ASF
Utilisez cette rubrique pour obtenir des informations générales sur la messagerie point-à-point à l'aide d'ASF.

Lorsqu'une application crée un objet ConnectionConsumer à partir d'un objet QueueConnection , elle spécifie un objet de file d'attente JMS et une chaîne de sélecteur. ConnectionConsumer commence alors à fournir des messages aux sessions du pool ServerSessionassocié. Les messages arrivent dans la file d'attente et, s'ils correspondent au sélecteur, ils sont distribués aux sessions du pool ServerSessionassocié.

En termes IBM MQ , l'objet file d'attente fait référence à un QLOCAL ou à un QALIAS sur le gestionnaire de files d'attente local. S'il s'agit d'un QALIAS, ce QALIAS doit faire référence à un QLOCAL. La valeur IBM MQ QLOCAL entièrement résolue est appelée *QLOCAL sous-jacente*. Un ConnectionConsumer est dit *actif* s'il n'est pas fermé et que son parent QueueConnection est démarré.

Il est possible que plusieurs ConnectionConsumers, chacun avec des sélecteurs différents, s'exécutent sur le même système QLOCAL sous-jacent. Pour conserver les performances, les messages indésirables ne doivent pas s'accumuler dans la file d'attente. Les messages non souhaités sont ceux pour lesquels aucun ConnectionConsumer actif ne possède de sélecteur correspondant. Vous pouvez définir la fabrique QueueConnectionde sorte que ces messages indésirables soient supprimés de la file d'attente (pour plus de détails, voir «Suppression de messages de la file d'attente dans ASF», à la page 345). Vous pouvez définir ce comportement de l'une des deux manières suivantes:

- Utilisez l'outil d'administration JMS pour définir la fabrique QueueConnectionsur MRET (NO).
- Dans votre programme, utilisez:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Si vous ne modifiez pas ce paramètre, la valeur par défaut consiste à conserver ces messages indésirables dans la file d'attente.

Lorsque vous configurez le gestionnaire de files d'attente IBM MQ , tenez compte des points suivants:

- Le système QLOCAL sous-jacent doit être activé pour l'entrée partagée. Pour ce faire, utilisez la commande MQSC suivante:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Votre gestionnaire de files d'attente doit disposer d'une file d'attente de rebut activée. Si un ConnectionConsumer rencontre un problème lorsqu'il place un message dans la file d'attente des messages non livrés, la distribution des messages à partir de la file d'attente QLOCAL sous-jacente s'arrête. Pour définir une file d'attente de rebut, utilisez:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- L'utilisateur qui exécute ConnectionConsumer doit avoir le droit d'exécuter MQOPEN avec MQOO_SAVE_ALL_CONTEXT et MQOO_PASS_ALL_CONTEXT. Pour plus de détails, voir la documentation IBM MQ de votre plateforme spécifique.
- Si des messages indésirables sont laissés dans la file d'attente, ils dégradent les performances du système. Par conséquent, planifiez vos sélecteurs de messages de sorte qu'entre eux, les ConnectionConsumers suppriment tous les messages de la file d'attente.

Pour plus de détails sur les commandes MQSC, voir Commandes MQSC.

Principes généraux de la messagerie de publication / abonnement à l'aide d'ASF
Les ConnectionConsumers reçoivent des messages pour une rubrique spécifiée. Un ConnectionConsumer peut être durable ou non durable. Vous devez spécifier la ou les files d'attente utilisées par ConnectionConsumer .

Lorsqu'une application crée un objet ConnectionConsumer à partir d'un objet TopicConnection , elle spécifie un objet Topic et une chaîne de sélecteur. ConnectionConsumer commence alors à recevoir des messages qui correspondent au sélecteur de cette rubrique, y compris les publications conservées pour la rubrique à laquelle il est abonné.

Une application peut également créer un ConnectionConsumer durable associé à un nom spécifique. Ce ConnectionConsumer reçoit les messages qui ont été publiés sur la rubrique depuis la dernière fois que le ConnectionConsumer durable a été actif. Il reçoit tous les messages de ce type qui correspondent au sélecteur sur la rubrique. Toutefois, si ConnectionConsumer utilise la fonction de lecture anticipée, il peut perdre les messages non persistants qui se trouvent dans la mémoire tampon du client lors de sa fermeture.

Si IBM MQ classes for JMS est en mode de migration du fournisseur de messagerie IBM MQ , une file d'attente distincte est utilisée pour les abonnements ConnectionConsumer non durables.L'option configurable CCSUB de la fabrique TopicConnectionspécifie la file d'attente à utiliser. Normalement, CCSUB spécifie une file d'attente unique à utiliser par tous les ConnectionConsumers qui utilisent la même fabrique TopicConnection. Toutefois, il est possible de faire en sorte que chaque ConnectionConsumer génère une file d'attente temporaire en spécifiant un préfixe de nom de file d'attente suivi d'un astérisque (*).

Si IBM MQ classes for JMS est en mode de migration du fournisseur de messagerie IBM MQ , la propriété CCDSUB de la rubrique indique la file d'attente à utiliser pour les abonnements durables. Encore une fois, il peut s'agir d'une file d'attente qui existe déjà ou d'un préfixe de nom de file d'attente suivi d'un astérisque (*). Si vous spécifiez une file d'attente qui existe déjà, tous les ConnectionConsumers durables qui s'abonnent à la rubrique utilisent cette file d'attente. Si vous spécifiez un préfixe de nom de file d'attente suivi d'un astérisque (*), une file d'attente est générée la première fois qu'un ConnectionConsumer durable est créé avec un nom particulier. Cette file d'attente est réutilisée ultérieurement lorsqu'un ConnectionConsumer durable est créé avec le même nom.

Lorsque vous configurez le gestionnaire de files d'attente IBM MQ , tenez compte des points suivants:

- Votre gestionnaire de files d'attente doit disposer d'une file d'attente de rebut activée. Si un ConnectionConsumer rencontre un problème lorsqu'il place un message dans la file d'attente des messages non livrés, la distribution des messages à partir de la file d'attente QLOCAL sous-jacente s'arrête. Pour définir une file d'attente de rebut, utilisez:

  ```
  ALTER QMGR DEADQ( your.dead.letter.queue.name )
  ```

- L'utilisateur qui exécute ConnectionConsumer doit avoir le droit d'exécuter MQOPEN avec MQOO_SAVE_ALL_CONTEXT et MQOO_PASS_ALL_CONTEXT. Pour plus de détails, voir la documentation IBM MQ de votre plateforme.
- Vous pouvez optimiser les performances d'un ConnectionConsumer individuel en créant une file d'attente distincte et dédiée pour ce dernier. Cela se fait au prix d'une utilisation supplémentaire des ressources.

Suppression de messages de la file d'attente dans ASF
Lorsqu'une application utilise ConnectionConsumers, JMS peut avoir besoin de supprimer des messages de la file d'attente dans un certain nombre de situations.

Ces situations sont les suivantes:

Message mal formaté
Il se peut qu'un message JMS ne puisse pas être analysé.

Message incohérent
Un message peut atteindre le seuil d'annulation, mais ConnectionConsumer ne parvient pas à le replacer dans la file d'attente d'annulation.

Aucun ConnectionConsumer intéressé
Pour la messagerie point-à-point, lorsque la fabrique QueueConnectionest définie de sorte qu'elle ne conserve pas les messages non souhaités, un message arrive qui n'est pas souhaité par l'un des ConnectionConsumers.

Dans ces situations, ConnectionConsumer tente de supprimer le message de la file d'attente. Les options de disposition dans la zone de rapport du MQMD du message définissent le comportement exact. Ces options sont les suivantes :

MQRO_DEAD_LETTER_Q
Le message est remis en file d'attente dans la file d'attente de rebut du gestionnaire de files d'attente. Il s'agit de l'option par défaut.

MQRO_DISCARD_MSG
Le message est supprimé.

ConnectionConsumer génère également un message de rapport, qui dépend également de la zone de rapport du MQMD du message. Ce message est envoyé à la file d'attente ReplyTodu message sur le gestionnaire de files d'attente ReplyTo. En cas d'erreur lors de l'envoi du message de rapport, le message est envoyé à la file d'attente des messages non livrés. Les options de rapport d'exception dans la zone de rapport du MQMD du message définissent les détails du message de rapport. Ces options sont les suivantes :

MQRO_EXCEPTION
Un message de rapport contenant le MQMD du message d'origine est généré. Il ne contient pas de données de corps de message.

MQRO_EXCEPTION_WITH_DATA
Un message de rapport est généré qui contient le MQMD, tous les en-têtes MQ et 100 octets de données de corps.

MQRO_EXCEPTION_WITH_FULL_DATA
Un message de rapport contenant toutes les données du message d'origine est généré.

par défaut
Aucun message de rapport n'est généré.

Lorsque des messages de rapport sont générés, les options suivantes sont utilisées:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Si un message incohérent ne peut pas être remis en file d'attente, peut-être parce que la file d'attente des messages non livrés est pleine ou que l'autorisation est mal spécifiée, ce qui se passe dépend de la persistance du message. Si le message est non persistant, il est supprimé et aucun message de rapport n'est généré. Si le message est persistant, la distribution des messages à tous les consommateurs de connexion à l'écoute sur cette destination s'arrête. Ces consommateurs de connexion doivent être fermés et le problème doit être résolu avant qu'ils puissent être recréés et que la distribution des messages puisse être redémarrée.

Il est important de définir une file d'attente de rebut et de la vérifier régulièrement pour s'assurer qu'aucun problème ne se produit. En particulier, assurez-vous que la file d'attente des messages non livrés n'atteint pas sa longueur maximale et que sa taille maximale de message est suffisante pour tous les messages.

Lorsqu'un message est replacé dans la file d'attente de rebut, il est précédé d'un en-tête de rebut IBM MQ (MQDLH). Voir En-tête MQDLH-Dead-letter pour plus de détails sur le format de MQDLH. Vous pouvez identifier les messages qu'un ConnectionConsumer a placés dans la file d'attente de rebut, ou signaler les messages qu'un ConnectionConsumer a générés, à l'aide des zones suivantes:

- PutApplLe type est `MQAT_JAVA (0x1C)`
- PutApplLe nom est " `MQ JMS ConnectionConsumer` "

Ces zones se trouvent dans le MQDLH des messages de la file d'attente des messages non livrés et dans le MQMD des messages de rapport. La zone de retour d'informations du MQMD et la zone Raison du MQDLH contiennent un code décrivant l'erreur. Pour plus de détails sur ces codes, voir «Codes raison et commentaires en retour dans ASF», à la page 347. Les autres zones sont décrites dans MQDLH-Dead-letter header.

Gestion des messages incohérents dans ASF
Dans les utilitaires du serveur d'applications, la gestion des messages incohérents est traitée de manière légèrement différente de celle effectuée ailleurs dans IBM MQ classes for JMS.

Pour plus d'informations sur la gestion des messages incohérents dans IBM MQ classes for JMS, voir «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.

Lorsque vous utilisez ASF (Application Server Facilities), les messages incohérents sont traités par l'objet ConnectionConsumer plutôt que par l'objet MessageConsumer. ConnectionConsumer replace les messages en file d'attente en fonction des propriétés BackoutThreshold et BackoutRequeueQName de celle-ci.

Lorsqu'une application utilise ConnectionConsumers, les circonstances dans lesquelles un message est annulé dépend de la session fournie par le serveur d'application :

- Dans le cas d'une session non transactionnelle avec AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, un message est annulé uniquement après une erreur système ou un arrêt imprévu de l'application.
- Lorsque la session n'est pas transactionnelle avec CLIENT_ACKNOWLEDGE, les messages sans accusé de réception peuvent être annulés par le serveur d'applications qui appelle Session.recover().

  Généralement, l'implémentation client de MessageListener ou le serveur d'applications appelle Message.acknowledge(). Message.acknowledge() accuse réception de tous les messages distribués sur la session jusqu'à présent.
- Lorsque la session est transactionnelle, les messages sans accusé de réception peuvent être annulés par le serveur d'applications appelant Session.rollback().
- Si le serveur d'applications fournit une XASession, les messages sont validés ou annulés en fonction d'une transaction répartie. Le serveur d'applications est responsable de l'exécution de la transaction.

Concepts associés
«Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240
Un message incohérent est un message qui ne peut pas être traité par une application de réception. Si un message incohérent est distribué à une application et annulé un certain nombre de fois, IBM MQ classes for JMS peut le déplacer dans une file d'attente d'annulation.

Traitement des erreurs

Cette section traite de divers aspects du traitement des erreurs, notamment «Reprise après des conditions d'erreur dans l'ASF», à la page 347 et «Codes raison et commentaires en retour dans ASF», à la page 347.

Reprise après des conditions d'erreur dans l'ASF
Si un ConnectionConsumer rencontre une erreur grave, la distribution des messages à tous les ConnectionConsumers ayant un intérêt dans le même système QLOCAL s'arrête. Dans ce cas, tout ExceptionListener enregistré auprès de la connexion affectée est notifié. Il existe deux façons pour une application de récupérer de ces conditions d'erreur.

En règle générale, une erreur grave de cette nature se produit si ConnectionConsumer ne peut pas remettre un message dans la file d'attente de rebut ou si une erreur se produit lors de la lecture des messages provenant de QLOCAL.

Etant donné que tout ExceptionListener enregistré auprès de la connexion affectée est notifié, vous pouvez l'utiliser pour identifier la cause du problème. Dans certains cas, l'administrateur système doit intervenir pour résoudre le problème.

Utilisez l'une des techniques suivantes pour récupérer de ces conditions d'erreur:

- Appelez `close()` sur tous les ConnectionConsumersaffectés. L'application peut créer de nouveaux ConnectionConsumers uniquement une fois que tous les ConnectionConsumers affectés ont été fermés et que les problèmes système ont été résolus.
- Appelez `stop()` sur toutes les connexions affectées. Une fois que toutes les connexions ont été arrêtées et que tous les problèmes système ont été résolus, l'application peut `start()` ses connexions avec succès.

Codes raison et commentaires en retour dans ASF
Utilisez les codes raison et de retour d'informations pour déterminer la cause d'une erreur. Les codes anomalie courants générés par ConnectionConsumer sont indiqués ici.

Pour déterminer la cause d'une erreur, utilisez les informations suivantes:

- Code retour dans tous les messages de rapport
- Code anomalie dans le MQDLH de tous les messages de la file d'attente de rebut

ConnectionConsumers génère les codes anomalie suivants.

MQRC_BACKOUT_THRESHOLD_ATTEINTES (0x93A; 2362)

Cause
Le message a atteint le seuil d'annulation défini sur QLOCAL, mais aucune file d'attente d'annulation n'est définie.

Sur les plateformes où vous ne pouvez pas définir la file d'attente d'annulation, le message a atteint le seuil d'annulation défini par JMSde 20.

Action
Si cela n'est pas souhaité, définissez la file d'attente d'annulation pour la file d'attente QLOCAL concernée. Recherchez également la cause des annulations multiples.

MQRC_MSG_NOT_APPARIÉ (0x93B; 2363)

Cause
Dans la messagerie point-à-point, il existe un message qui ne correspond à aucun des sélecteurs pour les ConnectionConsumers qui surveillent la file d'attente. Pour préserver les performances, le message est replacé dans la file d'attente des messages non livrés.

Action
Pour éviter cette situation, assurez-vous que les ConnectionConsumers qui utilisent la file d'attente fournissent un ensemble de sélecteurs qui traitent tous les messages, ou définissez la fabrique QueueConnectionpour qu'elle conserve les messages.

Vous pouvez également rechercher la source du message.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Cause
JMS ne peut pas interpréter le message dans la file d'attente.

Action

Recherchez l'origine du message. JMS distribue normalement des messages dont le format est inattendu (BytesMessage ou TextMessage). Parfois, cela échoue si le message est très mal formaté.

Les autres codes qui apparaissent dans ces zones sont dus à l'échec d'une tentative de remise en file d'attente du message dans une file d'attente d'annulation. Dans ce cas, le code décrit la raison pour laquelle la remise en file d'attente a échoué. Pour diagnostiquer la cause de ces erreurs, voir API> API completion and reason codes.

Si le message de rapport ne peut pas être inséré dans la file d'attente ReplyTo, il est inséré dans la file d'attente de rebut. Dans ce cas, la zone de retour d'informations du MQMD est renseignée comme décrit dans cette rubrique. La zone anomalie de MQDLH explique pourquoi le message de rapport n'a pas pu être placé sur la file d'attente ReplyTo.

Fonction d'un pool de sessions de serveur dans AFS

Cette rubrique récapitule la fonction d'un pool de sessions serveur.

Le Figure 45, à la page 349 récapitule les principes de la fonctionnalité ServerSessionPool et ServerSession .

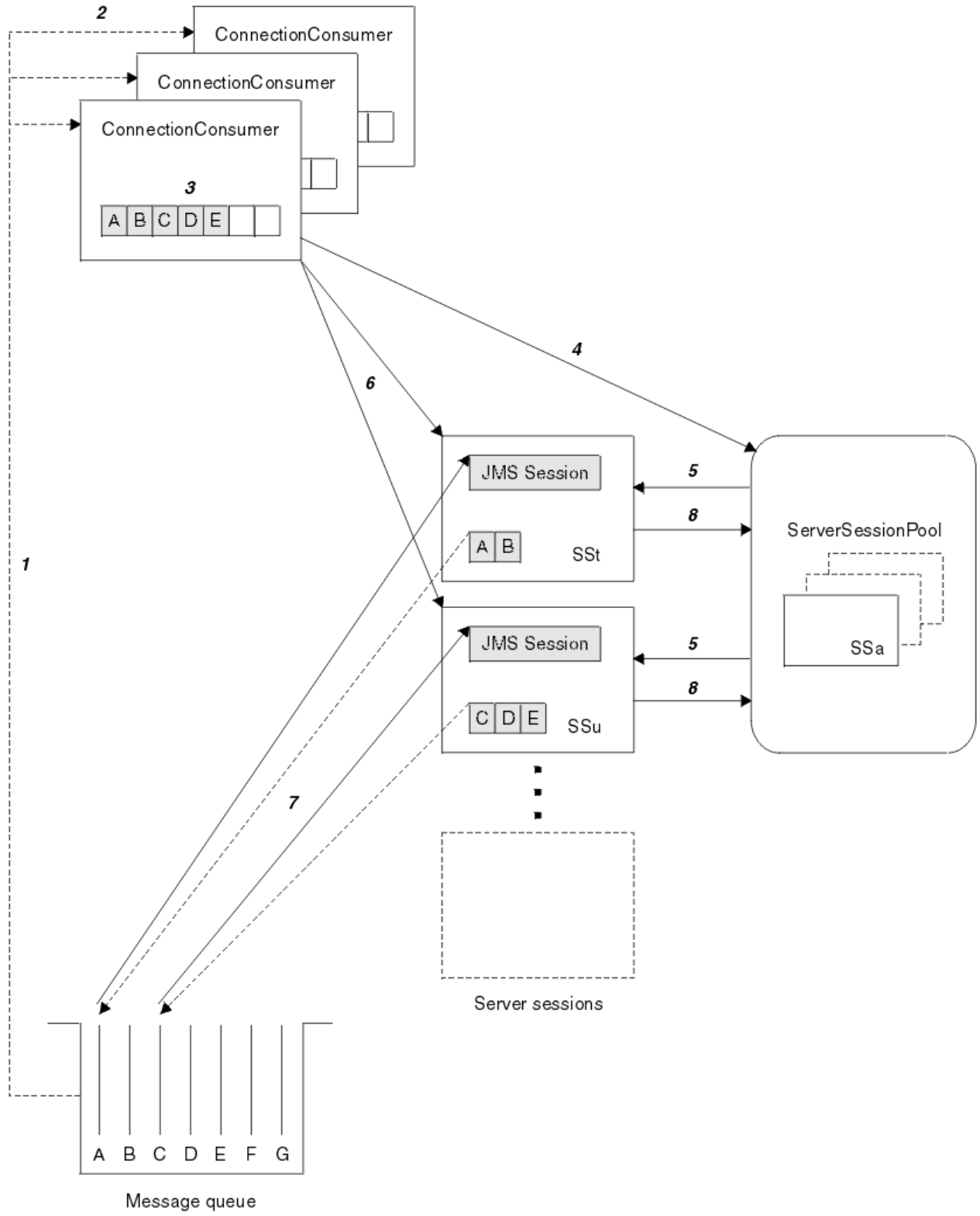


Figure 45. Fonctionnalité ServerSessionPool et ServerSession

1. Les ConnectionConsumers extrayez les références de message de la file d'attente.
2. Chaque ConnectionConsumer sélectionne des références de message spécifiques.
3. La mémoire tampon ConnectionConsumer contient les références de message sélectionnées.
4. ConnectionConsumer demande une ou plusieurs ServerSessions dans le pool ServerSession.

5. Les ServerSessions sont allouées à partir du pool ServerSession.
6. ConnectionConsumer affecte des références de message aux ServerSessions et démarre les unités d'exécution ServerSession en cours d'exécution.
7. Chaque ServerSession extrait ses messages référencés de la file d'attente. Il les transmet à la méthode onMessage à partir de MessageListener qui est associé à la session JMS .
8. Une fois le traitement terminé, la ServerSession est renvoyée au pool.

Un serveur d'applications fournit normalement la fonctionnalité ServerSessionPool et ServerSession .

z/OS Using IBM MQ classes for JMS in a CICS Liberty JVM server

Java programs running in a CICS Liberty JVM server can use the IBM MQ classes for JMS to access IBM MQ.

You must be using an IBM MQ 9.1.0 or later version of the IBM MQ resource adapter. You can obtain the resource adapter from Fix Central (see “Installation de l'adaptateur de ressources dans Liberty” on page 455).

There are two flavors of Liberty Profile JVMs available in CICS 5.3 and later, the types of connection possible to IBM MQ are restricted as follows:

CICS Liberty Standard

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode
- The IBM MQ resource adapter can connect to any in-service version of IBM MQ for z/OS in BINDINGS mode when there is no CICS connection (active CICS MQCONN resource definition) to the same queue manager from the same CICS region.

CICS Liberty Integrated

- The IBM MQ resource adapter can connect to any in-service version of IBM MQ in CLIENT mode.
- BINDINGS mode connection is not supported.

For details on setting up and configuring your system, see Using IBM MQ classes for JMS in a Liberty JVM server in the CICS documentation.

Utilisation de IBM MQ classes for JMS/ Jakarta Messaging dans IMS

La prise en charge de la messagerie basée sur des normes dans un environnement IMS est assurée par l'utilisation de IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging.

Vérifiez la configuration système requise pour le système IMS utilisé par votre entreprise. Pour plus d'informations, voir IMS 15.2 .

Cet ensemble de rubriques décrit comment configurer le IBM MQ classes for JMS dans un environnement IMS , ainsi que les restrictions d'API qui s'appliquent lors de l'utilisation des interfaces classiques (JMS 1.1) et simplifiées (JMS 2.0). Pour obtenir la liste des informations spécifiques à l'API, voir «Restrictions de l'API JMS», à la page 355 .

Remarque : Des restrictions similaires s'appliquent aux interfaces existantes spécifiques au domaine (JMS 1.0.2), mais elles ne sont pas spécifiquement décrites ici.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Régions dépendantes IMS prises en charge

Les types de région dépendante suivants sont pris en charge:

- MPR
- persistance gérée par le bean
- système FastPath IMS
- Machines virtuelles Java (JVM) JMP 31 et 64 bits
- Machines virtuelles Java JBP 31 et 64 bits

Sauf mention spécifique dans les rubriques suivantes, IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging se comportent de la même manière dans tous les types de région.

Machines virtuelles Java prises en charge

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging requièrent IBM Runtime Environment, Java Technology Edition 8. IBM Semeru Runtime Certified Edition for z/OS, Version 11 n'est pas pris en charge.

Autres restrictions

Les restrictions suivantes s'appliquent lors de l'utilisation de IBM MQ classes for JMS dans un environnement IMS :

- Les connexions en mode client ne sont pas prises en charge.
- Les connexions sont uniquement prises en charge pour les gestionnaires de files d'attente IBM MQ 8.0 à l'aide du IBM MQ fournisseur de messagerie `Normal`, mode.

  L'attribut **PROVIDERVERSION** de la fabrique de connexions doit être soit non spécifié, soit une valeur supérieure ou égale à sept.
- L'utilisation de l'une des fabriques de connexions XA, par exemple `com.ibm.mq.jms.MQXAConnectionFactory`, n'est pas prise en charge.

Tâches associées

Définition de IBM MQ sur IMS

Configuration de l'adaptateur IMS à utiliser avec IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging utilisent le même adaptateur IBM MQ-IMS que celui utilisé par d'autres langages de programmation. Cet adaptateur utilise la fonction ESAF (External Subsystem Attach Facility) IMS .

Avant de commencer

Avant d'effectuer la procédure suivante, vous devez configurer l'adaptateur IMS pour les gestionnaires de files d'attente appropriés et les régions de contrôle et dépendantes IMS , comme décrit dans Configuration de l'adaptateur IMS.

Avertissement : Vous n'avez pas besoin d'effectuer l'étape qui décrit la génération d'un stub dynamique, sauf si vous avez besoin du stub dynamique à d'autres fins.

Après avoir configuré l'adaptateur IMS , procédez comme suit.

Procédure

1. Mettez à jour la variable LIBPATH dans le membre de votre PROCLIB IMS référencé par le paramètre ??? ON dans le JCL de votre région dépendante (par exemple, DFSJVMEV) afin qu'elle inclue les bibliothèques natives IBM MQ classes for JMS .

C'est-à-dire le répertoire zFS qui contient `libmqjims.so`. Par exemple, DFSJVMEV peut ressembler à ce qui suit, où la dernière ligne correspond au répertoire contenant les bibliothèques natives IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging :

```
LIBPATH=>
/java/latest/bin/j9vm:>
/java/latest/bin:>
/ims/latest/dbdc/imsjava/classic/lib:>
/ims/latest/dbdc/imsjava/lib:>
/mqm/latest/java/lib
```

2. Ajoutez IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging au chemin d'accès aux classes de la machine virtuelle Java, utilisé par votre région dépendante IMS , en mettant à jour l'option `java.class.path` .

   Pour ce faire, suivez les instructions du membre DFSJVMMS du fichier IMS PROCLIB.

   Par exemple, vous pouvez utiliser ce qui suit, où la ligne en gras indique la mise à jour:

JM 3.0
```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
-Djava.class.path=/ims/latest/dbdc/imsjava/imsutm.jar:/ims/latest/dbdc/imsjava/imsudb.jar:
/mqm/latest/java/lib/com.ibm.mq.allclient.jar
```

   Remarque : Bien que de nombreux fichiers jar différents soient disponibles dans le répertoire contenant le fichier IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, vous n'avez besoin que de `com.ibm.mq.allclient.jar` (JMS 2.0) ou de `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0).
3. Arrêtez et redémarrez les régions dépendantes IMS qui utiliseront IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging.

Que faire ensuite

Créez et configurez des fabriques de connexions et des destinations.

Il existe trois approches possibles pour instancier les implémentations IBM MQ des fabriques de connexions et des destinations. Voir «Création et configuration de fabriques de connexions et de destinations», à la page 211 pour plus de détails.

Notez que ces trois approches sont toutes valides dans un environnement IMS .

Concepts associés

Configuration de l'adaptateur IMS

Définition de IBM MQ sur IMS

Comportement transactionnel

Les messages envoyés et reçus par IBM MQ classes for JMS dans un environnement IMS sont toujours associés à l'unité de travail (UOW) IMS active sur la tâche en cours.

Cette unité de travail ne peut être exécutée qu'en appelant les méthodes de validation ou d'annulation sur une instance de l'objet `com.ibm.ims.dli.tm.Transaction` , ou par la tâche IMS qui se termine normalement, auquel cas l'unité de travail est implicitement validée. Si la tâche IMS se termine de manière anormale, l'unité de travail est annulée.

Ainsi, les valeurs des arguments **`transacted`** et **`acknowledgeMode`** sont ignorés lors de l'appel de la méthode `Connection.createSession` ou `ConnectionFactory.createContext`. De plus, les méthodes ci-après ne sont pas prises en charge. Si vous appelez l'une des méthodes suivantes, une exception `IllegalStateException` est émise dans la session :

- `javax.jms.Session.commit()`

- `javax.jms.Session.recover()`
- `javax.jms.Session.rollback()`

et une exception `IllegalStateRuntimeSession` est émise dans le contexte JMS :

- `javax.jms.JMSContext.commit()`
- `javax.jms.JMSContext.recover()`
- `javax.jms.JMSContext.rollback()`

Il existe une exception à ce comportement. Si une session ou un contexte JMS est créé à l'aide de l'un des mécanismes suivants:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

alors le comportement de cette session, ou contexte JMS , est le suivant:

- Tous les messages envoyés sont envoyés en dehors de l'unité de travail IMS . C'est-à-dire qu'ils seront disponibles sur la destination cible immédiatement ou lorsque l'intervalle de délai de livraison fourni sera écoulé.
- Tous les messages non persistants seront reçus en dehors de l'unité de travail IMS , sauf si la propriété SYNCPOINTALLGETS a été spécifiée sur la fabrique de connexions qui a créé la session ou le contexte JMS .
- Les messages persistants sont toujours reçus dans l'unité de travail IMS .

Cela peut être utile si, par exemple, vous souhaitez écrire un message d'audit dans une file d'attente même si l'unité de travail est annulée.

Implications des points de synchronisation IMS

Le IBM MQ classes for JMS s'appuie sur la prise en charge de l'adaptateur IBM MQ existant qui utilise ESAF. Cela signifie que le comportement documenté s'applique, y compris tous les descripteurs ouverts qui sont fermés par l'adaptateur IMS lorsqu'un point de synchronisation se produit.

z/OS Pour plus d'informations, voir «Syncpoints in IMS applications», à la page 73.

Pour illustrer ce point, prenez en compte le code suivant qui s'exécute dans un environnement JMP. Le deuxième appel à `mp.send()` génère un `JMSException` car le code `messageQueue.getUnique(inputMessage)` entraîne la fermeture de toutes les connexions IBM MQ ouvertes et de tous les descripteurs d'objet.

Un comportement similaire est observé si l'appel `getUnique()` a été remplacé par `Transaction.commit()`, mais pas si `Transaction.rollback()` a été utilisé.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Get a message from an IMS message queue. This results in a GU call
//which results in all MQ handles being closed.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//This attempt to send another message will result in a JMSException containing a
```

```
//MQRC_HCONN_ERROR as the connection/handle has been closed.
mp.send(m);
```

Le code à utiliser dans ce scénario est le suivant. Dans ce cas, la connexion à IBM MQ est fermée avant l'appel de `getUnique()`. La connexion et la session sont ensuite recréées afin d'envoyer un autre message.

```
//Create a connection to queue manager MQ21.
MQConnectionFactory cf = new MQConnectionFactory();
cf.setQueueManager("MQ21");

Connection c = cf.createConnection();
Session s = c.createSession();

//Send a message to MQ queue Q1.
Queue q = new MQQueue("Q1");
MessageProducer mp = s.createProducer(q);
TextMessage m = s.createTextMessage("Hello world!");
mp.send(m);

//Close the connection to MQ, which closes all MQ object handles.
//The send of the message will be committed by the subsequent GU call.
c.close();
c = null;
s = null;
mp = null;

//Get a message from an IMS message queue. This results in a GU call.
Application a = ApplicationFactory.createApplication();
MessageQueue messageQueue = a.getMessageQueue();
IOMessage inputMessage = a.getIOMessage(MESSAGE_CLASS_NAME);
messageQueue.getUnique(inputMessage);

//Re-create the connection to MQ and send another message;
c = cf.createConnection();
s = c.createSession();
mp = s.createProducer(q);
m = s.createTextMessage("Hello world 2!");
mp.send(m);
```

Remarques relatives à l'utilisation de l'adaptateur IMS

Vous devez connaître les restrictions suivantes. Vous ne pouvez avoir qu'un seul descripteur de connexion pour chaque gestionnaire de files d'attente. L'interaction avec IBM MQ a des implications lorsque vous utilisez à la fois le code JMS et le code natif. L'authentification et l'autorisation de connexion sont soumises à des limitations.

Un descripteur de connexion pour chaque gestionnaire de files d'attente

Un seul descripteur de connexion à la fois à un gestionnaire de files d'attente spécifique est autorisé dans les régions IMS dépendantes. Toute tentative ultérieure de connexion au même gestionnaire de files d'attente réutilise le descripteur existant.

Bien que ce comportement ne doive pas entraîner de problèmes dans une application qui utilise uniquement IBM MQ classes for JMS, il peut entraîner des problèmes dans les applications qui interagissent avec IBM MQ, lors de l'utilisation de IBM MQ classes for JMS et de l'interface MQI dans du code natif écrit dans des langages, tels que COBOL ou C.

Implications de l'interaction avec IBM MQ lors de l'utilisation de JMS et de code natif

Des problèmes peuvent se produire lors de l'entrelacement du code Java et du code natif qui utilisent tous deux la fonctionnalité IBM MQ et lorsque la connexion à IBM MQ n'est pas fermée avant de quitter le code natif ou Java.

Par exemple, dans le pseudo-code suivant, un descripteur de connexion à un gestionnaire de files d'attente est initialement établi dans le code Java à l'aide de IBM MQ classes for JMS. Le descripteur de connexion est réutilisé dans le code COBOL et invalidé par un appel à MQDISC.

La prochaine fois que IBM MQ classes for JMS utilisera le descripteur de connexion, un JMSException avec le code anomalie MQRC_HCONN_ERROR sera généré.

```
COBOL code running in message processing region
    Use the Java Native Interface (JNI) to call Java code
       Create MQ connection and session - this creates an MQ connection handle
       Send message to MQ queue
       Store connection and session in static variable
       Return to COBOL code

    MQCONN - picks up MQ connection handle established in Java code
    MQDISC - invalidates connection handle

    Use the Java Native Interface (JNI) to call Java code
       Get session from static variable
       Create a message consumer - fails as connection handle invalidated
```

Il existe d'autres modèles d'utilisation similaires qui peuvent entraîner MQRC_HCONN_ERROR.

Bien qu'il soit possible de partager des descripteurs de connexion IBM MQ entre le code natif et le code Java (par exemple, l'exemple précédent fonctionnerait s'il n'y avait pas eu d'appel MQDISC) en général, la meilleure pratique consiste à fermer tous les descripteurs de connexion avant de passer de Java au code natif, ou l'inverse.

Authentification et autorisation de connexion

La spécification JMS permet de spécifier un nom d'utilisateur et un mot de passe pour l'authentification et l'autorisation lors de la création d'une connexion ou d'un objet de contexte JMS .

Cette opération n'est pas prise en charge dans un environnement IMS . La tentative de création d'une connexion lors de la spécification d'un nom d'utilisateur et d'un mot de passe entraîne l'émission d'un JMS Exception . La tentative de création d'un contexte JMS , lors de la spécification d'un nom d'utilisateur et d'un mot de passe, entraîne l'émission d'un JMSRuntimeException .

A la place, les mécanismes existants d'authentification et d'autorisation lors de la connexion à IBM MQ à partir d'un environnement IMS doivent être utilisés.

Pour plus d'informations, voir Configuration de la sécurité sous z/OS. En particulier, voir la section relative aux ID utilisateur pour la vérification de la sécurité, qui décrit les ID utilisateur que vous pouvez utiliser.

Tâches associées

Configuration de la sécurité sous z/OS

Restrictions de l'API JMS

Du point de vue de la spécification JMS , IBM MQ classes for JMS traite IMS comme un serveur d'applications compatible Java EE ou Jakarta EE , qui a toujours une transaction JTA en cours.

Par exemple, vous ne pouvez jamais appeler `javax.jms.Session.commit()` dans IMS, car la spécification JMS indique que vous ne pouvez pas l'appeler dans un EJB JEE ou un conteneur Web alors qu'une transaction JTA est en cours.

Il en résulte les restrictions suivantes pour l'API JMS , en plus de celles décrites dans «Comportement transactionnel», à la page 352.

Restrictions de l'API classique

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` émet toujours une exception JMSException.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` émet toujours une exception JMSException.
- Les trois variantes de `javax.jms.Connection.createSession` émettent toujours une exception JMSException si la connexion possède déjà une session existante active.

- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` émet toujours une exception `JMSException`.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` émet toujours une exception `JMSException`.
- `javax.jms.Connection.setClientID()` émet toujours une exception `JMSException`.
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` émet toujours une exception `JMSException`.
- `javax.jms.Connection.stop()` émet toujours une exception `JMSException`.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` émet toujours une exception `JMSException`.
- `javax.jms.MessageConsumer.getMessageListener()` émet toujours une exception `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` émet toujours une exception `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` émet toujours une exception `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` émet toujours une exception `JMSException`.
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener )` émet toujours une exception `JMSException`.
- `javax.jms.Session.run()` émet toujours une exception `JMSRuntimeException`.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` émet toujours une exception `JMSException`.
- `javax.jms.Session.getMessageListener()` émet toujours une exception `JMSException`.

Restrictions d'API simplifiées

- `javax.jms.JMSContext.createContext(int)` émet toujours une exception `JMSRuntimeException`.
- `javax.jms.JMSContext.setClientID(String)` émet toujours une exception `JMSRuntimeException`.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` émet toujours une exception `JMSRuntimeException`.
- `javax.jms.JMSContext.stop()` émet toujours une exception `JMSRuntimeException`.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` émet toujours une exception `JMSRuntimeException`.

Utilisation IBM MQ classes for Java

Utilisez IBM MQ dans un environnement Java . IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

Remarque :

Stabilized IBM n'apportera pas d'autres améliorations à IBM MQ classes for Java ; cette fonctionnalité est stabilisée au niveau livré dans IBM MQ 8.0. Les applications existantes qui utilisent IBM MQ classes for Java continuent d'être entièrement prises en charge, mais les nouvelles fonctions ne seront pas ajoutées et les demandes d'amélioration seront rejetées. "Intégralement prises en charge" signifie que les incidents seront corrigés et que toute modification nécessaire suite à la modification de la configuration système requise pour IBM MQ sera apportée.

Les IBM MQ classes for Java ne sont pas pris en charge dans IMS.

Les IBM MQ classes for Java ne sont pas pris en charge dans WebSphere Liberty. Ils ne doivent pas être utilisés avec la fonction de messagerie IBM MQ Liberty ni avec le support JCA générique. Pour plus d'informations, voir Utilisation des interfaces Java WebSphere MQ dans les environnements J2EE/JEE.

IBM MQ classes for Java est l'une des trois autres API que les applications Java peuvent utiliser pour accéder aux ressources IBM MQ . Les autres API sont les suivantes:

- JM 3.0 IBM MQ classes for Jakarta Messaging
- JMS 2.0 IBM MQ classes for JMS

Pour plus d'informations, voir «Accès à IBM MQ depuis Java -Choix de l'API», à la page 89.

Depuis la IBM MQ 9.3, les IBM MQ classes for Java sont générées avec Java 8. L'environnement d'exécution Java 8 prend en charge l'exécution de versions de fichier de classe antérieures.

IBM MQ classes for Java encapsulez l'interface MQI (Message Queue Interface), l'API IBM MQ native et utilisez un modèle d'objet similaire aux interfaces C++ et .NET à IBM MQ.

Les options programmables permettent à IBM MQ classes for Java de se connecter à IBM MQ de l'une des manières suivantes:

- En mode client en tant que IBM MQ MQI client à l'aide du protocole TCP/IP (Transmission Control Protocol/Internet Protocol)
- En mode liaisons, connexion directe à IBM MQ à l'aide de l'interface JNI (Java Native Interface)

Remarque : La reconnexion automatique du client n'est pas prise en charge par IBM MQ classes for Java.

Connexion en mode client

Une application IBM MQ classes for Java peut se connecter à n'importe quel gestionnaire de files d'attente pris en charge à l'aide du mode client.

Pour se connecter à un gestionnaire de files d'attente en mode client, une application IBM MQ classes for Java peut s'exécuter sur le même système que celui sur lequel le gestionnaire de files d'attente s'exécute ou sur un autre système. Dans chaque cas, IBM MQ classes for Java se connecte au gestionnaire de files d'attente via TCP/IP.

Pour plus d'informations sur la façon d'écrire des applications pour utiliser des connexions en mode client, voir «Modes de connexion IBM MQ classes for Java», à la page 382.

Connexion en mode liaisons

Lorsqu'il est utilisé en mode liaisons, IBM MQ classes for Java utilise l'interface JNI (Java Native Interface) pour appeler directement l'API de gestionnaire de files d'attente existante, plutôt que de communiquer via un réseau. Dans la plupart des environnements, la connexion en mode liaisons offre de meilleures performances pour les applications IBM MQ classes for Java que la connexion en mode client, en évitant le coût des communications TCP/IP.

Les applications qui utilisent IBM MQ classes for Java pour se connecter en mode liaisons doivent s'exécuter sur le même système que le gestionnaire de files d'attente auquel elles se connectent.

L'environnement d'exécution Java , qui est utilisé pour exécuter l'application IBM MQ classes for Java , doit être configuré pour charger les bibliothèques IBM MQ classes for Java ; voir «IBM MQ classes for Java bibliothèques», à la page 367 pour plus d'informations.

Pour plus d'informations sur l'écriture d'applications afin d'utiliser des connexions en mode liaisons, voir «Modes de connexion IBM MQ classes for Java», à la page 382.

Concepts associés

Interfaces de langage IBM MQ Java

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

Tâches associées

Traçage des applications IBM MQ classes for Java
Identification et résolution des problèmes liés à Java et JMS

Pourquoi utiliser IBM MQ classes for Java ?

Une application Java peut utiliser IBM MQ classes for Java ou IBM MQ classes for JMS pour accéder aux ressources IBM MQ .

Remarque : Bien que les applications existantes qui utilisent le IBM MQ classes for Java continuent d'être entièrement prises en charge, les nouvelles applications doivent utiliser le IBM MQ classes for Jakarta Messaging. Les fonctions qui ont récemment été ajoutées à IBM MQ, telles que la consommation asynchrone et la reconnexion automatique, ne sont pas disponibles dans IBM MQ classes for Java, mais dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging. Pour plus d'informations, voir «Pourquoi utiliser IBM MQ classes for JMS ?», à la page 87 et «Pourquoi utiliser IBM MQ classes for Jakarta Messaging ?», à la page 86.

Remarque :

Stabilized IBM n'apportera pas d'autres améliorations à IBM MQ classes for Java ; cette fonctionnalité est stabilisée au niveau livré dans IBM MQ 8.0. Les applications existantes qui utilisent IBM MQ classes for Java continuent d'être entièrement prises en charge, mais les nouvelles fonctions ne seront pas ajoutées et les demandes d'amélioration seront rejetées. "Intégralement prises en charge" signifie que les incidents seront corrigés et que toute modification nécessaire suite à la modification de la configuration système requise pour IBM MQ sera apportée.

Les IBM MQ classes for Java ne sont pas pris en charge dans IMS.

Les IBM MQ classes for Java ne sont pas pris en charge dans WebSphere Liberty. Ils ne doivent pas être utilisés avec la fonction de messagerie IBM MQ Liberty ni avec le support JCA générique. Pour plus d'informations, voir Utilisation des interfaces Java WebSphere MQ dans les environnements J2EE/JEE.

Concepts associés

«Accès à IBM MQ depuis Java -Choix de l'API», à la page 89
IBM MQ fournit trois interfaces de langage Java .

Prérequis pour IBM MQ classes for Java

Pour utiliser IBM MQ classes for Java, vous avez besoin de certains autres produits logiciels.

Pour plus d'informations sur les prérequis pour IBM MQ classes for Java, voir la page Web Configuration système requise pour IBM MQ .

Pour développer des applications IBM MQ classes for Java , vous avez besoin d'un kit JDK (Java Development Kit). Les détails des JDK pris en charge avec votre système d'exploitation sont disponibles dans les informations Configuration système requise pour IBM MQ .

Pour exécuter des applications IBM MQ classes for Java , vous avez besoin des composants logiciels suivants:

- Un gestionnaire de files d'attente IBM MQ , pour les applications qui se connectent à un gestionnaire de files d'attente
- Un environnement d'exécution Java (JRE), pour chaque système sur lequel vous exécutez des applications. Un environnement d'exécution Java approprié est fourni avec IBM MQ.
- IBM i Pour IBM i, QShell, qui est l'option 30 du système d'exploitation
- z/OS Pour z/OS, z/OS UNIX System Services (z/OS UNIX)

Si vous avez besoin de connexions TLS pour utiliser des modules cryptographiques certifiés FIPS 140-2, vous avez besoin du fournisseur IBM Java JSSE FIPS (IBMJSSEFIPS). Chaque kit JDK et JRE IBM version 1.4.2 ou ultérieure contient IBMJSSEFIPS.

Vous pouvez utiliser des adresses Internet Protocol version 6 (IPv6) dans vos IBM MQ classes for Java applications si IPv6 est pris en charge par votre machine virtuelle Java (JVM) et l'implémentation TCP/IP sur votre système d'exploitation.

Exécution d'applications IBM MQ classes for Java dans Java EE

Certaines restrictions et considérations de conception doivent être prises en compte avant d'utiliser IBM MQ classes for Java dans Java EE.

IBM MQ classes for Java comporte des restrictions lorsqu'il est utilisé dans un environnement Java Platform, Enterprise Edition (Java EE). Des considérations supplémentaires doivent également être prises en compte lors de la conception, de l'implémentation et de la gestion d'une application IBM MQ classes for Java qui s'exécute dans un environnement Java EE . Ces restrictions et considérations sont décrites dans les sections suivantes.

Restrictions des transactions JTA

Le seul gestionnaire de transactions pris en charge pour les applications utilisant IBM MQ classes for Java est IBM MQ lui-même. Bien qu'une application sous contrôle JTA puisse utiliser IBM MQ classes for Java, tout travail effectué via ces classes n'est pas contrôlé par les unités de travail JTA. Ils forment à la place des unités de travail locales distinctes de celles gérées par le serveur d'applications via les interfaces JTA. En particulier, toute annulation de la transaction JTA ne se traduit pas par une annulation des messages envoyés ou reçus. Cette restriction s'applique aux transactions gérées par application ou bean et aux transactions gérées par conteneur, ainsi qu'à tous les conteneurs Java EE . Pour effectuer un travail de messagerie directement avec IBM MQ dans les transactions coordonnées par le serveur d'applications, IBM MQ classes for JMS doit être utilisé à la place.

Création d'unités d'exécution

IBM MQ classes for Java crée des unités d'exécution en interne pour diverses opérations. Par exemple, lors de l'exécution en mode BINDINGS pour appeler directement sur un gestionnaire de files d'attente local, les appels sont effectués sur une unité d'exécution'worker'créée en interne par IBM MQ classes for Java. D'autres unités d'exécution peuvent être créées en interne, par exemple pour effacer les connexions inutilisées d'un pool de connexions ou pour supprimer des abonnements pour les applications de publication / abonnement arrêtées.

Certaines applications Java EE (par exemple celles qui s'exécutent dans des conteneurs EJB et Web) ne doivent pas créer de nouvelles unités d'exécution. A la place, tous les travaux doivent être effectués sur les unités d'exécution d'application principales gérées par le serveur d'applications. Lorsque les applications utilisent IBM MQ classes for Java, il se peut que le serveur d'applications ne puisse pas faire la distinction entre le code d'application et le code IBM MQ classes for Java . Par conséquent, les unités d'exécution précédemment décrites entraînent la non-conformité de l'application avec la spécification de conteneur. IBM MQ classes for JMS ne brise pas ces spécifications Java EE et peut donc être utilisé à la place.

Restrictions de sécurité

Les règles de sécurité implémentées par un serveur d'applications peuvent empêcher certaines opérations effectuées par l'API IBM MQ classes for Java , telles que la création et l'exploitation de nouvelles unités d'exécution de contrôle (comme décrit dans les sections précédentes).

Par exemple, les serveurs d'applications s'exécutent généralement avec Java Security désactivé par défaut et permettent son activation via une configuration spécifique au serveur d'applications (certains serveurs d'applications permettent également une configuration plus détaillée des règles utilisées dans Java Security). Lorsque la sécurité Java est activée, IBM MQ classes for Java peut enfreindre les règles d'unités d'exécution de la stratégie de sécurité Java définies pour le serveur d'applications et l'API peut

ne pas être en mesure de créer toutes les unités d'exécution dont elle a besoin pour fonctionner. Pour éviter les problèmes liés à la gestion des unités d'exécution, l'utilisation de IBM MQ classes for Java n'est pas prise en charge dans les environnements où la sécurité Java est activée.

Considérations relatives à l'isolement des applications

L'un des avantages de l'exécution d'applications dans un environnement Java EE est l'isolement des applications. La conception et l'implémentation de IBM MQ classes for Java sont antérieures à l'environnement Java EE . IBM MQ classes for Java peut être utilisé d'une manière qui ne prend pas en charge le concept d'isolement d'application. Voici des exemples précis de considérations dans ce domaine:

- L'utilisation de paramètres statiques (à l'échelle du processus JVM) dans la classe MQEnvironment, tels que:
  - l'ID utilisateur et le mot de passe à utiliser pour l'identification et l'authentification de la connexion
  - le nom d'hôte, le port et le canal utilisés pour les connexions client
  - Configuration TLS pour les connexions client sécurisées

  La modification des propriétés MQEnvironment au profit d'une application affecte également d'autres applications utilisant les mêmes propriétés. Lors de l'exécution dans un environnement multi-applications tel que Java EE, chaque application doit utiliser sa propre configuration distincte en créant des objets MQQueueManager avec un ensemble spécifique de propriétés, au lieu d'utiliser par défaut les propriétés configurées dans la classe MQEnvironment à l'échelle du processus.
- La classe MQEnvironment introduit un certain nombre de méthodes statiques qui agissent globalement sur toutes les applications utilisant IBM MQ classes for Java dans le même processus JVM, et il n'est pas possible de remplacer ce comportement pour des applications particulières. En voici quelques exemples :
  - configuration des propriétés TLS, telles que l'emplacement du magasin de clés
  - configuration des exits de canal client
  - activation ou désactivation de la fonction de trace de diagnostic
  - gestion du pool de connexions par défaut utilisé pour optimiser l'utilisation des connexions aux gestionnaires de files d'attente

  L'appel de ces méthodes affecte toutes les applications exécutées dans le même environnement Java EE .
- Le regroupement de connexions est activé pour optimiser le processus d'établissement de plusieurs connexions au même gestionnaire de files d'attente. Le gestionnaire de pools de connexions par défaut s'applique à l'ensemble du processus et est partagé par plusieurs applications. Les modifications apportées à la configuration du pool de connexions, telles que le remplacement du gestionnaire de connexions par défaut pour une application à l'aide de la méthode MQEnvironment.setDefaultConnectionManager(), affectent donc les autres applications exécutées sur le même serveur d'applications Java EE.
- TLS est configuré pour les applications utilisant IBM MQ classes for Java à l'aide de la classe MQEnvironment et des propriétés d'objet MQQueueManager . Il n'est pas intégré à la configuration de sécurité gérée du serveur d'applications lui-même. Vous devez vous assurer que vous configurez IBM MQ classes for Java de manière appropriée pour fournir le niveau de sécurité requis et ne pas utiliser la configuration du serveur d'applications.

Restrictions du mode de liaison

IBM MQ et WebSphere Application Server peuvent être installés sur la même machine de sorte que les versions principales du gestionnaire de files d'attente et de l'adaptateur de ressources IBM MQ (RA) fourni dans WebSphere Application Server soient différentes.

Si les versions principales du gestionnaire de files d'attente et de l'adaptateur de ressources sont différentes, les connexions de liaisons ne peuvent pas être utilisées. Toute connexion de WebSphere

Application Server au gestionnaire de files d'attente à l'aide de l'adaptateur de ressources doit utiliser des connexions de type client. Les connexions de liaisons peuvent être utilisées si les versions sont identiques.

Conversions de chaînes de caractères dans IBM MQ classes for Java

Les IBM MQ classes for Java utilisent CharsetEncoders et CharsetDecoders directement pour la conversion de chaînes de caractères. Le comportement par défaut de la conversion de chaîne de caractères peut être configuré avec deux propriétés système. Le traitement des messages contenant des caractères non mappables peut être configuré via `com.ibm.mq.MQMD`.

Avant IBM MQ 8.0, les conversions de chaîne dans IBM MQ classes for Java étaient effectuées en appelant les méthodes `java.nio.charset.Charset.decode(ByteBuffer)` et `Charset.encode(CharBuffer)` .

L'utilisation de l'une ou l'autre de ces méthodes entraîne le remplacement par défaut (`REPLACE`) de données syntaxiquement incorrectes ou non traduisibles. Ce comportement peut masquer les erreurs dans les applications et entraîner des caractères inattendus, par exemple `?`, dans les données traduites.

Depuis la IBM MQ 8.0, pour détecter ces problèmes plus tôt et plus efficacement, les IBM MQ classes for Java utilisent directement CharsetEncoders et CharsetDecoders et configurent explicitement le traitement des données syntaxiquement incorrectes et non traduisibles. Le comportement par défaut consiste à `REPORT` de tels problèmes en émettant un `MQException`approprié.

Configuration

La conversion de UTF-16 (la représentation de caractères utilisée dans Java) en un jeu de caractères natif, tel que UTF-8, est appelée *codage*, tandis que la conversion dans la direction opposée est appelée *décodage*.

Le décodage utilise le comportement par défaut pour `CharsetDecoders`, qui signale les erreurs en émettant une exception.

Un paramètre est utilisé pour spécifier un paramètre `java.nio.charset.CodingErrorAction` afin de contrôler le traitement des erreurs lors du codage et du décodage. Un autre paramètre est utilisé pour contrôler le ou les octets de remplacement lors du codage. La chaîne de remplacement Java par défaut sera utilisée dans les opérations de décodage.

Configuration du traitement des données non traduisibles dans IBM MQ classes for Java

Depuis IBM MQ 8.0, `com.ibm.mq.MQMD` inclut les deux zones suivantes:

byte [ ] unMappableRemplacement
Séquence d'octets qui sera écrite dans une chaîne codée si un caractère d'entrée ne peut pas être traduit et que vous avez spécifié `REPLACE`.

Valeur par défaut: "?".getBytes()
La chaîne de remplacement Java par défaut est utilisée dans les opérations de décodage.

java.nio.charset.CodingErrorAction unmappableAction
Indique l'action à effectuer pour les données non traduisibles lors du codage et du décodage:

Valeur par défaut: CodingErrorAction.REPORT;

Propriétés système pour la définition des valeurs par défaut du système

Depuis la IBM MQ 8.0, les deux propriétés système Java suivantes sont disponibles pour configurer le comportement par défaut de la conversion de chaîne de caractères.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Indique l'action à effectuer pour les données non traduisibles lors du codage et du décodage. La valeur peut être `REPORT`, `REPLACE`ou `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Définit ou obtient les octets de remplacement à appliquer lorsqu'un caractère ne peut pas être mappé dans une opération de codage. La chaîne de remplacement Java par défaut est utilisée dans les opérations de décodage.

Pour éviter toute confusion entre les représentations de caractères Java et d'octets natifs, vous devez spécifier `com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement` en tant que nombre décimal représentant l'octet de remplacement dans le jeu de caractères natifs.

Par exemple, la valeur décimale de ?, en tant qu'octet natif, est 63 si le jeu de caractères natif est basé sur ASCII, tel que ISO-8859-1, et 111 si le jeu de caractères natif est EBCDIC.

Remarque : Notez que si les zones **`unmappableAction`** ou **`unMappableReplacement`** sont définies pour un objet MQMD ou MQMessage, les valeurs de ces zones sont prioritaires sur les propriétés système Java . Cela permet de remplacer les valeurs spécifiées par les propriétés système Java pour chaque message si nécessaire.

Concepts associés

«Conversions de chaînes de caractères dans IBM MQ classes for JMS», à la page 142
Les IBM MQ classes for JMS utilisent CharsetEncoders et CharsetDecoders directement pour la conversion de chaînes de caractères. Le comportement par défaut de la conversion de chaîne de caractères peut être configuré avec deux propriétés système. Le traitement des messages contenant des caractères non mappables peut être configuré via des propriétés de message pour définir l'action UnmappableCharacteret les octets de remplacement.

Installation et configuration de IBM MQ classes for Java

Cette section décrit les répertoires et les fichiers créés lors de l'installation de IBM MQ classes for Javaet explique comment configurer IBM MQ classes for Java après l'installation.

Ce qui est installé pour IBM MQ classes for Java

La version la plus récente d' IBM MQ classes for Java est installée avec IBM MQ. Vous devrez peut-être remplacer les options d'installation par défaut pour vous assurer que cette opération est effectuée.

Pour plus d'informations sur l'installation de IBM MQ , voir:

- Multi Installation de IBM MQ
- z/OS Installation du produit IBM MQ for z/OS

IBM MQ classes for Java sont contenus dans les fichiers d'archive Java (JAR), `com.ibm.mq.jar`et `com.ibm.mq.jmqi.jar`.

La prise en charge des en-têtes de message standard, tels que PCF (Programmable Command Format), est contenue dans le fichier JAR `com.ibm.mq.headers.jar`.

La prise en charge du format PCF (Programmable Command Format) est contenue dans le fichier JAR `com.ibm.mq.pcf.jar`.

Remarque : Il n'est pas recommandé d'utiliser IBM MQ classes for Java dans un serveur d'applications. Pour plus d'informations sur les restrictions qui s'appliquent lors de l'exécution dans cet environnement, voir «Exécution d'applications IBM MQ classes for Java dans Java EE», à la page 359. Pour plus d'informations, voir Utilisation des interfaces WebSphere MQ Java dans les environnements J2EE/JEE.

Important : Outre les fichiers JAR relocalisables décrits dans «Fichiers JAR IBM MQ classes for Java relocalisables», à la page 362, la copie des fichiers JAR ou des bibliothèques natives IBM MQ classes for Java sur d'autres machines ou à un autre emplacement sur une machine où IBM MQ classes for Java a été installé n'est pas prise en charge.

Fichiers JAR IBM MQ classes for Java relocalisables

Les fichiers JAR relocalisables peuvent être déplacés vers des systèmes qui doivent exécuter IBM MQ classes for Java.

Important :

- Outre les fichiers JAR relocalisables décrits dans Fichiers JAR relocalisables, la copie des fichiers JAR IBM MQ classes for Java ou des bibliothèques natives vers d'autres machines ou vers un autre emplacement sur une machine où IBM MQ classes for Java a été installé n'est pas prise en charge.
- Pour éviter les conflits de chargeur de classe, il n'est pas recommandé de regrouper les fichiers JAR relocalisables dans plusieurs applications au sein du même environnement d'exécution Java . Dans ce scénario, pensez à rendre les fichiers JAR relocalisables IBM MQ disponibles dans le chemin d'accès aux classes de l'environnement d'exécution Java .
- N'incluez pas les fichiers JAR relocalisables dans les applications déployées sur les serveurs d'applications Java EE , tels que WebSphere Application Server. Dans ces environnements, l'adaptateur de ressources IBM MQ doit être déployé et utilisé à la place, car il contient le IBM MQ classes for Java. Notez que WebSphere Application Server intègre l'adaptateur de ressources IBM MQ , il n'est donc pas nécessaire de le déployer manuellement dans cet environnement. En outre, les IBM MQ classes for Java ne sont pas pris en charge dans WebSphere Liberty. Pour plus d'informations, voir «Liberty et l'adaptateur de ressources IBM MQ», à la page 451.
- Si vous groupez les fichiers JAR relocalisables dans vos applications, veillez à inclure tous les fichiers JAR prérequis, comme décrit dans Fichiers JAR relocalisables. Vous devez également vous assurer que vous disposez des procédures appropriées pour mettre à jour les fichiers JAR intégrés dans le cadre de la maintenance de l'application, afin de vous assurer que IBM MQ classes for Java reste à jour et que les problèmes connus font l'objet d'une nouvelle médiation.

Fichiers JAR relocalisables

Dans une entreprise, les fichiers suivants peuvent être déplacés vers des systèmes qui doivent exécuter des applications IBM MQ classes for Java :

- JMS 2.0 `com.ibm.mq.allclient.jar` «1», à la page 363
- JM 3.0 `com.ibm.mq.jakarta.client.jar` «2», à la page 363
- V 9.4.0 `bcpkix-jdk18on.jar` «3», à la page 363
- `bcpkix-jdk15to18.jar` «4», à la page 363
- V 9.4.0 `bcprov-jdk18on.jar` «3», à la page 363
- `bcprov-jdk15to18.jar` «4», à la page 363
- V 9.4.0 `bcutil-jdk18on.jar` «3», à la page 363
- `bcutil-jdk15to18.jar` «4», à la page 363
- `org.json.jar`

Remarques :

1. JMS 2.0 et JMS 1.1
2. Jakarta Messaging 3.0
3. Du IBM MQ 9.4.0
4. AvantIBM MQ 9.4.0

Le fournisseur de sécurité Bouncy Castle et CMS prennent en charge les fichiers JAR

Le fournisseur de sécurité Bouncy Castle et les fichiers JAR de support CMS sont requis. Pour plus d'informations, voir Support for non-IBM JREs with AMS.

V 9.4.0 Les fichiers JAR suivants sont requis:

- `bcpkix-jdk18on.jar`
- `bcprov-jdk18on.jar`
- `bcutil-jdk18on.jar`

org.json.jar

Le fichier `org.json.jar` est requis si votre application IBM MQ classes for Java utilise une table de définition de canal du client au format JSON.

com.ibm.mq.allclient.jar et com.ibm.mq.jakarta.client.jar

Les fichiers `com.ibm.mq.allclient.jar` et `com.ibm.mq.jakarta.client.jar` contiennent les classes IBM MQ classes for JMS, IBM MQ classes for Javaet PCF et Headers. Si vous déplacez ces fichiers vers un nouvel emplacement, veillez à prendre les mesures nécessaires pour conserver ce nouvel emplacement avec les nouveaux groupes de correctifs IBM MQ . Assurez-vous également que l'utilisation des fichiers est connue du support IBM si vous obtenez un correctif temporaire.

Pour déterminer la version du fichier `com.ibm.mq.allclient.jar` ou du fichier `com.ibm.mq.jakarta.client.jar` , utilisez la commande suivante:

JM 3.0
```
java -jar com.ibm.mq.jakarta.client.jar
```

JMS 2.0
```
java -jar com.ibm.mq.allclient.jar
```

L'exemple suivant illustre un exemple de sortie de cette commande:

```
C:\Program Files\IBM\MQ_1\java\lib>java -jar com.ibm.mq.allclient.jar
Name:       Java Message Service Client
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ classes for Java Message Service
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       IBM MQ JMS Provider
Version:    9.3.0.0
Level:      p000-L140428.1 mqjbnd=p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar

Name:       Common Services for Java Platform, Standard Edition
Version:    9.3.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Répertoires d'installation pour IBM MQ classes for Java
Les fichiers et les exemples IBM MQ classes for Java sont installés dans des emplacements différents en fonction de la plateforme. L'emplacement de l'environnement d'exécution Java (JRE) installé avec IBM MQ varie également en fonction de la plateforme.

Répertoires d'installation des fichiers IBM MQ classes for Java

Tableau 49, à la page 364 indique l'emplacement d'installation des fichiers IBM MQ classes for Java .

Tableau 49. Répertoires d'installation de IBM MQ classes for Java

Plateforme	Répertoire
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib

Tableau 49. Répertoires d'installation de IBM MQ classes for Java (suite)	
Plateforme	**Répertoire**
Linux	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Répertoires d'installation des exemples

Certains modèles d'application, tels que les programmes de vérification d'installation (IVP), sont fournis avec IBM MQ. Tableau 50, à la page 365 indique l'emplacement d'installation des modèles d'application. Les exemples IBM MQ classes for Java se trouvent dans un sous-répertoire appelé `wmqjava`. Les exemples PCF se trouvent dans un sous-répertoire appelé `pcf`.

Tableau 50. Répertoires d'exemples	
Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples
Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/samples

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Répertoires d'installation de JRE

IBM MQ classes for JMS requiert un environnement d'exécution Java 7 (ou supérieur) Java Runtime Environment (JRE). Un environnement d'exécution Java approprié est installé avec IBM MQ. Tableau 51, à la page 365 indique l'emplacement d'installation de cet environnement d'exécution Java. Pour exécuter des programmes Java tels que les exemples fournis, à l'aide de cet environnement d'exécution Java (JRE), appelez explicitement *JRE_LOCATION*`/bin/java` ou ajoutez *JRE_LOCATION*`/bin` à l'environnement PATH (ou équivalent) de votre plateforme, où *JRE_LOCATION* est le répertoire indiqué dans Tableau 51, à la page 365.

Tableau 51. Répertoires JRE	
Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i	/QIBM/ProdData/mqm/java/jre
Linux	*MQ_INSTALLATION_PATH*/java/jre
Windows	*MQ_INSTALLATION_PATH*\java\jre
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/jre

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Variables d'environnement relatives à IBM MQ classes for Java
Si vous souhaitez exécuter des applications IBM MQ classes for Java , leur chemin d'accès aux classes doit inclure les répertoires IBM MQ classes for Java et des exemples.

Pour que les applications IBM MQ classes for Java s'exécutent, leur chemin d'accès aux classes doit inclure le répertoire IBM MQ classes for Java approprié. Pour exécuter les exemples d'application, le chemin d'accès aux classes doit également inclure les répertoires d'exemples appropriés. Ces informations peuvent être fournies dans la commande d'appel Java ou dans la variable d'environnement **CLASSPATH** .

Important : La définition de l'option Java `-Xbootclasspath` pour inclure le IBM MQ classes for Java n'est pas prise en charge.

Le Tableau 52, à la page 366 montre le paramètre **CLASSPATH** approprié à utiliser sur chaque plateforme pour exécuter des applications IBM MQ classes for Java , y compris les modèles d'application.

*Tableau 52. Paramètre **CLASSPATH** permettant d'exécuter des applications IBM MQ classes for Java , y compris les modèles d'application IBM MQ classes for Java*

Plateforme	CLASSPATH définition
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/wmqjava/samples:
Linux	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
z/OS	CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R4M0/java/samples/pcf

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Si vous effectuez la compilation à l'aide de l'option `-Xlint` , un message d'avertissement peut s'afficher indiquant que `com.ibm.mq.ese.jar` n'est pas présent. Vous pouvez ignorer l'avertissement. Ce fichier n'est présent que si vous avez installé Advanced Message Security.

Les scripts fournis avec IBM MQ classes for JMS utilisent les variables d'environnement suivantes:

MQ_JAVA_DATA_PATH
Cette variable d'environnement indique le répertoire de la sortie de journal et de trace.

MQ_JAVA_INSTALL_PATH
Cette variable d'environnement indique le répertoire dans lequel IBM MQ classes for Java est installé, comme indiqué dans les répertoires d'installationIBM MQ classes for Java.

MQ_JAVA_LIB_PATH
Cette variable d'environnement indique le répertoire dans lequel sont stockées les bibliothèques IBM MQ classes for Java , comme indiqué dans L'emplacement des bibliothèques IBM MQ classes for Java pour chaque plateforme. Certains scripts fournis avec IBM MQ classes for Java, tels qu'IVTRun, utilisent cette variable d'environnement.

Windows Sous Windows, toutes les variables d'environnement sont définies automatiquement lors de l'installation.

Linux AIX Sous AIX and Linux, vous pouvez utiliser le script `setjmsenv` (si vous utilisez une machine virtuelle Java 32 bits) ou `setjmsenv64` (si vous utilisez une machine virtuelle Java 64 bits) pour définir les variables d'environnement. Ces scripts se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/java/bin .

IBM i Sous IBM i, la variable d'environnement **QIBM_MULTI_THREADED** doit être définie sur Y. Vous pouvez ensuite exécuter des applications à unités d'exécution multiples de la même manière que vous exécutez des applications à unités d'exécution uniques. Pour plus d'informations, voir Configuration de IBM MQ avec Java et JMS.

IBM MQ classes for Java requiert un environnement d'exécution Java 7 Java (JRE). Pour plus d'informations sur l'emplacement d'un JRE approprié installé avec IBM MQ, voir «Répertoires d'installation pour IBM MQ classes for Java», à la page 364.

IBM MQ classes for Java bibliothèques
L'emplacement des bibliothèques IBM MQ classes for Java varie en fonction de la plateforme. Indiquez cet emplacement lorsque vous démarrez une application.

Pour spécifier l'emplacement des bibliothèques JNI (Java Native Interface), démarrez votre application à l'aide d'une commande **java** au format suivant:

```
java -Djava.library.path= library_path application_name
```

où *chemin_bibliothèque* est le chemin d'accès à IBM MQ classes for Java, qui inclut les bibliothèques JNI. Tableau 53, à la page 367 indique l'emplacement des bibliothèques IBM MQ classes for Java pour chaque plateforme. Dans ce tableau, *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Tableau 53. Emplacement des bibliothèques IBM MQ classes for Java pour chaque plateforme

Plateforme	**Répertoire contenant les bibliothèques IBM MQ classes for Java**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Linux Linux (plateformex86)	*MQ_INSTALLATION_PATH*/java/lib
Linux Linux (plateformes POWER, x86-64 et zSeries s390x)	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliothèques 64 bits)
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java/lib (bibliothèques 32 bits et 64 bits)

Remarque :

1. Linux AIX Sous AIX ou Linux (plateforme Power), utilisez les bibliothèques 32 bits ou 64 bits. Utilisez les bibliothèques 64 bits uniquement si vous exécutez votre application dans une machine virtuelle Java (JVM) 64 bits sur une plateforme 64 bits. Sinon, utilisez les bibliothèques 32 bits.
2. Windows Sous Windows, vous pouvez utiliser la variable d'environnement PATH pour spécifier l'emplacement des bibliothèques IBM MQ classes for Java au lieu de spécifier leur emplacement dans la commande **java** .
3. IBM i Pour utiliser IBM MQ classes for Java en mode liaisons sous IBM i, vérifiez que la bibliothèque QMQMJAVA figure dans votre liste de bibliothèques.
4. z/OS Sous z/OS, vous pouvez utiliser une machine virtuelle Java (JVM) 32 bits ou 64 bits. Vous n'avez pas besoin de spécifier les bibliothèques à utiliser ; IBM MQ classes for Java peut déterminer pour lui-même les bibliothèques JNI à charger.

Concepts associés

Utilisation IBM MQ classes for Java
Après avoir installé IBM MQ classes for Java, vous pouvez configurer votre installation afin d'exécuter vos propres applications.

Prise en charge d'OSGi avec IBM MQ classes for Java
OSGi fournit une infrastructure qui prend en charge le déploiement d'applications en tant que bundles. Trois bundles OSGi sont fournis dans le cadre de IBM MQ classes for Java.

OSGi fournit une infrastructure Java générale, sécurisée et gérée, qui prend en charge le déploiement des applications fournies sous la forme de bundles. Les unités conformes à OSGi peuvent télécharger et installer des bundles et les supprimer lorsqu'ils ne sont plus nécessaires. L'infrastructure gère l'installation et la mise à jour des bundles de manière dynamique et évolutive.

Les IBM MQ classes for Java incluent les bundles OSGi suivants.

com.ibm.mq.osgi.java_*version_number*.jar
Fichiers JAR permettant aux applications d'utiliser IBM MQ classes for Java.

com.ibm.mq.jakarta.osgi.allclient_*version_number*.jar
JM 3.0 Pour Jakarta Messaging 3.0, ce fichier JAR permet aux applications d'utiliser à la fois IBM MQ classes for JMS et IBM MQ classes for Java, et inclut également le code permettant de gérer les messages PCF.

com.ibm.mq.osgi.allclient_*version_number*.jar
JMS 2.0 Pour JMS 2.0, ce fichier JAR permet aux applications d'utiliser à la fois IBM MQ classes for JMS et IBM MQ classes for Java, et inclut également le code permettant de gérer les messages PCF.

com.ibm.mq.jakarta.osgi.allclientprereqs_*version_number*.jar
JM 3.0 Pour Jakarta Messaging 3.0, ce fichier JAR fournit les prérequis pour `com.ibm.mq.jakarta.osgi.allclient_version_number.jar`.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
JMS 2.0 Pour JMS 2.0, ce fichier JAR fournit les prérequis pour `com.ibm.mq.osgi.allclient_version_number.jar`.

où *version_number* est le numéro de version de IBM MQ qui est installé.

Les bundles sont installés dans le sous-répertoire `java/lib/OSGi` de votre installation IBM MQ ou dans le dossier `java\lib\OSGi` sous Windows.

Depuis IBM MQ 8.0, utilisez les bundles `com.ibm.mq.osgi.allclient_8.0.0.0.jar`et `com.ibm.mq.osgi.allclientprereqs_8.0.0.0.jar` pour toute nouvelle application. L'utilisation de ces bundles supprime la restriction de ne pas pouvoir exécuter à la fois IBM MQ classes for JMS et IBM MQ classes for Java dans la même infrastructure OSGi. Toutes les autres restrictions s'appliquent

toutefois. Pour les versions de IBM MQ antérieures à IBM MQ 8.0, la restriction d'utilisation de IBM MQ classes for JMS ou de IBM MQ classes for Java s'applique.

Neuf autres bundles sont également installés dans le sous-répertoire `java/lib/OSGi` de votre installation IBM MQ ou dans le dossier `java\lib\OSGi` sous Windows. Ces bundles font partie de IBM MQ classes for JMSet ne doivent pas être chargés dans un environnement d'exécution OSGi dans lequel le bundle IBM MQ classes for Java est chargé. Si le bundle OSGi IBM MQ classes for Java est chargé dans un environnement d'exécution OSGi dans lequel les bundles IBM MQ classes for JMS sont également chargés, des erreurs se produisent, comme illustré dans l'exemple suivant, lorsque des applications utilisant le bundle IBM MQ classes for Java ou les bundles IBM MQ classes for JMS sont exécutées:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

Le bundle OSGi pour IBM MQ classes for Java a été écrit dans la spécification OSGi Release 4 ; il ne fonctionne pas dans un environnement OSGi Release 3.

Vous devez définir correctement le chemin du système ou le chemin de la bibliothèque pour que l'environnement d'exécution OSGi puisse trouver les fichiers DLL ou les bibliothèques partagées requis.

Si vous utilisez le bundle OSGi pour IBM MQ classes for Java, les classes d'exit de canal écrites en Java ne sont pas prises en charge en raison d'un problème inhérent au chargement des classes dans un environnement de chargeur de classes multiples tel qu'OSGi. Un bundle d'utilisateurs peut connaître le bundle IBM MQ classes for Java , mais le bundle IBM MQ classes for Java ne connaît aucun bundle d'utilisateurs. Par conséquent, le chargeur de classe utilisé dans un bundle IBM MQ classes for Java ne peut pas charger une classe d'exit de canal qui se trouve dans un bundle utilisateur.

Pour plus d'informations sur OSGi, voir le site Web OSGi alliance .

z/OS *Installation of IBM MQ classes for Java on z/OS*

On z/OS, the STEPLIB used at runtime must contain the IBM MQ SCSQAUTH and SCSQANLE libraries.

From z/OS UNIX System Services, you can add these libraries by using a line in your `.profile` as shown in the following example, replacing `thlqual` with the high level data set qualifier that you chose when installing IBM MQ:

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

In other environments, you typically need to edit the startup JCL to include SCSQAUTH on the STEPLIB concatenation:

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

Le fichier de configuration IBM MQ classes for Java

Un fichier de configuration IBM MQ classes for Java spécifie les propriétés utilisées pour configurer IBM MQ classes for Java.

Le format d'un fichier de configuration IBM MQ classes for Java est celui d'un fichier de propriétés Java standard.

Un exemple de fichier de configuration, `mqjava.config`, est fourni dans le sous-répertoire `bin` du répertoire d'installation IBM MQ classes for Java . Ce fichier documente toutes les propriétés prises en charge et leurs valeurs par défaut.

Remarque : L'exemple de fichier de configuration est remplacé lorsque l'installation de IBM MQ est mise à niveau vers un futur groupe de correctifs. Par conséquent, il est recommandé d'effectuer une copie de l'exemple de fichier de configuration à utiliser avec vos applications.

Vous pouvez choisir le nom et l'emplacement d'un fichier de configuration IBM MQ classes for Java . Lorsque vous démarrez votre application, utilisez une commande **java** au format suivant:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

Dans la commande, *config_file_url* est un URL (uniform resource locator) qui spécifie le nom et l'emplacement du fichier de configuration IBM MQ classes for Java . Les URL des types suivants sont prises en charge: `http`, `file`, `ftp`et `jar`.

L'exemple suivant illustre une commande **java** :

```
java -Dcom.ibm.msg.client.config.location=file:/D:/mydir/mqjava.config MyAppClass
```

Cette commande identifie le fichier de configuration IBM MQ classes for Java en tant que fichier `D:\mydir\mqjava.config` sur le système Windows local.

Lorsqu'une application démarre, IBM MQ classes for Java lit le contenu du fichier de configuration et stocke les propriétés spécifiées dans un magasin de propriétés interne. Si la commande **java** n'identifie pas de fichier de configuration ou si le fichier de configuration est introuvable, IBM MQ classes for Java utilise les valeurs par défaut pour toutes les propriétés. Si nécessaire, vous pouvez remplacer n'importe quelle propriété dans le fichier de configuration en la spécifiant comme propriété système dans la commande **java** .

Un fichier de configuration IBM MQ classes for Java peut être utilisé avec tous les transports pris en charge entre une application et un gestionnaire de files d'attente ou un courtier.

Remplacement des propriétés spécifiées dans un fichier de configuration IBM MQ MQI client

Un fichier de configuration IBM MQ MQI client peut également spécifier les propriétés utilisées pour configurer IBM MQ classes for Java. Toutefois, les propriétés spécifiées dans un fichier de configuration IBM MQ MQI client s'appliquent uniquement lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client.

Si nécessaire, vous pouvez remplacer n'importe quel attribut dans un fichier de configuration IBM MQ MQI client en le spécifiant en tant que propriété dans un fichier de configuration IBM MQ classes for Java . Pour remplacer un attribut dans un fichier de configuration IBM MQ MQI client , utilisez une entrée au format suivant dans le fichier de configuration IBM MQ classes for Java :

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Les variables de l'entrée ont les significations suivantes:

section
Nom de la section dans le fichier de configuration IBM MQ MQI client qui contient l'attribut.

propName
Nom de l'attribut tel qu'il est spécifié dans le fichier de configuration IBM MQ MQI client .

propValue
Valeur de la propriété qui remplace la valeur de l'attribut spécifiée dans le fichier de configuration IBM MQ MQI client .

Vous pouvez également remplacer un attribut dans un fichier de configuration IBM MQ MQI client en spécifiant la propriété en tant que propriété système dans la commande **java** . Utilisez le format précédent pour spécifier la propriété en tant que propriété système.

Seuls les attributs suivants d'un fichier de configuration IBM MQ MQI client sont pertinents pour IBM MQ classes for Java. Si vous spécifiez ou remplacez d'autres attributs, cela n'a aucun effet. En particulier, notez que les fichiers `ChannelDefinitionFile` et `ChannelDefinitionDirectory` de la section CHANNELS du fichier de configuration client ne sont pas utilisés. Pour plus de détails sur l'utilisation de

la table de définition de canal du client avec IBM MQ classes for Java, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for Java», à la page 386 .

Tableau 54. Quelle section du fichier de configuration du client contient quel attribut

section	Attribut
Strophe CHANNELS du fichier de configuration client	Put1DefaultAlwaysSync
Strophe CHANNELS du fichier de configuration client	PasswordProtection
ClientExitPath du fichier de configuration du client	ExitsDefaultPath
ClientExitPath du fichier de configuration du client	ExitsDefaultPath64
ClientExitPath du fichier de configuration du client	JavaExitsChemin d'accès aux classes
Strophe JMQI du fichier de configuration du client	useMQCSPauthentication
sectionMessageBuffer du fichier de configuration du client	MaximumSize
sectionMessageBuffer du fichier de configuration du client	PurgeTime
sectionMessageBuffer du fichier de configuration du client	UpdatePercentage
Strophe TCP du fichier de configuration du client	ClntRcvBuffSize
Strophe TCP du fichier de configuration du client	ClntSndBuffSize
Strophe TCP du fichier de configuration du client	Connect_Timeout
Strophe TCP du fichier de configuration du client	KeepAlive

Pour plus d'informations sur la configuration IBM MQ MQI client , voir le fichier de configuration IBM MQ MQI client , `mqclient.ini`.

Tâches associées

Traçage des classes IBM MQ pour les applications Java

Utilisation de Java Standard Environment Trace pour configurer la trace Java

Utilisez la section Paramètres de trace d'environnement standard Java pour configurer la fonction de trace IBM MQ classes for Java .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputNom*

traceOutputName est le répertoire et le nom de fichier dans lesquels la sortie de trace est envoyée.

Par défaut, les informations de trace sont écrites dans un fichier de trace dans le répertoire de travail en cours de l'application. Le nom du fichier de trace dépend de l'environnement dans lequel l'application s'exécute:

- Si l'application a chargé le fichier IBM MQ classes for Java à partir du fichier JAR relocalisable `com.ibm.mq.allclient.jar`, la trace est écrite dans un fichier appelé `mqjavaclient_%PID%.cl%u.trc`.
- Si l'application a chargé le fichier IBM MQ classes for Java à partir du fichier JAR `com.ibm.mq.jar`, la trace est écrite dans un fichier appelé `mqjava_%PID%.cl%u.trc`.

où *%PID%* est l'identificateur de processus de l'application qui est tracée, et *%u* est un numéro unique permettant de différencier les fichiers entre les unités d'exécution qui exécutent la trace sous différents chargeurs de classe Java.

Si un ID de processus n'est pas disponible, un nombre aléatoire est généré et précédé de la lettre `f`. Pour inclure l'ID de processus dans un nom de fichier que vous spécifiez, utilisez la chaîne `%PID%`.

Si vous spécifiez un autre répertoire, il doit exister et vous devez disposer d'un droit d'accès en écriture pour ce répertoire. Si vous ne disposez pas de droits d'accès en écriture, la sortie de trace est écrite dans `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList est une liste de packages et de classes tracés, ou les valeurs spéciales `ALL` ou `NONE`.

Séparez les noms de package ou de classe par un point-virgule, `;`. *includeList* prend par défaut la valeur `ALL`et trace tous les packages et classes dans IBM MQ classes for Java.

Remarque : Vous pouvez inclure un package, mais exclure ensuite les sous-packages de ce package. Par exemple, si vous incluez le package `a.b` et le package d'exclusion `a.b.x`, la trace inclut tout ce qui se trouve dans `a.b.y` et `a.b.z`, mais pas dans `a.b.x` ou `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList est une liste de packages et de classes qui ne sont pas tracés, ou les valeurs spéciales `ALL` ou `NONE`.

Séparez les noms de package ou de classe par un point-virgule, `;`. *excludeList* prend par défaut la valeur `NONE`et n'exclut donc aucun package et aucune classe dans IBM MQ classes for JMS de la trace.

Remarque : Vous pouvez exclure un package, puis inclure des sous-packages de ce package. Par exemple, si vous excluez le package `a.b` et incluez le package `a.b.x`, la trace inclut tout ce qui se trouve dans `a.b.x` et `a.b.x.1`, mais pas dans `a.b.y` ou `a.b.z`.

Tout package ou classe spécifié, au même niveau, comme inclus et exclu, est inclus.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayOctets*
maxArrayBytes est le nombre maximal d'octets tracés à partir de n'importe quel tableau d'octets.

Si *maxArrayBytes* est défini sur un entier positif, il limite le nombre d'octets dans un tableau d'octets qui sont écrits dans le fichier de trace. Il tronque le tableau d'octets après avoir écrit *maxArrayBytes* . La définition de *maxArrayBytes* réduit la taille du fichier de trace résultant et réduit l'effet de la fonction de trace sur les performances de l'application.

La valeur `0` pour cette propriété signifie qu'aucun contenu d'un tableau d'octets n'est envoyé au fichier de trace.

La valeur par défaut est `-1`, ce qui supprime toute limite sur le nombre d'octets d'un tableau d'octets envoyés au fichier de trace.

com.ibm.msg.client.commonservices.trace.limit = *maxTraceOctets*
maxTraceBytes est le nombre maximal d'octets écrits dans un fichier de sortie de trace.

maxTraceBytes fonctionne avec *traceCycles*. Si le nombre d'octets de trace écrits est proche de la limite, le fichier est fermé et un nouveau fichier de sortie de trace est démarré.

La valeur `0` signifie qu'un fichier de sortie de trace a une longueur nulle. La valeur par défaut est `-1`, ce qui signifie que la quantité de données à écrire dans un fichier de sortie de trace est illimitée.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles est le nombre de fichiers de sortie de trace à parcourir.

Si le fichier de sortie de trace en cours atteint la limite spécifiée par *maxTraceBytes*, le fichier est fermé. Une sortie de trace supplémentaire est écrite dans le fichier de sortie de trace suivant dans l'ordre. Chaque fichier de sortie de trace se distingue par un suffixe numérique ajouté au nom de fichier. Le fichier de sortie de trace en cours ou le plus récent est `mqjms.trc.0`, le prochain fichier de sortie de trace le plus récent est `mqjms.trc.1`. Les anciens fichiers de trace suivent le même modèle de numérotation jusqu'à la limite.

La valeur par défaut de *traceCycles* est 1. Si *traceCycles* a la valeur 1, lorsque le fichier de sortie de trace en cours atteint sa taille maximale, le fichier est fermé et supprimé. Un nouveau fichier de

sortie de trace portant le même nom est démarré. Par conséquent, il n'existe qu'un seul fichier de sortie de trace à la fois.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters contrôle si les paramètres de méthode et les valeurs de retour sont inclus dans la trace.

traceParameters prend par défaut la valeur TRUE. Si *traceParameters* est défini sur FALSE, seules les signatures de méthode sont tracées.

com.ibm.msg.client.commonservices.trace.startup = *démarrage*
Il existe une phase d'initialisation de IBM MQ classes for Java au cours de laquelle des ressources sont allouées. La fonction de trace principale est initialisée lors de la phase d'allocation de ressources.

Si *startup* est défini sur TRUE, la trace de démarrage est utilisée. Les informations de trace sont générées immédiatement et incluent la configuration de tous les composants, y compris la fonction de trace elle-même. Les informations de trace de démarrage peuvent être utilisées pour diagnostiquer les problèmes de configuration. Les informations de trace de démarrage sont toujours écrites dans `System.err`.

startup prend par défaut la valeur FALSE.

startup est vérifié avant la fin de l'initialisation. Pour cette raison, spécifiez uniquement la propriété sur la ligne de commande en tant que propriété système Java . Ne l'indiquez pas dans le fichier de configuration IBM MQ classes for Java .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Définissez *compressedTrace* sur TRUE pour compresser la sortie de trace.

La valeur par défaut de *compressedTrace* est FALSE.

Si *compressedTrace* est défini sur TRUE, la sortie de trace est compressée. Le nom du fichier de sortie de trace par défaut porte l'extension `.trz`. Si la compression est définie sur FALSE, la valeur par défaut, le fichier a l'extension `.trc` pour indiquer qu'il est décompressé. Toutefois, si le nom de fichier de la sortie de trace a été spécifié dans *traceOutputName* , ce nom est utilisé à la place ; aucun suffixe n'est appliqué au fichier.

La sortie de trace compressée est plus petite que la sortie non compressée. Etant donné qu'il y a moins d'E-S, il peut être écrit plus rapidement que la trace non compressée. La fonction de trace compressée a moins d'effet sur les performances de IBM MQ classes for Java que la fonction de trace non compressée.

Si *maxTraceBytes* et *traceCycles* sont définis, plusieurs fichiers de trace compressés sont créés à la place de plusieurs fichiers à plat.

Si IBM MQ classes for Java se termine de manière non contrôlée, il se peut qu'un fichier de trace compressé ne soit pas valide. Pour cette raison, la compression de trace ne doit être utilisée que lorsque IBM MQ classes for Java se ferme de manière contrôlée. N'utilisez la compression de trace que si les problèmes examinés ne provoquent pas l'arrêt inattendu de la machine virtuelle Java elle-même. N'utilisez pas la compression de trace pour diagnostiquer les problèmes qui peuvent entraîner des arrêts de la machine virtuelle `System.Halt()` ou des arrêts anormaux et non contrôlés de la machine virtuelle Java.

com.ibm.msg.client.commonservices.trace.level = *traceLevel*
traceLevel indique un niveau de filtrage pour la trace. Les niveaux de trace définis sont les suivants:

- `TRACE_NONE: 0`
- `TRACE_EXCEPTION: 1`
- `TRACE_WARNING: 3`
- `TRACE_INFO: 6`
- `TRACE_ENTRYEXIT: 8`
- `TRACE_DATA: 9`
- `TRACE_ALL: Integer.MAX_VALUE`

Chaque niveau de trace inclut tous les niveaux inférieurs. Par exemple, si le niveau de trace est défini sur TRACE_INFO, tout point de trace dont le niveau défini est TRACE_EXCEPTION, TRACE_WARNINGou TRACE_INFO est consigné dans la trace. Tous les autres points de trace sont exclus.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace contrôle si le service de traçage du client IBM MQ classes for Java est utilisé dans un environnement WebSphere Application Server .

Si *standaloneTrace* est défini sur TRUE, les propriétés de trace du client IBM MQ classes for Java sont utilisées pour déterminer la configuration de la trace.

Si *standaloneTrace* est défini sur FALSEet que le client IBM MQ classes for Java s'exécute dans un conteneur WebSphere Application Server , le service de trace WebSphere Application Server est utilisé. Les informations de trace générées dépendent des paramètres de trace du serveur d'applications.

La valeur par défaut de *standaloneTrace* est FALSE.

IBM MQ classes for Java et outils de gestion de logiciels
Les outils de gestion de logiciels tels que Apache Maven peuvent être utilisés avec IBM MQ classes for Java.

De nombreuses grandes organisations de développement utilisent ces outils pour gérer de manière centralisée les référentiels de bibliothèques tierces.

Les IBM MQ classes for Java sont composés d'un certain nombre de fichiers JAR. Lorsque vous développez des applications en langage Java à l'aide de cette API, une installation d'un serveur IBM MQ , d'un client IBM MQ ou d'un IBM MQ client SupportPac est requise sur la machine sur laquelle l'application est développée.

Si vous souhaitez utiliser un outil de gestion de logiciels et ajouter les fichiers JAR qui constituent le IBM MQ classes for Java à un référentiel géré de manière centralisée, les points suivants doivent être observés:

- Un référentiel ou un conteneur ne doit être mis à la disposition que des développeurs de votre organisation. Toute distribution en dehors de l'organisation n'est pas autorisée.
- Le référentiel doit contenir un ensemble complet et cohérent de fichiers JAR provenant d'une seule édition ou d'un seul groupe de correctifs IBM MQ .
- Vous êtes responsable de la mise à jour du référentiel avec toute maintenance fournie par le support IBM .

Depuis IBM MQ 8.0, le fichier JAR `com.ibm.mq.allclient.jar` doit être installé dans le référentiel.

Depuis la IBM MQ 9.0, le fournisseur de sécurité Bouncy Castle et les fichiers JAR de support CMS sont requis. Pour plus d'informations, voir «Fichiers JAR IBM MQ classes for Java relocalisables», à la page 362 et Support for non-IBM JREs.

Configuration post-installation pour les applications IBM MQ classes for Java

Après avoir installé IBM MQ classes for Java, vous pouvez configurer votre installation afin d'exécuter vos propres applications.

N'oubliez pas de consulter le fichier Readme du produit IBM MQ pour obtenir les informations les plus récentes ou des informations plus spécifiques sur votre environnement. La dernière version du fichier Readme du produit est disponible sur la page Web Fichiers Readme des produits IBM MQ, WebSphere MQet MQSeries .

Avant de tenter d'exécuter une application IBM MQ classes for Java en mode liaisons, vérifiez que vous avez configuré IBM MQ comme décrit dans Configuration.

Configuration de votre gestionnaire de files d'attente pour l'acceptation des connexions client depuis IBM MQ classes for Java
Pour configurer votre gestionnaire de files d'attente afin qu'il accepte les demandes de connexion entrantes des clients, définissez et autorisez l'utilisation d'un canal de connexion serveur et démarrez un programme d'écoute.

Pour plus d'informations, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.

Exécution d'applications IBM MQ classes for Java sous Java security manager
IBM MQ classes for Java peut s'exécuter avec le Java security manager activé. Pour exécuter correctement des applications avec Java security manager activé, vous devez configurer votre Java Virtual Machine (JVM) avec un fichier de définition de règle approprié.

Le moyen le plus simple de créer un fichier de définition de règle approprié consiste à modifier le fichier de règles fourni avec Java runtime environment (JRE). Sur la plupart des systèmes, ce fichier est stocké dans le répertoire `path lib/security/java.policy`, relatif à votre répertoire JRE. Vous pouvez éditer les fichiers de règles à l'aide de l'éditeur de votre choix ou à l'aide du programme policytool fourni avec votre environnement d'exécution Java.

Vous devez accorder des droits d'accès au fichier `com.ibm.mq.jmqi.jar` pour qu'il puisse:

- Créer des sockets (en mode client)
- Charger la bibliothèque native (en mode liaisons)
- Lire diverses propriétés à partir de l'environnement

La propriété système **`os.name`** doit être disponible pour IBM MQ classes for Java lors de l'exécution sous Java security manager.

Si votre application Java utilise Java security manager, vous devez ajouter les droits suivants au fichier `java.security.policy` utilisé par l'application. Sinon, des exceptions sont émises à l'application:

```
permission java.lang.RuntimePermission "modifyThread";
```

Ce droit d'exécution est requis par le client pour la gestion de l'affectation et de la clôture de conversations multiplexées sur des connexions TCP/IP à des gestionnaires de files d'attente.

Exemple d'entrée de fichier de règles

Voici un exemple d'entrée de fichier de règles qui permet à IBM MQ classes for Java de s'exécuter correctement sous le gestionnaire de sécurité par défaut. Remplacez la chaîne *MQ_INSTALLATION_PATH* dans cet exemple par l'emplacement où IBM MQ classes for Java est installé sur votre système.

```
grant codeBase "file: MQ_INSTALLATION_PATH/java/lib/*" {
//We need access to these properties, mainly for tracing
permission java.util.PropertyPermission "user.name","read";
permission java.util.PropertyPermission "os.name","read";
permission java.util.PropertyPermission "user.dir","read";
permission java.util.PropertyPermission "line.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.log.*","read";
permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.trace.*","read";
permission java.util.PropertyPermission "Diagnostics.Java.Errors.Destination.Filename","read";
permission java.util.PropertyPermission "com.ibm.mq.commonservices","read";
permission java.util.PropertyPermission "com.ibm.mq.cfg.*","read";

//Tracing - we need the ability to control java.util.logging
permission java.util.logging.LoggingPermission "control";
// And access to create the trace file and read the log file - assumed to be in the current
directory
permission java.io.FilePermission "*","read,write";

// Required to allow a trace file to be written to the filesystem.
// Replace 'TRACE_FILE_DIRECTORY' with the directory name where trace is to be written to
permission java.io.FilePermission "TRACE_FILE_DIRECTORY","read,write";
permission java.io.FilePermission "TRACE_FILE_DIRECTORY/*","read,write";
```

```
// We'd like to set up an mBean to control trace
permission javax.management.MBeanServerPermission "createMBeanServer";
permission javax.management.MBeanPermission "*","*";

// We need to be able to read manifests etc from the jar files in the installation directory
permission java.io.FilePermission "MQ_INSTALLATION_PATH/java/lib/-","read";

//Required if mqclient.ini/mqs.ini configuration files are used
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqclient.ini","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/mqs.ini","read";

//For the client transport type.
permission java.net.SocketPermission "*","connect,resolve";

//For the bindings transport type.
permission java.lang.RuntimePermission "loadLibrary.*";

//For applications that use CCDT tables (access to the CCDT AMQCLCHL.TAB)
permission java.io.FilePermission "MQ_DATA_DIRECTORY/qmgrs/QM_NAME/@ipcc/AMQCLCHL.TAB","read";

//For applications that use User Exits
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits/*","read";
permission java.io.FilePermission "MQ_DATA_DIRECTORY/exits64/*","read";
permission java.lang.RuntimePermission "createClassLoader";

//Required for the z/OS platform
permission java.util.PropertyPermission "com.ibm.vm.bitmode","read";

// Used by the internal ConnectionFactory implementation
permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

// Used for controlled class loading
permission java.lang.RuntimePermission "setContextClassLoader";

// Used to default the Application name in Client mode connections
permission java.util.PropertyPermission "sun.java.command","read";

// Used by the IBM JSSE classes
permission java.util.PropertyPermission "com.ibm.crypto.provider.AESNITrace","read";

//Required to determine if an IBM Java Runtime is running in FIPS mode,
//and to modify the property values status as required.
permission java.util.PropertyPermission "com.ibm.jsse2.usefipsprovider","read,write";
permission java.util.PropertyPermission "com.ibm.jsse2.JSSEFIPS","read,write";
//Required if an IBM FIPS provider is to be used for SSL communication.
permission java.security.SecurityPermission "insertProvider.IBMJCEFIPS";

// Required for non-IBM Java Runtimes that establish secure client
// transport mode connections using mutual TLS authentication
permission java.util.PropertyPermission "javax.net.ssl.keyStore","read";
permission java.util.PropertyPermission "javax.net.ssl.keyStorePassword","read";

// Required for Java applications that use the Java Security Manager
permission java.lang.RuntimePermission "modifyThread";
};
```

Cet exemple de fichier de règles permet à IBM MQ classes for Java de fonctionner correctement sous le gestionnaire de sécurité, mais vous devrez peut-être encore activer votre propre code pour qu'il s'exécute correctement avant que vos applications ne fonctionnent.

L'exemple de code fourni avec IBM MQ classes for Java n'a pas été spécifiquement activé pour être utilisé avec le gestionnaire de sécurité ; toutefois, les tests IVT s'exécutent avec ce fichier de règles et le gestionnaire de sécurité par défaut en place.

Important :

La fonction de trace IBM MQ classes for Java requiert des droits d'accès supplémentaires car elle effectue des requêtes supplémentaires sur les propriétés système, ainsi que des opérations supplémentaires sur le système de fichiers.

Un modèle de fichier de règles de sécurité adapté à l'exécution sous un gestionnaire de sécurité avec la fonction de trace activée est fourni dans le répertoire `samples/wmqjava` de l'installation IBM MQ en tant que `example.security.policy`.

Pour une installation par défaut, le fichier `example.security.policy` se trouve dans:

Windows
dans `C:\Program Files\IBM\MQ\Tools\wmqjava\samples\example.security.policy`

Linux
dans `/opt/mqm/samp/wmqjava/samples/example.security.policy`

Solaris
dans `/opt/mqm/samp/wmqjava/samples/example.security.policy`

AIX
dans `/usr/mqm/samp/wmqjava/samples/example.security.policy`

Running IBM MQ classes for Java applications under CICS Transaction Server
An IBM MQ classes for Java application can be run as a transaction under CICS Transaction Server.

To run an IBM MQ classes for Java application as a transaction under CICS Transaction Server for z/OS, perform the following steps:

1. Define the application and transaction to CICS by using the supplied CEDA transaction.
2. Ensure that the IBM MQ CICS adapter is installed in your CICS system. z/OS (See Using IBM MQ with CICS for details.)
3. Ensure that the JVM environment specified in CICS includes the appropriate CLASSPATH and LIBPATH entries.
4. Initiate the transaction by using any of your normal processes.

For more information on running CICS Java transactions, refer to your CICS system documentation.

Vérification de l'installation d'IBM MQ classes for Java

Un programme de vérification de l'installation, MQIVP, est fourni avec IBM MQ classes for Java. Vous pouvez utiliser ce programme pour tester tous les modes de connexion de IBM MQ classes for Java.

Le programme vous invite à indiquer un certain nombre de choix et d'autres données afin de déterminer le mode de connexion à vérifier. Utilisez la procédure suivante pour vérifier votre installation:

1. Si vous prévoyez d'exécuter le programme en mode client, configurez votre gestionnaire de files d'attente comme décrit dans «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098. La file d'attente à utiliser est SYSTEM.DEFAULT.LOCAL.QUEUE
2. Si vous prévoyez d'exécuter le programme en mode client, voir aussi «Utilisation IBM MQ classes for Java», à la page 356.

   Effectuez les étapes restantes de cette procédure sur le système sur lequel vous allez exécuter le programme.
3. Vérifiez que vous avez mis à jour votre variable d'environnement CLASSPATH conformément aux instructions de la rubrique «Variables d'environnement relatives à IBM MQ classes for Java», à la page 366.
4. Accédez au répertoire *MQ_INSTALLATION_PATH*`/mqm/samp/wmqjava/samples`, où *MQ_INSTALLATION_PATH* est le chemin d'accès à votre installation IBM MQ . Ensuite, à l'invite de commande, entrez:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   où *chemin_bibliothèque* est le chemin d'accès aux bibliothèques IBM MQ classes for Java (voir «IBM MQ classes for Java bibliothèques», à la page 367).

   A l'invite marquée (1):

   - Pour utiliser une connexion TCP/IP, entrez un nom d'hôte de serveur IBM MQ .
   - Pour utiliser la connexion native (mode liaisons), laissez la zone vide (n'entrez pas de nom).

   Le programme tente de:

- 1. Connexion au gestionnaire de files d'attente
- 2. Ouvrez la file d'attente SYSTEM.DEFAULT.LOCAL.QUEUE, insertion d'un message dans la file d'attente, extraction d'un message de la file d'attente, puis fermeture de la file d'attente
- 3. Déconnexion du gestionnaire de files d'attente
- 4. Renvoie un message si les opérations aboutissent

Voici un exemple des invites et des réponses que vous pouvez voir. Les invites réelles et vos réponses dépendent de votre réseau IBM MQ .

```
Please enter the IP address of the MQ server    : ipaddress(1)
Please enter the port to connect to             : (1414) (2)
Please enter the server connection channel name : channelname (2)
Please enter the queue manager name             : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Remarque :

1. z/OS Sous z/OS, laissez la zone vide à l'invite marquée (1).
2. Si vous choisissez la connexion serveur, les invites marquées (2)ne s'affichent pas.
3. IBM i Sous IBM i, vous pouvez uniquement exécuter la commande `java MQIVP` à partir de QShell. Vous pouvez également exécuter l'application à l'aide de la commande CL `RUNJVA CLASS(MQIVP)`.

Utilisation des modèles d'application IBM MQ classes for Java

Les exemples d'application IBM MQ classes for Java fournissent une présentation des fonctions communes de l'API IBM MQ classes for Java . Vous pouvez les utiliser pour vérifier la configuration de votre installation et de votre serveur de messagerie et pour vous aider à créer vos propres applications.

Pourquoi et quand exécuter cette tâche

Si vous avez besoin d'aide pour créer vos propres applications, vous pouvez utiliser les modèles d'application comme base de départ. La version source et la version compilée sont fournies pour chaque application. Examinez l'exemple de code source et identifiez les étapes clés pour créer chaque objet requis pour votre application (MQQueueManager, MQConstants, MQMessage, MQPutMessageOptions et MQDestination), et pour définir les propriétés spécifiques requises pour spécifier le mode de fonctionnement de votre application. Pour plus d'informations, voir «Ecriture d'applications IBM MQ classes for Java», à la page 381. Les exemples peuvent être sujets à des modifications dans les éditions futures de IBM MQ Java.

La Tableau 55, à la page 378 montre où les modèles d'application IBM MQ classes for Java sont installés sur chaque plateforme:

Tableau 55. Répertoires d'installation des exemples d'application IBM MQ classes for Java

Plateforme	Répertoire
AIX AIX Linux Linux	*MQ_INSTALLATION_PATH*`/samp/wmqjava/samples`

Tableau 55. Répertoires d'installation des exemples d'application IBM MQ classes for Java (suite)	
Plateforme	**Répertoire**
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\samples
IBM i	/qibm/proddata/mqm/java/samples/wmqjava/samples
z/OS	*MQ_INSTALLATION_PATH*/java/samples/wmqjava

La Tableau 56, à la page 379 présente les ensembles d'exemples d'application fournis avec IBM MQ classes for Java.

Tableau 56. IBM MQ classes for Java modèles d'application	
Nom du modèle	**Description**
IMSBridgeSample.java	Programme simple pour illustrer l'utilisation de IMS Bridge avec IBM MQ classes for Java.
MQIVP.java	Programme de vérification de l'installation de IBM MQ Java .
MQMessagePropertiesSample.java	Démontre l'utilisation de l'API Propriétés de message.
MQPubSubApiSample.java	Démontre l'utilisation de l'API de publication / abonnement.
MQSample.java	Programme simple pour illustrer l'insertion et l'obtention d'un message à partir d'une file d'attente.
MQSampleMessageManager.java	Classe utilitaire pour le traitement des messages dans les exemples IBM MQ de base Java .
mqjcivp.properties	Ce regroupement de ressources contient les messages utilisés par le programme de vérification de l'installation IBM MQ classes for Java (MQIVP.java).

IBM MQ classes for Java fournit un script appelé runjms qui peut être utilisé pour exécuter les modèles d'application. Ce script configure l'environnement IBM MQ pour vous permettre d'exécuter les modèles d'application IBM MQ classes for Java .

Tableau 57, à la page 379 indique l'emplacement du script sur chaque plateforme:

Tableau 57. Emplacement du script runjms	
Plateforme	**Répertoire**
AIX Linux	*MQ_INSTALLATION_PATH*/java/bin/runjms
Windows	*MQ_INSTALLATION_PATH*\java\bin\runjms.bat
IBM i	/qibm/proddata/mqm/java/bin/runjms ou /qibm/proddata/mqm/java/bin/runjms64
z/OS	*MQ_INSTALLATION_PATH*java/bin/runjms

Pour utiliser le script `runjms` afin d'appeler un exemple d'application, procédez comme suit:

Procédure

1. Ouvrez une invite de commande et accédez au répertoire contenant le modèle d'application que vous souhaitez exécuter.
2. Entrez la commande suivante :

   ```
   Path to the runjms script/runjms sample_application_name
   ```

   Le modèle d'application affiche la liste des paramètres dont il a besoin.
3. Entrez la commande suivante pour exécuter l'exemple avec ces paramètres:

   ```
   Path to the runjms script/runjms sample_application_name parameters
   ```

Exemple

Linux Par exemple, pour exécuter l'exemple MQIVP sous Linux, entrez les commandes suivantes:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Concepts associés

«Ce qui est installé pour IBM MQ classes for JMS», à la page 92
Un certain nombre de fichiers et de répertoires sont créés lorsque vous installez IBM MQ classes for JMS. Sous Windows, une partie de la configuration est effectuée lors de l'installation en définissant automatiquement des variables d'environnement. Sur d'autres plateformes et dans certains environnements Windows , vous devez définir des variables d'environnement avant de pouvoir exécuter des applications IBM MQ classes for JMS .

Résolution des problèmes liés à IBM MQ classes for Java

Exécutez initialement le programme de vérification de l'installation. Vous devrez peut-être également utiliser la fonction de trace.

Si une application ne se termine pas correctement, exécutez le programme de vérification de l'installation et suivez les conseils donnés dans les messages de diagnostic. Le programme de vérification de l'installation est décrit dans «Vérification de l'installation d'IBM MQ classes for Java», à la page 377.

Si les problèmes persistent et que vous devez contacter l'équipe de maintenance IBM , vous pouvez être invité à activer la fonction de trace. Procédez comme indiqué dans l'exemple suivant.

Pour tracer le programme `MQIVP` :

- Créez un fichier de propriétés `com.ibm.mq.commonservices` (voir Utilisation de com.ibm.mq.commonservices).
- Entrez la commande suivante :

  ```
  java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
  -Djava.library.path= library_path MQIVP -trace
  ```

où :

- *commonservices_properties_file* est le chemin d'accès (y compris le nom de fichier) au fichier de propriétés `com.ibm.mq.commonservices` .
- *chemin_bibliothèque* est le chemin d'accès aux bibliothèques IBM MQ classes for Java (voir «IBM MQ classes for Java bibliothèques», à la page 367).

Pour plus d'informations sur l'utilisation de la fonction de trace, voir Traçage des applications IBM MQ classes for Java.

z/OS MQ Adv. VUE Java client connectivity to batch applications running on z/OS

Under certain conditions, an IBM MQ classes for Java application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

By using a client connection, an IBM MQ classes for Java application can connect to a remote z/OS queue manager if the following conditions apply:

- The application is running in a batch environment.
- Le gestionnaire de files d'attente connecté est en cours d'exécution avec l'autorisation d'utilisation d' IBM MQ Advanced for z/OS Value Unit Edition et par conséquent, le paramètre **ADVCAP** est défini sur ENABLED.

Pour plus d'informations sur IBM MQ Advanced for z/OS Value Unit Edition , voir IBM MQ product Identifier and export information.

Voir DISPLAY QMGR pour plus d'informations sur **ADVCAP** et START QMGR pour plus d'informations sur **QMGRPROD**.

An IBM MQ classes for Java application on z/OS cannot use a client mode connection to connect to a queue manager that is not running on z/OS

If an IBM MQ classes for Java application on z/OS attempts to connect using client mode, and is not allowed to do so, MQRC_ENVIRONMENT_ERROR is returned.

Advanced Message Security (AMS) support

IBM MQ classes for Java client applications can use AMS when connecting to remote z/OS queue managers, subject to the conditions previously described in this topic.

Pour utiliser AMS de cette manière, les applications client doivent utiliser un type de magasin de clés `jceracfks` dans `keystore.conf`, où:

- Le préfixe de nom de propriété est `jceracfks` et ce préfixe de nom est insensible à la casse.
- Le magasin de clés est un fichier de clés RACF .
- Les mots de passe ne sont pas obligatoires et seront ignorés. En effet, les fichiers de clés RACF n'utilisent pas de mots de passe.
- Si vous spécifiez le fournisseur, celui-ci doit être `IBMJCE`.

Lorsque vous utilisez `jceracfks` avec AMS, le magasin de clés doit être au format suivant: `safkeyring://`*`user`*`/`*`keyring`*, où:

- `safkeyring` est un littéral et ce nom est insensible à la casse
- *`user`* est l'ID utilisateur RACF qui possède le fichier de clés
- *`keyring`* est le nom du fichier de clés RACF et le nom du fichier de clés est sensible à la casse

L'exemple suivant utilise le fichier de clés AMS standard pour l'utilisateur `JOHNDOE`:

```
jceracfks.keystore=safkeyring://JOHNDOE/drq.ams.keyring
```

Related concepts

“JMS/Jakarta Messaging client connectivity to batch applications running on z/OS” on page 130
Under certain conditions, an IBM MQ classes for JMS/Jakarta Messaging application on z/OS can connect to a queue manager on z/OS by using a client connection. Use of a client connection can simplify IBM MQ topologies.

Ecriture d'applications IBM MQ classes for Java

Cette collection de rubriques fournit des informations pour vous aider à écrire des applications Java afin d'interagir avec les systèmes IBM MQ .

Pour utiliser IBM MQ classes for Java afin d'accéder aux files d'attente IBM MQ , vous devez écrire des applications Java contenant des appels qui placent des messages dans des files d'attente IBM MQ et en extraire des messages. Pour plus de détails sur les classes individuelles, voir IBM MQ classes for Java.

Remarque : La reconnexion automatique du client n'est pas prise en charge par IBM MQ classes for Java.

Interface du IBM MQ classes for Java

L'interface de programme d'application IBM MQ procédurale utilise des verbes qui agissent sur les objets. L'interface de programmation Java utilise des objets sur lesquels vous agissez en appelant des méthodes.

L'interface de programme d'application IBM MQ procédurale est construite autour d'instructions telles que les suivantes:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Ces instructions prennent toutes, en tant que paramètre, un descripteur de l'objet IBM MQ sur lequel elles doivent fonctionner. Votre programme se compose d'un ensemble d'objets IBM MQ sur lesquels vous agissez en appelant des méthodes sur ces objets.

Lorsque vous utilisez l'interface de procédure, vous vous déconnectez d'un gestionnaire de files d'attente à l'aide de l'appel MQDISC (Hconn, CompCode, Reason), où *Hconn* est un descripteur du gestionnaire de files d'attente.

Dans l'interface Java , le gestionnaire de files d'attente est représenté par un objet de classe MQQueueManager. Vous vous déconnectez du gestionnaire de files d'attente en appelant la méthode disconnect () sur cette classe.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

Modes de connexion IBM MQ classes for Java

La façon dont vous programmez pour IBM MQ classes for Java comporte des dépendances sur les modes de connexion que vous souhaitez utiliser.

Si vous utilisez des connexions client, il existe un certain nombre de différences par rapport à IBM MQ MQI client , mais elles sont similaires sur le plan conceptuel. Si vous utilisez le mode liaisons, vous pouvez utiliser des liaisons de raccourci et exécuter la commande MQBEGIN. Vous spécifiez le mode à utiliser en définissant des variables dans la classe MQEnvironment.

IBM MQ classes for Java connexions client

Lorsque IBM MQ classes for Java est utilisé en tant que client, il est similaire à IBM MQ MQI client, mais présente un certain nombre de différences.

Si vous programmez *IBM MQ classes for Java* pour une utilisation en tant que client, tenez compte des différences suivantes:

- Il ne prend en charge que TCP/IP.
- Il ne lit aucune variable d'environnement IBM MQ au démarrage.
- Les informations qui seraient stockées dans une définition de canal et dans des variables d'environnement peuvent être stockées dans une classe appelée Environnement. Ces informations peuvent également être transmises en tant que paramètres lors de la connexion.
- Les conditions d'erreur et d'exception sont consignées dans un journal spécifié dans la classe `MQException` . La destination d'erreur par défaut est la console Java .

- Seuls les attributs suivants d'un fichier de configuration client IBM MQ sont pertinents pour IBM MQ classes for Java. Si vous spécifiez d'autres attributs, ils sont inefficaces.

section	Attribut
ClientExitPath du fichier de configuration du client	ExitsDefaultPath
ClientExitPath du fichier de configuration du client	ExitsDefaultPath64
ClientExitPath du fichier de configuration du client	JavaExitsClasspath
sectionMessageBuffer du fichier de configuration du client	MaximumSize
sectionMessageBuffer du fichier de configuration du client	PurgeTime
sectionMessageBuffer du fichier de configuration du client	UpdatePercentage
Strophe TCP du fichier de configuration du client	ClntRcvBuffSize
Strophe TCP du fichier de configuration du client	ClntSndBuffSize
Strophe TCP du fichier de configuration du client	Connect_Timeout
Strophe TCP du fichier de configuration du client	KeepAlive

- Si vous vous connectez à un gestionnaire de files d'attente qui requiert la conversion de données de type caractères, le client V7 Java est désormais capable d'effectuer la conversion si le gestionnaire de files d'attente ne peut pas le faire. La machine virtuelle Java client doit prendre en charge la conversion entre le CCSID du client et celui du gestionnaire de files d'attente.
- La reconnexion client automatique n'est pas prise en charge par IBM MQ classes for Java.

Lorsqu'il est utilisé en mode client, *IBM MQ classes for Java* ne prend pas en charge l'appel MQBEGIN.

IBM MQ classes for Java Mode LIAISONS
Le mode de liaison de IBM MQ classes for Java diffère du mode client de trois manières principales.

Lorsqu'il est utilisé en mode liaisons, IBM MQ classes for Java utilise l'interface JNI (Java Native Interface) pour appeler directement l'API de gestionnaire de files d'attente existante, plutôt que de communiquer via un réseau.

Par défaut, les applications qui utilisent IBM MQ classes for Java en mode liaisons se connectent à un gestionnaire de files d'attente à l'aide de *ConnectOption*, MQCNO_STANDARD_BINDINGS.

Les IBM MQ classes for Java prennent en charge les *ConnectOptions*suivantes:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_LIEN_PARTAGE
- MQCNO_LIEN_ISOLÉ_LIAISON

Pour plus d'informations sur *ConnectOptions*, voir «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 755.

Le mode liaisons prend en charge l'appel MQBEGIN pour initier des unités de travail globales qui sont coordonnées par le gestionnaire de files d'attente, sur toutes les plateformes à l'exception de IBM MQ for IBM i et IBM MQ for z/OS.

La plupart des paramètres fournis par la classe `MQEnvironment` ne sont pas pertinents pour le mode liaisons et sont ignorés.

Définition de la connexion IBM MQ classes for Java à utiliser
Le type de connexion à utiliser est déterminé par la définition des variables dans la classe MQEnvironment.

Deux variables sont utilisées:

MQEnvironment.properties
Le type de connexion est déterminé par la valeur associée au nom de clé `CMQC.TRANSPORT_PROPERTY`. Les valeurs admises sont les suivantes :

CMQC.TRANSPORT_MQSERIES_BINDINGS
Se connecter en mode liaisons

CMQC.TRANSPORT_MQSERIES_CLIENT
Se connecter en mode client

CMQC.TRANSPORT_MQSERIES
Le mode de connexion est déterminé par la valeur de la propriété *hostname*

MQEnvironment.hostname
Définissez la valeur de cette variable comme suit:

- Pour les connexions client, définissez la valeur de cette variable sur le nom d'hôte du serveur IBM MQ auquel vous souhaitez vous connecter
- Pour le mode liaisons, ne définissez pas cette variable ou définissez-la sur null

Opérations sur les gestionnaires de files d'attente
Cette collection de rubriques décrit comment se connecter et se déconnecter d'un gestionnaire de files d'attente à l'aide d' IBM MQ classes for Java.

Configuration de l'environnement IBM MQ pour IBM MQ classes for Java
Pour qu'une application puisse se connecter à un gestionnaire de files d'attente en mode client, elle doit spécifier le nom de canal, le nom d'hôte et le numéro de port.

Remarque : Les informations de cette rubrique ne sont pertinentes que si votre application se connecte à un gestionnaire de files d'attente en mode client. Il n'est pas pertinent s'il se connecte en mode liaisons. Voir «Modes de connexion pour IBM MQ classes for JMS», à la page 113

Vous pouvez spécifier le nom de canal, le nom d'hôte et le numéro de port de deux manières: en tant que zones dans la classe MQEnvironment ou en tant que propriétés de l'objet MQQueueManager .

Si vous définissez des zones dans la classe MQEnvironment, elles s'appliquent à l'ensemble de votre application, sauf lorsqu'elles sont remplacées par une table de hachage de propriétés. Pour spécifier le nom de canal et le nom d'hôte dans MQEnvironment, utilisez le code suivant:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel = "java.client.channel";
```

Cela revient à définir une variable d'environnement **MQSERVER** :

```
"java.client.channel/TCP/host.domain.com".
```

Par défaut, les clients Java tentent de se connecter à un programme d'écoute IBM MQ sur le port 1414. Pour spécifier un port différent, utilisez le code suivant:

```
MQEnvironment.port = nnnn;
```

où nnnn est le numéro de port requis

Si vous transmettez des propriétés à un objet de gestionnaire de files d'attente lors de sa création, elles s'appliquent uniquement à ce gestionnaire de files d'attente. Créez des entrées dans un objet Hashtable avec les clés **hostname**, **channel**et, facultativement, **port**, et avec les valeurs appropriées. Pour utiliser

le port par défaut, 1414, vous pouvez omettre l'entrée **port** . Créez l'objet MQQueueManager à l'aide d'un constructeur qui accepte la table de hachage des propriétés.

Identification d'une connexion au gestionnaire de files d'attente en définissant un nom d'application

Une application peut définir un nom identifiant sa connexion au gestionnaire de files d'attente. Ce nom d'application est affiché par la commande **DISPLAY CONN MQSC/PCF** (où la zone est appelée **APPLTAG**) ou dans l'écran **Connexions d'application** de IBM MQ Explorer (où la zone est appelée **App name**).

Les noms d'application sont limités à 28 caractères, de sorte que les noms plus longs sont tronqués. Si aucun nom d'application n'est spécifié, une valeur par défaut est fournie. Le nom par défaut est basé sur la classe (principale) appelante, mais si ces informations ne sont pas disponibles, le texte `IBM MQ Client for Java` est utilisé.

Si le nom de la classe appelante est utilisé, il est ajusté pour tenir en supprimant les noms de package de début, si nécessaire. Par exemple, si la classe appelante est `com.example.MainApp`, le nom complet est utilisé, mais si la classe appelante est `com.example.dictionaryAndThesaurus.multilingual.mainApp`, le nom `multilingual.mainApp` est utilisé, car il s'agit de la combinaison la plus longue du nom de classe et du nom de package le plus à droite qui correspond à la longueur disponible.

Si le nom de classe lui-même comporte plus de 28 caractères, il est tronqué pour tenir. Par exemple, `com.example.mainApplicationForSecondTestCase` devient `mainApplicationForSecondTest`.

Pour définir un nom d'application dans la classe MQEnvironment, ajoutez le nom à la table de hachage MQEnvironment.properties , avec la clé **MQConstants.APPNAME_PROPERTY**, à l'aide du code suivant:

```
MQEnvironment.properties.put(MQConstants.APPNAME_PROPERTY, "my_application_name");
```

Pour définir un nom d'application dans la table de hachage des propriétés qui est transmise au constructeur MQQueueManager , ajoutez le nom à la table de hachage des propriétés avec la clé **MQConstants.APPNAME_PROPERTY**.

Remplacement des propriétés spécifiées dans un fichier de configuration client IBM MQ

Un fichier de configuration client IBM MQ peut également spécifier les propriétés utilisées pour configurer IBM MQ classes for Java. Toutefois, les propriétés spécifiées dans un fichier de configuration IBM MQ MQI client s'appliquent uniquement lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client.

Si nécessaire, vous pouvez remplacer n'importe quel attribut dans un fichier de configuration IBM MQ de l'une des manières suivantes. Les options sont affichées par ordre de priorité.

- Définissez une propriété système Java pour la propriété de configuration.
- Définissez la propriété dans la mappe MQEnvironment.properties .
- Sous Java5 et éditions ultérieures, définissez une variable d'environnement système.

Seuls les attributs suivants d'un fichier de configuration client IBM MQ sont pertinents pour IBM MQ classes for Java. Si vous spécifiez ou remplacez d'autres attributs, cela n'a aucun effet.

section	Attribut
ClientExitPath du fichier de configuration du client	ExitsDefaultPath
ClientExitPath du fichier de configuration du client	ExitsDefaultPath64
ClientExitPath du fichier de configuration du client	JavaExitsClasspath

section	Attribut
sectionMessageBuffer du fichier de configuration du client	MaximumSize
sectionMessageBuffer du fichier de configuration du client	PurgeTime
sectionMessageBuffer du fichier de configuration du client	UpdatePercentage
Strophe TCP du fichier de configuration du client	ClntRcvBufSize
Strophe TCP du fichier de configuration du client	ClntSndBufSize
Strophe TCP du fichier de configuration du client	Connect_Timeout
Strophe TCP du fichier de configuration du client	KeepAlive

Connexion à un gestionnaire de files d'attente dans IBM MQ classes for Java
Connectez-vous à un gestionnaire de files d'attente en créant une nouvelle instance de la classe MQQueueManager . Déconnectez-vous d'un gestionnaire de files d'attente en appelant la méthode disconnect ().

Vous êtes maintenant prêt à vous connecter à un gestionnaire de files d'attente en créant une nouvelle instance de la classe MQQueueManager :

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Pour vous déconnecter d'un gestionnaire de files d'attente, appelez la méthode disconnect () sur le gestionnaire de files d'attente:

```
queueManager.disconnect();
```

Si vous appelez la méthode de déconnexion, toutes les files d'attente ouvertes et tous les processus auxquels vous avez accédé via ce gestionnaire de files d'attente sont fermés. Toutefois, il est recommandé de fermer ces ressources de manière explicite lorsque vous avez fini de les utiliser. Pour ce faire, utilisez la méthode close () sur les objets appropriés.

Les méthodes commit () et backout () d'un gestionnaire de files d'attente sont équivalentes aux appels MQCMIT et MQBACK utilisés avec l'interface procédurale.

Utilisation d'une table de définition de canal du client avec IBM MQ classes for Java
Une application client IBM MQ classes for Java peut utiliser des définitions de canal de connexion client stockées dans une table de définition de canal du client (CCDT).

Au lieu de créer une définition de canal de connexion client en définissant certaines zones et propriétés d'environnement dans la classe `MQEnvironment` ou en les transmettant à un `MQQueueManager` dans une table de hachage de propriétés, une application client IBM MQ classes for Java peut utiliser des définitions de canal de connexion client stockées dans une table de définition de canal client. Ces définitions sont créées à l'aide de commandes IBM MQ Script (MQSC) ou de commandes PCF (IBM MQ Programmable Command Format), ou à l'aide de IBM MQ Explorer .

Lorsque l'application crée un objet `MQQueueManager` , le client IBM MQ classes for Java recherche dans la table de définition de canal du client une définition de canal de connexion client appropriée et utilise la définition de canal pour démarrer un canal MQI. Pour plus d'informations sur les tables de définition de canal du client et sur la façon d'en créer une, voir Table de définition de canal du client.

Pour utiliser une table de définition de canal du client, une application doit d'abord créer un objet URL. L'objet URL encapsule un URL qui identifie le nom et l'emplacement du fichier contenant la table de définition de canal du client et indique comment accéder au fichier.

Par exemple, si le fichier `ccdt1.tab` contient une table de définition de canal du client et qu'il est stocké sur le même système que celui sur lequel l'application est en cours d'exécution, l'application peut créer un objet URL de la manière suivante:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

Comme autre exemple, supposons que le fichier `ccdt2.tab` contienne une table de définition de canal du client et qu'il soit stocké sur un système différent de celui sur lequel l'application s'exécute. Si le fichier est accessible à l'aide du protocole FTP, l'application peut créer un objet URL de la manière suivante:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

Une fois que l'application a créé un objet URL, elle peut créer un objet `MQQueueManager` à l'aide de l'un des constructeurs qui utilise un objet URL comme paramètre. Par exemple :

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

Cette instruction permet au client IBM MQ classes for Java d'accéder à la table de définition de canal du client identifiée par l'objet URL chanTab2, de rechercher dans la table une définition de canal de connexion client appropriée, puis d'utiliser la définition de canal pour démarrer un canal MQI vers le gestionnaire de files d'attente appelé MARS.

Notez les points suivants qui s'appliquent si une application utilise une table de définition de canal du client:

- Lorsque l'application crée un objet `MQQueueManager` à l'aide d'un constructeur qui prend un objet URL comme paramètre, aucun nom de canal ne doit être défini dans la classe `MQEnvironment` , que ce soit en tant que zone ou en tant que propriété d'environnement. Si un nom de canal est défini, le client IBM MQ classes for Java émet une exception `MQException`. La zone ou la propriété d'environnement spécifiant le nom de canal est considérée comme étant définie si sa valeur est autre que null, une chaîne vide ou une chaîne contenant tous les caractères blancs.
- Le paramètre **queueManagerName** sur le constructeur `MQQueueManager` peut avoir l'une des valeurs suivantes:
  - Nom d'un gestionnaire de files d'attente
  - Un astérisque (*) suivi du nom d'un groupe de gestionnaires de files d'attente
  - Un astérisque (*)
  - Null, une chaîne vide ou une chaîne contenant tous les caractères vides

  Ces valeurs sont les mêmes que celles qui peuvent être utilisées pour le paramètre **QMgrName** sur un appel MQCONN émis par une application client qui utilise l'interface MQI (Message Queue Interface). Pour plus d'informations sur la signification de ces valeurs, voir«Présentation de l'interface de file d'attente de messages», à la page 739.

  Si votre application utilise le regroupement de connexions, voir«Contrôle du pool de connexions par défaut dans IBM MQ classes for Java», à la page 407.
- Lorsque le client IBM MQ classes for Java trouve une définition de canal de connexion client appropriée dans la table de définition de canal du client, il utilise uniquement les informations extraites de cette définition de canal pour démarrer un canal MQI. Les zones de canal ou les propriétés d'environnement que l'application peut avoir définies dans la classe `MQEnvironment` sont ignorées.

  En particulier, notez les points suivants si vous utilisez le protocole TLS (Transport Layer Security):
  - Un canal MQI utilise TLS uniquement si la définition de canal extraite de la table de définitions de canal du client indique le nom d'un CipherSpec pris en charge par le client IBM MQ classes for Java .
  - Une table de définition de canal du client contient également des informations sur l'emplacement des serveurs LDAP (Lightweight Directory Access Protocol) qui contiennent des listes de révocation

de certificats (CRL). Le client IBM MQ classes for Java utilise uniquement ces informations pour accéder aux serveurs LDAP qui contiennent des listes de révocation de certificat.

– Une table de définition de canal du client peut également contenir l'emplacement d'un répondeur OCSP. IBM MQ classes for Java ne peut pas utiliser les informations OCSP dans un fichier de table de définition de canal du client. Toutefois, vous pouvez configurer OCSP comme décrit dans la section Utilisation du protocole de certificat en ligne

Pour plus d'informations sur l'utilisation de TLS avec une table de définition de canal du client, voir Spécification du fait qu'un canal MQI utilise TLS.

Notez également les points suivants si vous utilisez des exits de canal:

– Un canal MQI utilise les exits de canal et les données utilisateur associées spécifiées par la définition de canal extraite de la table de définition de canal du client, de préférence aux exits de canal et aux données spécifiées à l'aide d'autres méthodes.

– Une définition de canal extraite d'une table de définition de canal du client peut spécifier des exits de canal écrits en Java, C ou C + +. Pour plus d'informations sur l'écriture d'un exit de canal dans Java , voir«Création d'un exit de canal dans IBM MQ classes for Java», à la page 401. Pour plus d'informations sur l'écriture d'un exit de canal dans d'autres langues, voir «Utilisation d'exits de canal non écrits dans Java avec IBM MQ classes for Java», à la page 404.

Spécification d'une plage de ports pour les connexions client IBM MQ classes for Java
Vous pouvez spécifier un port ou une plage de ports auxquels une application peut être liée de deux manières.

Lorsqu'une application IBM MQ classes for Java tente de se connecter à un gestionnaire de files d'attente IBM MQ en mode client, un pare-feu peut autoriser uniquement les connexions provenant de ports spécifiés ou d'une plage de ports. Dans cette situation, vous pouvez spécifier un port, ou une plage de ports, auquel l'application peut être liée. Vous pouvez spécifier le ou les ports de l'une des manières suivantes:

- Vous pouvez définir la zone localAddressdans la classe MQEnvironment. Par exemple :

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- Vous pouvez définir la propriété d'environnement CMQC.LOCAL_ADDRESS_PROPERTY. Par exemple :

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Lorsque vous pouvez construire l'objet MQQueueManager , vous pouvez transmettre une table de hachage de propriétés contenant une propriété LOCAL_ADDRESS_PROPERTY avec la valeur "192.0.2.0(2000,3000)"

Dans chacun de ces exemples, lorsque l'application se connecte ultérieurement à un gestionnaire de files d'attente, elle se lie à une adresse IP locale et à un numéro de port compris entre 192.0.2.0(2000) et 192.0.2.0(3000).

Dans un système comportant plusieurs interfaces réseau, vous pouvez également utiliser la zone localAddressSetting ou la propriété d'environnement CMQC.LOCAL_ADDRESS_PROPERTY, pour spécifier l'interface réseau à utiliser pour une connexion.

Des erreurs de connexion peuvent se produire si vous limitez la plage de ports. Si une erreur se produit, une exception MQException est émise avec le code anomalie IBM MQ MQRC_Q_MGR_NOT_AVAILABLE et le message suivant:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Une erreur peut se produire si tous les ports de la plage spécifiée sont utilisés ou si l'adresse IP, le nom d'hôte ou le numéro de port spécifié n'est pas valide (un numéro de port négatif, par exemple).

Accès aux files d'attente, aux rubriques et aux processus dans IBM MQ classes for Java

Pour accéder aux files d'attente, aux rubriques et aux processus, utilisez les méthodes de la classe MQQueueManager . Le MQOD (structure de descripteur d'objet) est réduit dans les paramètres de ces méthodes.

Files d'attente

Pour ouvrir une file d'attente, vous pouvez utiliser la méthode accessQueue de la classe MQQueueManager . Par exemple, sur un gestionnaire de files d'attente appelé queueManager, utilisez le code suivant:

```
MQQueue queue = queueManager.accessQueue("qName",CMQC.MQOO_OUTPUT);
```

La méthode accessQueue renvoie un nouvel objet de la classe MQQueue.

Une fois que vous avez fini d'utiliser la file d'attente, utilisez la méthode close () pour la fermer, comme dans l'exemple suivant:

```
queue.close();
```

Vous pouvez également créer une file d'attente à l'aide du constructeur MQQueue. Les paramètres sont exactement les mêmes que pour la méthode accessQueue , avec l'ajout d'un paramètre de gestionnaire de files d'attente. Exemple :

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

Vous pouvez spécifier un certain nombre d'options lorsque vous créez des files d'attente. Pour plus de détails, voir Class.com.ibm.mq.MQQueue. La construction d'un objet de file d'attente de cette manière vous permet d'écrire vos propres sous-classes de MQQueue.

Rubriques

De même, vous pouvez ouvrir une rubrique à l'aide de la méthode accessTopic de la classe MQQueueManager . Par exemple, sur un gestionnaire de files d'attente appelé queueManager, utilisez le code suivant pour créer un abonné et un diffuseur de publications:

```
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);
```

Une fois que vous avez fini d'utiliser la rubrique, utilisez la méthode close () pour la fermer.

Vous pouvez également créer une rubrique à l'aide du constructeur MQTopic. Les paramètres sont exactement les mêmes que pour la méthode accessTopic , avec l'ajout d'un paramètre de gestionnaire de files d'attente. Exemple :

```
MQTopic subscriber = new
      MQTopic(queueManager,"TOPICSTRING","TOPICNAME",
      CMQC.MQTOPIC_OPEN_AS_SUBSCRIPTION, CMQC.MQSO_CREATE);
```

Vous pouvez spécifier un certain nombre d'options lorsque vous créez des rubriques. Pour plus de détails, voir Class com.ibm.mq.MQTopic. La construction d'un objet de rubrique de cette manière vous permet d'écrire vos propres sous-classes de MQTopic.

Une rubrique doit être ouverte pour publication ou pour abonnement. La classe MQQueueManager comporte huit méthodes accessTopic et la classe Topic comporte huit constructeurs. Dans chaque cas, quatre d'entre eux ont un paramètre **destination** et quatre ont un paramètre **subscriptionName** (dont deux ont les deux). Ils ne peuvent être utilisés que pour ouvrir la rubrique pour les abonnements. Les deux méthodes restantes ont un paramètre **openAs** et la rubrique peut être ouverte pour la publication ou l'abonnement en fonction de la valeur du paramètre **openAs** .

Pour créer une rubrique en tant qu'abonné durable, utilisez une méthode accessTopic de la classe MQQueueManager ou un constructeur MQTopic qui accepte un nom d'abonnement et, dans les deux cas, définissez le CMQC CMQC.MQSO_DURABLE .

Processus

Pour accéder à un processus, utilisez la méthode accessProcess de MQQueueManager. Par exemple, sur un gestionnaire de files d'attente appelé queueManager, utilisez le code suivant pour créer un objet MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Pour accéder à un processus, utilisez la méthode accessProcess de MQQueueManager.

La méthode accessProcess renvoie un nouvel objet de la classe MQProcess.

Une fois que vous avez fini d'utiliser l'objet de processus, utilisez la méthode close () pour le fermer, comme dans l'exemple suivant:

```
process.close();
```

Vous pouvez également créer un processus à l'aide du constructeur MQProcess. Les paramètres sont exactement les mêmes que pour la méthode accessProcess , avec l'ajout d'un paramètre de gestionnaire de files d'attente. Exemple :

```
MQProcess process =
      new MQProcess(queueManager,"PROCESSNAME",
      CMQC.MQOO_FAIL_IF_QUIESCING);
```

La construction d'un objet de processus de cette manière vous permet d'écrire vos propres sous-classes de MQProcess.

Traitement des messages dans IBM MQ classes for Java

Les messages sont représentés par la classe MQMessage. Vous pouvez insérer et extraire des messages à l'aide des méthodes de la classe MQDestination, qui comporte des sous-classes de MQQueue et MQTopic.

Insertion de messages dans des files d'attente ou des rubriques à l'aide de la méthode put () de la classe MQDestination. Vous obtenez les messages des files d'attente ou des rubriques à l'aide de la méthode get () de la classe MQDestination. Contrairement à l'interface de procédure, dans laquelle MQPUT et MQGET placent et obtiennent des tableaux d'octets, le langage de programmation Java insère et extrait des instances de la classe MQMessage. La classe MQMessage encapsule la mémoire tampon de données qui contient les données de message réelles, ainsi que tous les paramètres MQMD (descripteur de message) et les propriétés de message qui décrivent ce message.

Pour générer un nouveau message, créez une nouvelle instance de la classe MQMessage et utilisez les méthodes writeXXX pour insérer des données dans la mémoire tampon de messages.

Lorsque la nouvelle instance de message est créée, tous les paramètres MQMD sont automatiquement définis sur leurs valeurs par défaut, comme défini dans Valeurs initiales et déclarations de langage pour MQMD. La méthode put () de MQDestination prend également une instance de la classe d'options MQPutMessagecomme paramètre. Cette classe représente la structure MQPMO. L'exemple suivant crée un message et le place dans une file d'attente:

```
// 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);
```

La méthode get () de MQDestination renvoie une nouvelle instance de MQMessage, qui représente le message provenant de la file d'attente. Il prend également une instance de la classe MQGetMessageOptions comme paramètre. Cette classe représente la structure MQGMO.

Vous n'avez pas besoin de spécifier une taille de message maximale, car la méthode get () ajuste automatiquement la taille de sa mémoire tampon interne pour qu'elle corresponde au message entrant. Utilisez les méthodes readXXX de la classe MQMessage pour accéder aux données du message renvoyé.

L'exemple suivant montre comment extraire un message d'une file d'attente:

```
// 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);
```

Vous pouvez modifier le format numérique utilisé par les méthodes de lecture et d'écriture en définissant la variable de membre *encoding* .

Vous pouvez modifier le jeu de caractères à utiliser pour la lecture et l'écriture de chaînes en définissant la variable de membre *characterSet* .

Pour plus d'informations, voir Classe MQMessage .

Remarque : La méthode writeUTF() de MQMessage code automatiquement la longueur de la chaîne ainsi que les octets Unicode qu'elle contient. Lorsque votre message sera lu par un autre programme Java (à l'aide de readUTF()), il s'agit du moyen le plus simple d'envoyer des informations de chaîne.

Amélioration des performances des messages non persistants dans IBM MQ classes for Java
Pour améliorer les performances lors de l'exploration de messages ou de la consommation de messages non persistants à partir d'une application client, vous pouvez utiliser la *lecture anticipée*. Les applications client utilisant MQGET ou la consommation asynchrone bénéficieront des améliorations de performances lors de la navigation dans les messages ou de la consommation de messages non persistants.

Pour des informations générales sur la fonction de lecture anticipée, voir la rubrique connexe.

Dans IBM MQ classes for Java, vous utilisez le CMQC CMQC.MQSO_READ_AHEAD et CMQC.MQSO_NO_READ_AHEAD d'un objet MQQueue ou MQTopic pour déterminer si les consommateurs de message et les navigateurs de file d'attente sont autorisés à utiliser la lecture anticipée sur cet objet.

Insertion de messages de manière asynchrone à l'aide de IBM MQ classes for Java
Pour insérer un message de manière asynchrone, définissez MQPMO_ASYNC_RESPONSE.

Vous placez des messages dans des files d'attente ou des rubriques à l'aide de la méthode put () de la classe MQDestination. Pour insérer un message de manière asynchrone, c'est-à-dire pour permettre à l'opération de se terminer sans attendre de réponse du gestionnaire de files d'attente, vous pouvez définir MQPMO_ASYNC_RESPONSE dans la zone d'options des options MQPutMessage. Pour déterminer la réussite ou l'échec des insertions asynchrones, utilisez l'appel de statut MQQueueManager.getAsync.

Publication / abonnement dans IBM MQ classes for Java
Dans IBM MQ classes for Java, la rubrique est représentée par la classe MQTopic et vous y publiez à l'aide des méthodes MQTopic.put().

Pour obtenir des informations générales sur la publication / l'abonnement IBM MQ , voir Messagerie de publication / abonnement.

Traitement des en-têtes de message IBM MQ avec IBM MQ classes for Java
Des classes Java sont fournies pour représenter différents types d'en-tête de message. Deux classes auxiliaires sont également fournies.

Interface MQHeader

Les objets d'en-tête sont décrits par l'interface MQHeader, qui fournit des méthodes générales permettant d'accéder aux zones d'en-tête et de lire et d'écrire du contenu de message. Chaque type d'en-tête possède sa propre classe qui implémente l'interface MQHeader et ajoute des méthodes d'accès get et set pour des zones individuelles. Par exemple, le type d'en-tête MQRFH2 est représenté par la classe MQRFH2 , le type d'en-tête MQDLH par la classe MQDLH, etc. Les classes d'en-tête effectuent automatiquement toute conversion de données nécessaire et peuvent lire ou écrire des données dans n'importe quel codage numérique ou jeu de caractères (CCSID) spécifié.

Important : Les classes d'en-têtes MQRFH2 traitent le message comme un fichier d'accès aléatoire, ce qui signifie que le curseur doit être positionné au début du message. Avant d'utiliser une classe d'en-tête de message interne telle que MQRFH, MQRFH2, MQCIH, MQDEAD, MQIIH ou MQXMIT, veillez à mettre à jour la position du curseur du message à l'emplacement approprié avant de transmettre le message à la classe.

Classes auxiliaires

Deux classes auxiliaires, MQHeaderIterator et MQHeaderList, facilitent la lecture et le décodage (analyse syntaxique) du contenu d'en-tête dans les messages:

- La classe MQHeaderIterator fonctionne comme un java.util.Iterator. Tant qu'il y a plus d'en-têtes dans le message, la méthode next () renvoie true et la méthode nextHeader() ou next () renvoie l'objet d'en-tête suivant.
- MQHeaderList fonctionne comme java.util.List. Tout comme MQHeaderIterator, il analyse le contenu de l'en-tête, mais il vous permet également de rechercher des en-têtes particuliers, d'ajouter de nouveaux en-têtes, de supprimer des en-têtes existants, de mettre à jour des zones d'en-tête, puis d'écrire le contenu de l'en-tête dans un message. Vous pouvez également créer une MQHeaderListvide, puis la remplir avec des instances d'en-tête et l'écrire dans un message une ou plusieurs fois.

Les classes MQHeaderIterator et MQHeaderList utilisent les informations du registre MQHeaderRegistry pour savoir quelles classes d'en-tête IBM MQ sont associées à des types et des formats de message particuliers. MQHeaderRegistry est configuré avec la connaissance de tous les formats et types d'en-tête IBM MQ en cours et de leurs classes d'implémentation, et vous pouvez également enregistrer vos propres types d'en-tête.

La prise en charge est fournie pour les en-têtes IBM MQ couramment utilisés suivants

- MQRFH-Règles et en-tête de formatage

- MQRFH2 -Comme MQRFH, utilisé pour transmettre des messages vers et depuis un courtier de messages appartenant à IBM Integration Bus. Egalement utilisé pour contenir les propriétés de message
- MQCIH-Pont CICS
- MQDLH-En-tête de rebut
- MQIIH - En-tête d'informations IMS
- MQRMH-en-tête de message de référence
- MQSAPH-en-tête SAP
- MQWIH-En-tête d'informations de travail
- MQXQH-en-tête de file d'attente de transmission
- En-tête MQDH-Distribution
- MQEPH-En-tête PCF Encapsulé

Vous pouvez également définir des classes représentant vos propres en-têtes.

Pour utiliser un MQHeaderIterator afin d'obtenir un en-tête RFH2 , définissez MQGMO_PROPERTIES_FORCE_MQRFH2 dans les options GetMessageou définissez la propriété de file d'attente PROPCTL sur FORCE.

Impression de tous les en-têtes d'un message à l'aide de IBM MQ classes for Java
Dans cet exemple, une instance de MQHeaderIterator analyse les en-têtes d'un MQMessage qui a été reçu d'une file d'attente. Les objets MQHeader renvoyés par la méthode nextHeader() affichent leur structure et leur contenu lorsque leur méthode toString est appelée.

```
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);
}
```

Ignorer les en-têtes d'un message à l'aide de IBM MQ classes for Java
Dans cet exemple, la méthode skipHeaders() de MQHeaderIterator positionne le curseur de lecture du message immédiatement après le dernier en-tête.

```
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 ();
```

Recherche du code anomalie dans un message au rebut à l'aide de IBM MQ classes for Java
Dans cet exemple, la méthode read remplit l'objet MQDLH en lisant le message. Après l'opération de lecture, le curseur de lecture de message est positionné immédiatement après le contenu de l'en-tête MQDLH.

Les messages de la file d'attente de rebut du gestionnaire de files d'attente sont précédés d'un en-tête de rebut (MQDLH). Pour décider comment traiter ces messages-par exemple, pour déterminer s'ils doivent être réessayés ou supprimés-une application de traitement des messages non livrés doit examiner le code anomalie contenu dans le 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 ());
```

Toutes les classes d'en-tête fournissent également un constructeur pratique pour s'initialiser directement à partir du message en une seule étape. Le code de cet exemple peut donc être simplifié comme suit:

```
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 ());
```

Lecture et suppression de l'en-tête d'un message de rebut à l'aide de IBM MQ classes for Java
Dans cet exemple, MQDLH est utilisé pour supprimer l'en-tête d'un message de rebut.

Une application de traitement des messages non livrés soumet généralement à nouveau les messages qui ont été rejetés si leur code anomalie indique une erreur transitoire. Avant de soumettre à nouveau le message, il doit supprimer l'en-tête MQDLH.

Cet exemple effectue les étapes suivantes (voir les commentaires dans l'exemple de code):

1. MQHeaderList lit l'intégralité du message et chaque en-tête rencontré dans le message devient un élément de la liste.
2. Les messages non livrés contiennent un MQDLH comme premier en-tête, ce qui se trouve dans le premier élément de la liste d'en-têtes. Le MQDLH a déjà été renseigné à partir du message lors de la génération de MQHeaderList . Il n'est donc pas nécessaire d'appeler sa méthode de lecture.
3. Le code anomalie est extrait à l'aide de la méthode getReason() fournie par la classe MQDLH.
4. Le code raison a été inspecté et indique qu'il est approprié de soumettre à nouveau le message. Le MQDLH est supprimé à l'aide de la méthode MQHeaderList remove ().
5. MQHeaderList écrit son contenu restant dans un nouvel objet de message. Le nouveau message contient désormais tout ce qui se trouve dans le message d'origine, à l'exception du MQDLH, et peut être écrit dans une file d'attente. L'argument **true** du constructeur et de la méthode d'écriture indique que le corps du message doit être conservé dans MQHeaderList, puis réécrit.
6. La zone de format du descripteur de message du nouveau message contient maintenant la valeur qui était précédemment dans la zone de format MQDLH. Les données de message correspondent au codage numérique et au CCSID définis dans le descripteur de message.

```
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.
```

Impression du contenu d'un message à l'aide de IBM MQ classes for Java
Cet exemple utilise MQHeaderList pour imprimer le contenu d'un message, y compris ses en-têtes.

La sortie contient une vue de tout le contenu de l'en-tête ainsi que le corps du message. La classe MQHeaderList décode tous les en-têtes en une seule fois, tandis que le MQHeaderIterator les décode un par un sous le contrôle de l'application. Vous pouvez utiliser cette technique pour fournir un outil de débogage simple lors de l'écriture d'applications 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));
```

Cet exemple imprime également les zones de descripteur de message à l'aide de la classe MQMD. La méthode copyFrom() de la classe com.ibm.mq.headers.MQMD remplit l'objet d'en-tête à partir des zones de descripteur de message du MQMessage plutôt qu'en lisant le corps du message.

```
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));
```

Recherche d'un type spécifique d'en-tête dans un message à l'aide de IBM MQ classes for Java
Cet exemple utilise la méthode indexOf(String) de MQHeaderList pour rechercher un en-tête MQRFH2 dans un message, le cas échéant.

```
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);
        ...
}
```

Analyse d'un en-tête MQRFH2 à l'aide de IBM MQ classes for Java
Cet exemple montre comment accéder à une valeur de zone connue dans un dossier nommé, à l'aide de la classe MQRFH2 .

La classe MQRFH2 offre un certain nombre de moyens d'accéder non seulement aux zones de la partie fixe de la structure, mais également au contenu de dossier codé XML contenu dans la zone de données NameValue. Cet exemple montre comment accéder à une valeur de zone connue dans un dossier nommé- dans ce cas, la zone Rto dans le dossier jms, qui représente le nom de la file d'attente de réponses dans un message MQ JMS .

```
MQRFH2 rfh = ...

String value = rfh.getStringFieldValue ("jms", "Rto");
```

Pour reconnaître le contenu d'un MQRFH2 (au lieu de demander directement des zones spécifiques), vous pouvez utiliser la méthode getFolders pour renvoyer une liste de MQRFH2.Element, qui représente la structure d'un dossier pouvant contenir des zones et d'autres dossiers. La définition de la valeur

null pour une zone ou un dossier la supprime de MQRFH2. Lorsque vous manipulez le contenu du dossier de données NameValuede cette manière, la zone StrucLength est automatiquement mise à jour en conséquence.

Lecture et écriture de flux d'octets autres que des objets MQMessage à l'aide de IBM MQ classes for Java
Ces exemples utilisent les classes d'en-tête pour analyser et manipuler le contenu de l'en-tête IBM MQ lorsque la source de données n'est pas un objet MQMessage.

Vous pouvez utiliser les classes d'en-tête pour analyser et manipuler le contenu de l'en-tête IBM MQ même lorsque la source de données est autre chose qu'un objet MQMessage. L'interface MQHeader implémentée par chaque classe d'en-tête fournit les méthodes `int read (java.io.DataInput message, int encoding, int characterSet)` et `int write (java.io.DataOutput message, int encoding, int characterSet)`. La classe com.ibm.mq.MQMessage implémente les interfaces java.io.DataInput et java.io.DataOutput . Cela signifie que vous pouvez utiliser les deux méthodes MQHeader pour lire et écrire du contenu MQMessage, en remplaçant le codage et le CCSID spécifiés dans le descripteur de message. Ceci est utile pour les messages qui contiennent une chaîne d'en-têtes dans différents codages.

Vous pouvez également obtenir des objets DataInput et DataOutput à partir d'autres flux de données, par exemple des flux de fichiers ou de sockets, ou des tableaux d'octets transportés dans des messages JMS . Les classes java.io.DataInputStream implémentent DataInput et les classes java.io.DataOutputStream implémentent DataOutput. Cet exemple lit le contenu d'en-tête IBM MQ à partir d'un tableau d'octets:

```
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 ligne commençant par MQHeaderIterator peut être remplacée par

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

Cet exemple écrit dans un tableau d'octets à l'aide d'un flux DataOutput:

```
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 ();
```

Lorsque vous utilisez les flux de cette manière, veillez à utiliser les valeurs correctes pour le codage et les arguments characterSet . Lors de la lecture des en-têtes, indiquez le codage et le CCSID avec lesquels le contenu en octets a été écrit à l'origine. Lors de l'écriture d'en-têtes, indiquez le codage et le CCSID que vous souhaitez générer. La conversion des données est effectuée automatiquement par les classes d'en-tête.

Création de classes pour les nouveaux types d'en-tête à l'aide de IBM MQ classes for Java
Vous pouvez créer des classes Java pour les types d'en-tête non fournis avec IBM MQ classes for Java.

Pour ajouter une classe Java représentant un nouveau type d'en-tête que vous pouvez utiliser de la même manière que n'importe quelle classe d'en-tête fournie avec IBM MQ classes for Java, créez une classe qui implémente l'interface MQHeader. L'approche la plus simple consiste à étendre la classe com.ibm.mq.headers.impl.Header . Cet exemple génère une classe entièrement fonctionnelle représentant la structure d'en-tête MQTM. Vous n'avez pas besoin d'ajouter des méthodes d'accès get et set individuelles pour chaque zone, mais cela est utile pour les utilisateurs de la classe d'en-tête. Les méthodes génériques getValue et setValue qui utilisent une chaîne pour le nom de zone fonctionnent pour toutes les zones définies dans le type d'en-tête. Les méthodes de lecture, d'écriture et de taille

héritées permettront aux instances du nouveau type d'en-tête d'être lues et écrites et calculeront la taille d'en-tête correctement en fonction de sa définition de zone. La définition de type est créée une seule fois, mais de nombreuses instances de cette classe d'en-tête sont créées. Pour rendre la nouvelle définition d'en-tête disponible pour le décodage à l'aide des classes MQHeaderIterator ou MQHeaderList , vous devez l'enregistrer à l'aide de MQHeaderRegistry. Notez toutefois que la classe d'en-tête MQTM est en fait déjà fournie dans ce package et enregistrée dans le registre par défaut.

```
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.
}
```

Traitement des messages PCF avec IBM MQ classes for Java

Les classes Java sont fournies pour créer et analyser des messages structurés PCF et pour faciliter l'envoi de demandes PCF et la collecte de réponses PCF.

Les classes PCFMessage et MQCFGR représentent des tableaux de structures de paramètres PCF. Ils fournissent des méthodes pratiques pour ajouter et extraire des paramètres PCF.

Les structures de paramètres PCF sont représentées par les classes MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL et MQCFGR. Ces interfaces partagent des interfaces opérationnelles de base:

- Méthodes de lecture et d'écriture de contenu de message: read (), write () et size ()
- Méthodes de manipulation des paramètres: getValue (), setValue (), getParameter () et autres
- Méthode d'énumération.nextParameter (), qui analyse le contenu PCF dans un MQMessage

Le paramètre de filtre PCF est utilisé dans les commandes d'interrogation pour fournir une fonction de filtrage. Il est encapsulé dans les classes suivantes:

- MQCFIF-filtre de type entier
- MQCFSF-filtre de chaîne
- MQCFBF-filtre d'octets

Deux classes d'agent, PCFAgent et PCFMessageAgent , sont fournies pour gérer la connexion à un gestionnaire de files d'attente, à la file d'attente du serveur de commandes et à une file d'attente de réponses associée. PCFMessageAgent étend PCFAgent et doit normalement être utilisé de préférence. La

classe PCFMessageAgent convertit les messages MQMessage reçus et les transmet à l'appelant sous la forme d'un tableau PCFMessage. PCFAgent renvoie un tableau de MQMessages que vous devez analyser avant d'utiliser.

Traitement des propriétés de message dans IBM MQ classes for Java

Les appels de fonction pour traiter les descripteurs de message n'ont pas d'équivalent dans IBM MQ classes for Java. Pour définir, renvoyer ou supprimer des propriétés de descripteur de message, utilisez les méthodes de la classe MQMessage.

Pour des informations générales sur les propriétés de message, voir «Noms de propriétés», à la page 29.

Dans IBM MQ classes for Java , l'accès aux messages se fait via la classe MQMessage. Les descripteurs de message ne sont donc pas fournis dans l'environnement Java et il n'existe pas d'équivalent aux appels de fonction IBM MQ MQCRTMH, MQDLTMH, MQMHBUF et MQBUFMH

Pour définir les propriétés de descripteur de message dans l'interface de procédure, utilisez l'appel MQSETMP. Dans IBM MQ classes for Java, utilisez la méthode appropriée de la classe MQMessage:

- Propriété setBoolean
- Propriété setByte
- Propriété setBytes
- Propriété setShort
- Propriété setInt
- setInt2Property
- setInt4Property
- setInt8Property
- Propriété setLong
- Propriété setFloat
- Propriété setDouble
- Propriété setString
- Propriété setObject

Ces méthodes sont parfois appelées collectivement méthodes *set*property* .

Pour renvoyer la valeur des propriétés de descripteur de message dans l'interface de procédure, utilisez l'appel MQINQMP. Dans IBM MQ classes for Java, utilisez la méthode appropriée de la classe MQMessage:

- Propriété getBoolean
- Propriété getByte
- Propriété getBytes
- Propriété getShort
- Propriété getInt
- getInt2Property
- getInt4Property
- getInt8Property
- Propriété getLong
- Propriété getFloat
- Propriété getDouble
- Propriété getString
- Propriété getObject

Ces méthodes sont parfois appelées collectivement méthodes *get*property* .

Pour supprimer la valeur des propriétés de descripteur de message dans l'interface de procédure, utilisez l'appel MQDLTMP. Dans IBM MQ classes for Java, utilisez la méthode deleteProperty de la classe MQMessage.

Traitement des erreurs dans IBM MQ classes for Java

Gérez les erreurs provenant de IBM MQ classes for Java à l'aide de blocs Java `try` et `catch` .

Les méthodes de l'interface Java ne renvoient pas de code achèvement ni de code anomalie. Au lieu de cela, ils émettent une exception chaque fois que le code achèvement et le code anomalie résultant d'un appel IBM MQ ne sont pas tous les deux nuls. Cela simplifie la logique du programme de sorte que vous n'ayez pas à vérifier les codes retour après chaque appel à IBM MQ. Vous pouvez décider à quels points de votre programme vous souhaitez faire face à la possibilité d'un échec. A ces points, vous pouvez entourer votre code de blocs `try` et `catch` , comme dans l'exemple suivant:

```
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() );
}
```

Les codes raison d'appel IBM MQ renvoyés dans les exceptions Java pour z/OS sont documentés dans Codes achèvement d'API et codes raison.

Les exceptions émises lors de l'exécution d'une application IBM MQ classes for Java sont également consignées dans le journal. Toutefois, une application peut appeler la méthode MQException.logExclude() pour empêcher la consignation des exceptions associées à un code anomalie spécifique. Vous pouvez être amené à effectuer cette opération dans les situations où vous vous attendez à ce que de nombreuses exceptions associées à un code anomalie spécifique soient émises et que vous ne souhaitez pas que le journal soit rempli avec ces exceptions. Par exemple, si votre application tente d'extraire un message d'une file d'attente chaque fois qu'elle effectue une itération autour d'une boucle et que, pour la plupart de ces tentatives, vous vous attendez à ce qu'aucun message approprié ne soit dans la file d'attente, vous pouvez empêcher la consignation d'exceptions associées au code anomalie MQRC_NO_MSG_AVAILABLE. Si une application a précédemment empêché la consignation des exceptions associées à un code anomalie spécifique, elle peut autoriser la consignation de ces exceptions à nouveau en appelant la méthode MQException.logInclude().

Parfois, le code raison ne transmet pas tous les détails associés à l'erreur. Pour chaque exception émise, une application doit vérifier l'exception liée. L'exception associée proprement dite peut comporter une autre exception associée ; les exceptions associées forment donc une chaîne renvoyant à l'incident sous-jacent d'origine. Une exception associée est implémentée à l'aide du mécanisme d'exceptions chaînées de la classe java.lang.Throwable et une application obtient une exception associée en appelant la méthode Throwable.getCause(). A partir d'une exception qui est une instance de MQException, MQException.getCause() extrait l'instance sous-jacente de com.ibm.mq.jmqi.JmqiException, et getCause de cette exception extrait l'exception java.lang.Exception sous-jacente qui a causé l'erreur.

Obtention et définition de valeurs d'attribut dans IBM MQ classes for Java

Les méthodes getXXX() et setXXX() sont fournies pour de nombreux attributs communs. D'autres sont accessibles à l'aide des méthodes génériques inquire () et set ().

Pour la plupart des attributs communs, les classes MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess et MQQueueManager contiennent les méthodes getXXX() et setXXX(). Ces méthodes vous permettent d'obtenir et de définir leurs valeurs d'attribut. Notez que pour MQDestination, MQQueue et MQTopic, les méthodes ne fonctionnent que si vous spécifiez les indicateurs d'interrogation et de définition appropriés lorsque vous ouvrez l'objet.

Pour les attributs moins courants, les classes MQQueueManager, MQDestination, MQQueue, MQTopic et MQProcess héritent toutes d'une classe appelée MQManagedObject. Cette classe définit les interfaces inquire () et set ().

Lorsque vous créez un objet de gestionnaire de files d'attente à l'aide de l'opérateur *new* , il est automatiquement ouvert pour l'interrogation. Lorsque vous utilisez la méthode accessProcess() pour accéder à un objet de processus, cet objet est automatiquement ouvert pour l'interrogation. Lorsque vous utilisez la méthode accessQueue() pour accéder à un objet file d'attente, cet objet n'est pas ouvert automatiquement pour les opérations d'interrogation ou de définition. En effet, l'ajout automatique de ces options peut entraîner des problèmes avec certains types de files d'attente éloignées. Pour utiliser les méthodes inquire, set, getXXXet setXXX sur une file d'attente, vous devez spécifier les indicateurs inquire et set appropriés dans le paramètre openOptions de la méthode accessQueue(). Il en est de même pour les objets de destination et de rubrique.

Les méthodes d'interrogation et de définition prennent trois paramètres:

- tableau de sélecteurs
- Tableau intAttrs
- Tableau charAttrs

Vous n'avez pas besoin des paramètres SelectorCount, IntAttrCount et CharAttrLength trouvés dans MQINQ, car la longueur d'un tableau dans Java est toujours connue. L'exemple suivant montre comment effectuer une interrogation sur une file d'attente:

```
// 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));
```

Programmes à unités d'exécution multiples dans Java

L'environnement d'exécution Java est intrinsèquement à unités d'exécution multiples. IBM MQ classes for Java permet à un objet de gestionnaire de files d'attente d'être partagé par plusieurs unités d'exécution, mais garantit que tous les accès au gestionnaire de files d'attente cible sont synchronisés.

Les programmes à unités d'exécution multiples sont difficiles à éviter dans Java. Prenons l'exemple d'un programme simple qui se connecte à un gestionnaire de files d'attente et ouvre une file d'attente au démarrage. Le programme affiche un seul bouton à l'écran. Lorsqu'un utilisateur clique sur ce bouton, le programme extrait un message de la file d'attente.

L'environnement d'exécution Java est intrinsèquement à unités d'exécution multiples. Par conséquent, l'initialisation de votre application se produit dans une unité d'exécution et le code qui s'exécute en réponse à la pression du bouton s'exécute dans une unité d'exécution distincte (l'unité d'exécution de l'interface utilisateur).

Avec le IBM MQ MQI clientbasé sur C, cela provoquerait un problème, car il existe des limitations au partage des descripteurs par plusieurs unités d'exécution. IBM MQ classes for Java assouplit cette contrainte, permettant à un objet de gestionnaire de files d'attente (et à ses objets de file d'attente, de rubrique et de processus associés) d'être partagé par plusieurs unités d'exécution.

L'implémentation de IBM MQ classes for Java garantit que, pour une connexion particulière (instance d'objetMQQueueManager), tous les accès au gestionnaire de files d'attente IBM MQ cible sont synchronisés. Une unité d'exécution qui souhaite émettre un appel à un gestionnaire de files d'attente

est bloquée jusqu'à ce que tous les autres appels en cours pour cette connexion soient terminés. Si vous avez besoin d'un accès simultané au même gestionnaire de files d'attente à partir de plusieurs unités d'exécution de votre programme, créez un nouvel objet MQQueueManager pour chaque unité d'exécution nécessitant un accès simultané. (Cela revient à émettre un appel MQCONN distinct pour chaque unité d'exécution.)

Remarque : Les instances de la classe `com.ibm.mq.MQGetMessageOptions` ne doivent pas être partagées entre les unités d'exécution qui demandent des messages simultanément. Les instances de cette classe sont mises à jour avec les données lors de la demande MQGET correspondante, ce qui peut entraîner des conséquences inattendues lorsque plusieurs unités d'exécution s'exécutent simultanément sur la même instance de l'objet.

Utilisation des exits de canal dans IBM MQ classes for Java

Présentation de l'utilisation des exits de canal dans une application à l'aide de IBM MQ classes for Java.

Les rubriques suivantes décrivent comment écrire un exit de canal dans Java, comment l'affecter et comment lui transmettre des données. Ils décrivent ensuite comment utiliser les exits de canal écrits en C et comment utiliser une séquence d'exits de canal.

Votre application doit disposer des droits de sécurité appropriés pour charger la classe d'exit de canal.

Création d'un exit de canal dans IBM MQ classes for Java

Vous pouvez fournir vos propres exits de canal en définissant une classe Java qui implémente une interface appropriée.

Pour implémenter un exit, vous devez définir une nouvelle classe Java qui implémente l'interface appropriée. Trois interfaces d'exit sont définies dans le package com.ibm.mq.exits :

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Remarque : Les exits de canal sont pris en charge pour les connexions client uniquement ; ils ne sont pas pris en charge pour les connexions de liaisons. Vous ne pouvez pas utiliser un exit de canal Java en dehors de IBM MQ classes for Java, par exemple si vous utilisez une application client écrite en C.

Tout chiffrement TLS défini pour une connexion est effectué *après l'appel des exits d'envoi et de sécurité* . De même, le déchiffrement est effectué *avant* l'appel des exits de réception et de sécurité.

L'exemple suivant définit une classe qui implémente les trois 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
```

```
  }
}
```

Chaque exit reçoit un objet MQCXP et un objet MQCD. Ces objets représentent les structures MQCXP et MQCD définies dans l'interface de procédure.

Toute classe d'exit que vous écrivez doit avoir un constructeur. Il peut s'agir du constructeur par défaut ou d'un constructeur qui prend un argument de chaîne. S'il utilise une chaîne, les données utilisateur sont transmises à la classe d'exit lors de leur création. Si la classe d'exit contient à la fois un constructeur par défaut et un constructeur à un seul argument, le constructeur à un seul argument a la priorité.

Pour les exits d'envoi et de sécurité, votre code d'exit doit renvoyer les données que vous souhaitez envoyer au serveur. Pour un exit de réception, votre code d'exit doit renvoyer les données modifiées que IBM MQ doit interpréter.

Le corps de sortie le plus simple possible est:

```
{ return agentBuffer; }
```

Ne fermez pas le gestionnaire de files d'attente à partir d'un exit de canal.

Utilisation de classes d'exit de canal existantes

Dans les versions de IBM MQ antérieures à 7.0, vous implémentez ces exits à l'aide des interfaces MQSendExit, MQReceiveExitet MQSecurityExit, comme dans l'exemple suivant. Cette méthode reste valide, mais la nouvelle méthode est préférable pour améliorer les fonctionnalités et les performances.

```
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
  }
}
```

Affectation d'un exit de canal dans IBM MQ classes for Java
Vous pouvez affecter un exit de canal à l'aide de IBM MQ classes for Java.

Il n'existe pas d'équivalent direct du canal IBM MQ dans IBM MQ classes for Java. Les exits de canal sont affectés à un MQQueueManager. Par exemple, après avoir défini une classe qui implémente l'interface WMQSecurityExit , une application peut utiliser l'exit de sécurité de l'une des quatre manières suivantes:

- En affectant une instance de la classe à la zone MQEnvironment.channelSecurityExit avant de créer un objet MQQueueManager
- En définissant la zone MQEnvironment.channelSecurityExit sur une chaîne représentant la classe d'exit de sécurité avant de créer un objet MQQueueManager

- En créant une paire clé / valeur dans la table de hachage des propriétés transmise à MQQueueManager avec la clé CMQC.SECURITY_EXIT_PROPERTY
- Utilisation d'une table de définition de canal du client (CCDT)

Tout exit affecté en définissant la zone MQEnvironment.channelSecurityExit sur une chaîne, en créant une paire clé / valeur dans la table de hachage des propriétés ou en utilisant une table de définition de canal du client, doit être écrit avec un constructeur par défaut. Un exit affecté en tant qu'instance d'une classe n'a pas besoin d'un constructeur par défaut, en fonction de l'application.

Une application peut utiliser un exit d'émission ou de réception de la même manière. Par exemple, le fragment de code suivant vous montre comment utiliser les exits de sécurité, d'envoi et de réception qui sont implémentés dans la classe MyMQExits, qui a été définie précédemment, à l'aide de MQEnvironment:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.channelSecurityExit = myexits;
MQEnvironment.channelSendExit = myexits;
MQEnvironment.channelReceiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

Si plusieurs méthodes sont utilisées pour affecter un exit de canal, l'ordre de priorité est le suivant:

1. Si l'URL d'une table de définition de canal du client est transmise à MQQueueManager, le contenu de la table de définition de canal du client détermine les exits de canal à utiliser et les définitions d'exit dans MQEnvironment ou la table de hachage des propriétés sont ignorées.
2. Si aucune URL CCDT n'est transmise, les définitions d'exit de MQEnvironment et de la table de hachage sont fusionnées
   - Si le même type d'exit est défini à la fois dans MQEnvironment et dans la table de hachage, la définition de la table de hachage est utilisée.
   - Si des types d'exit équivalents anciens et nouveaux sont spécifiés (par exemple, la zone sendExit , qui ne peut être utilisée que pour le type d'exit utilisé dans les versions antérieures à IBM WebSphere MQ 7.0, et la zone d'exit channelSend, qui peut être utilisée pour n'importe quel exit d'émission), le nouvel exit (channelSendExit) est utilisé à la place de l'ancien exit.

Si vous avez déclaré un exit de canal sous forme de chaîne, vous devez activer IBM MQ pour localiser le programme d'exit de canal. Vous pouvez le faire de différentes manières, en fonction de l'environnement dans lequel l'application s'exécute et de la façon dont les programmes d'exit de canal sont conditionnés.

- Pour une application qui s'exécute dans un serveur d'applications, vous devez stocker les fichiers dans le répertoire indiqué dans Tableau 58, à la page 404 ou dans des fichiers JAR référencés par **exitClasspath**.
- Pour une application qui n'est pas exécutée dans un serveur d'applications, les règles suivantes s'appliquent:
  - Si vos classes d'exit de canal sont conditionnées dans des fichiers JAR distincts, ces fichiers JAR doivent être inclus dans le fichier **exitClasspath**.
  - Si vos classes d'exit de canal ne sont pas conditionnées dans des fichiers JAR, les fichiers de classe peuvent être stockés dans le répertoire indiqué dans Tableau 58, à la page 404 ou dans n'importe quel répertoire du chemin d'accès aux classes système de la machine virtuelle Java ou dans **exitClasspath**.

La propriété **exitClasspath** peut être spécifiée de quatre manières. Par ordre de priorité, ces méthodes sont les suivantes:

1. La propriété système com.ibm.mq.exitClasspath (définie sur la ligne de commande à l'aide de l'option `-D`)
2. La section exitPath du fichier `mqclient.ini`
3. Une entrée de table de hachage avec la clé CMQC.EXIT_CLASSPATH_PROPERTY EXIT_CLASSPATH_PROPERTY

4. La variable MQEnvironment **exitClasspath**

Séparez plusieurs chemins à l'aide du caractère java.io.File.pathSeparator .

Tableau 58. Répertoire des programmes d'exit de canal

Plateforme	Répertoire
AIX AIX Linux Linux	/var/mqm/exits (programmes d'exit de canal 32 bits) /var/mqm/exits64 (programmes d'exit de canal 64 bits)
Windows	*rép_données_installation*\exits

Remarque : *rép_install_data_dir* est le répertoire que vous avez choisi pour les fichiers de données IBM MQ lors de l'installation. Le répertoire par défaut est C:\ProgramData\IBM\MQ.

Transmission de données aux exits de canal dans IBM MQ classes for Java
Vous pouvez transmettre des données aux exits de canal et renvoyer des données des exits de canal à votre application.

Paramètre agentBuffer

Pour un exit d'émission, le paramètre *agentBuffer* contient les données qui sont sur le point d'être envoyées. Pour un exit de réception ou de sécurité, le paramètre *agentBuffer* contient les données qui viennent d'être reçues. Vous n'avez pas besoin de paramètre de longueur, car l'expression agentBuffer.limit () indique la longueur du tableau.

Pour les exits d'envoi et de sécurité, votre code d'exit doit renvoyer les données que vous souhaitez envoyer au serveur. Pour un exit de réception, votre code d'exit doit renvoyer les données modifiées que IBM MQ doit interpréter.

Le corps de sortie le plus simple possible est:

```
{ return agentBuffer; }
```

Les exits de canal sont appelés avec une mémoire tampon comportant un tableau de sauvegarde. Pour de meilleures performances, l'exit doit renvoyer une mémoire tampon avec un tableau de sauvegarde.

Données utilisateur

Si une application se connecte à un gestionnaire de files d'attente en définissant l'exit channelSecurity, l'exit channelSendou l'exit channelReceive, 32 octets de données utilisateur peuvent être transmis à la classe d'exit de canal appropriée lorsqu'elle est appelée, à l'aide des zones channelSecurityExitUserData, channelSendExitUserData ou channelReceiveExitUserData. Ces données utilisateur sont disponibles pour la classe d'exit de canal mais sont actualisées chaque fois que l'exit est appelé. Toute modification apportée aux données utilisateur dans l'exit de canal sera donc perdue. Si vous souhaitez apporter des modifications persistantes aux données dans un exit de canal, utilisez la zone exitUserde MQCXP. Les données de cette zone sont conservées entre les appels de l'exit.

Si l'application définit securityExit, sendExitou receiveExit, aucune donnée utilisateur ne peut être transmise à ces classes d'exit de canal.

Si une application utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, toutes les données utilisateur spécifiées dans une définition de canal de connexion client sont transmises aux classes d'exit de canal lorsqu'elles sont appelées. Pour plus d'informations sur l'utilisation d'une table de définition de canal du client, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for Java», à la page 386.

Utilisation d'exits de canal non écrits dans Java avec IBM MQ classes for Java
Comment utiliser des programmes d'exit de canal écrits en C à partir d'une application Java .

Dans IBM MQ, vous pouvez spécifier le nom d'un programme d'exit de canal écrit en C sous la forme d'une chaîne transmise à l'exit channelSecurity, à l'exit channelSendou aux zones d'exit channelReceivedans l'objet MQEnvironment ou la table de hachage des propriétés. Toutefois, vous ne pouvez pas utiliser un exit de canal écrit en Java dans une application écrite dans une autre langue.

Indiquez le nom du programme d'exit au format `library(function)` et vérifiez que l'emplacement du programme d'exit est indiqué comme indiqué dans Chemin d'accès aux exits.

Pour plus d'informations sur l'écriture d'un exit de canal en C, voir «Programmes d'exit de canal pour les canaux de messagerie», à la page 988.

Utilisation d'une séquence d'exits d'émission ou de réception de canal dans IBM MQ classes for Java
Une application IBM MQ classes for Java peut utiliser une séquence d'exits d'émission ou de réception de canal qui sont exécutés successivement.

Pour utiliser une séquence d'exits d'envoi, une application peut créer une liste ou une chaîne contenant les exits d'envoi. Si une liste est utilisée, chaque élément de la liste peut être l'un des suivants:

- Instance d'une classe définie par l'utilisateur qui implémente l'interface WMQSendExit
- Une instance d'une classe définie par l'utilisateur qui implémente l'interface MQSendExit (pour un exit d'émission écrit en Java)
- Une instance de la classe d'exit MQExternalSend(pour un exit d'émission non écrit dans Java)
- Une instance de la classe de chaîne MQSendExit
- Une instance de la classe String

Une liste ne peut pas contenir une autre liste.

L'application peut utiliser une séquence d'exits de réception de la même manière.

Si une chaîne est utilisée, elle doit être constituée d'une ou de plusieurs définitions d'exit séparées par des virgules, chacune pouvant être le nom d'une classe Java ou d'un programme C au format `library(function)`.

L'application affecte ensuite l'objet Liste ou Chaîne à la zone MQEnvironment.channelSendExit avant de créer un objet MQQueueManager .

Le contexte des informations transmises aux exits se trouve uniquement dans le domaine des exits. Par exemple, si un exit Java et un exit C sont chaînés, la présence de l'exit Java n'a aucun effet sur l'exit C.

Utilisation des classes de chaîne d'exit

Dans les versions antérieures à IBM WebSphere MQ 7.0, deux classes ont été fournies pour autoriser les séquences d'exits:

- Chaîne MQSendExit, qui implémente l'interface MQSendExit
- MQReceiveExitChaîne qui implémente l'interface MQReceiveExit

L'utilisation de ces classes reste valide, mais la nouvelle méthode est préférée. L'utilisation des interfaces IBM MQ Classes for Java signifie que votre application a toujours une dépendance sur `com.ibm.mq.jar` . Si le nouvel ensemble d'interfaces du package `com.ibm.mq.exits` est utilisé, il n'y a pas de dépendance sur `com.ibm.mq.jar`.

Pour utiliser une séquence d'exits d'envoi, une application a créé une liste d'objets, où chaque objet était l'un des suivants:

- Une instance d'une classe définie par l'utilisateur qui implémente l'interface MQSendExit (pour un exit d'émission écrit en Java)
- Une instance de la classe d'exit MQExternalSend(pour un exit d'émission non écrit dans Java)
- Une instance de la classe de chaîne MQSendExit

L'application a créé un objet de chaîne MQSendExiten transmettant cette liste d'objets en tant que paramètre sur le constructeur. L'application aurait ensuite affecté l'objet de chaîne MQSendExità la zone MQEnvironment.sendExit avant de créer un objet MQQueueManager .

Compression de canal dans IBM MQ classes for Java

La compression des données qui circulent sur un canal peut améliorer les performances du canal et réduire le trafic réseau. IBM MQ classes for Java utilise la fonction de compression intégrée à IBM MQ.

A l'aide de la fonction fournie avec IBM MQ, vous pouvez compresser les données qui circulent sur les canaux de transmission de messages et les canaux MQI et, sur l'un ou l'autre type de canal, vous pouvez compresser les données d'en-tête et les données de message indépendamment les unes des autres. Par défaut, aucune donnée n'est compressée sur un canal. Pour une description complète de la compression de canal, y compris de la manière dont elle est implémentée dans IBM MQ, voir Data compression (COMPMSG) et Header compression (COMPHDR).

Une application IBM MQ classes for Java spécifie les techniques qui peuvent être utilisées pour compresser les données d'en-tête ou de message sur une connexion client en créant un objet java.util.Collection . Chaque technique de compression est un objet Integer dans la collection, et l'ordre dans lequel l'application ajoute les techniques de compression à la collection est l'ordre dans lequel les techniques de compression sont négociées avec le gestionnaire de files d'attente lorsque la connexion client démarre. L'application peut ensuite affecter la collection à la zone de liste hdrComp, pour les données d'en-tête, ou à la zone de liste msgComp, pour les données de message, dans la classe MQEnvironment. Lorsque l'application est prête, elle peut démarrer la connexion client en créant un objet MQQueueManager .

Les fragments de code suivants illustrent l'approche décrite. Le premier fragment de code vous montre comment implémenter la compression des données d'en-tête:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(CMQXC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Le deuxième fragment de code vous montre comment implémenter la compression des données de message:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(CMQXC.MQCOMPRESS_RLE));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_ZLIBHIGH));
msgComp.add(new Integer(CMQXC.MQCOMPRESS_LZ4HIGH));
:
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

Dans le second example, les techniques de compression sont négociées dans l'ordre RLE, puis ZLIBHIGH, lorsque la connexion client démarre. La technique de compression sélectionnée ne peut pas être modifiée pendant la durée de vie de l'objet MQQueueManager .

Les techniques de compression des données d'en-tête et de message prises en charge par le client et le gestionnaire de files d'attente sur une connexion client sont transmises à un exit de canal en tant que collections dans les zones hdrCompList et msgCompList d'un objet MQChannelDefinition . Les techniques utilisées actuellement pour compresser les données d'en-tête et de message sur une connexion client sont transmises à un exit de canal dans les zones CurHdrCompression et CurMsgCompression d'un objet MQChannelExit .

Si la compression est utilisée sur une connexion client, les données sont compressées avant le traitement des exits d'émission de canal et leur extraction après le traitement des exits de réception de canal. Les données transmises aux exits d'émission et de réception sont donc à l'état compressé.

Pour plus d'informations sur la spécification des techniques de compression et sur les techniques de compression disponibles, voir Class com.ibm.mq.MQEnvironment et Interface com.ibm.mq.MQC.

Partage d'une connexion TCP/IP dans IBM MQ classes for Java

Plusieurs instances d'un canal MQI peuvent être créées pour partager une connexion TCP/IP unique.

Dans IBM MQ classes for Java, vous utilisez la variable MQEnvironment.sharingConversations pour contrôler le nombre de conversations pouvant partager une seule connexion TCP/IP.

L'attribut SHARECNV est une approche optimale du partage de connexion. Par conséquent, lorsqu'une valeur de SHARECNV supérieure à 0 est utilisée avec IBM MQ classes for Java , il n'est pas garanti qu'une nouvelle demande de connexion partagera toujours une connexion déjà établie.

Regroupement de connexions dans IBM MQ classes for Java

IBM MQ classes for Java permet de mettre en pool les connexions de secours en vue de leur réutilisation.

IBM MQ classes for Java fournit une prise en charge supplémentaire pour les applications qui gèrent plusieurs connexions aux gestionnaires de files d'attente IBM MQ . Lorsqu'une connexion n'est plus nécessaire, au lieu de la détruire, elle peut être mise en pool et réutilisée ultérieurement. Cela peut fournir une amélioration substantielle des performances pour les applications et les middlewares qui se connectent en série à des gestionnaires de files d'attente arbitraires.

IBM MQ fournit un pool de connexions par défaut. Les applications peuvent activer ou désactiver ce pool de connexions en enregistrant et désenregistrer des jetons via la classe MQEnvironment. Si le pool est actif lorsque IBM MQ classes for Java construit un objet MQQueueManager , il recherche ce pool par défaut et réutilise toute connexion appropriée. Lorsqu'un appel MQQueueManager.disconnect () est émis, la connexion sous-jacente est renvoyée au pool.

Les applications peuvent également construire un pool de connexions MQSimpleConnectionManager pour une utilisation particulière. Ensuite, l'application peut spécifier ce pool lors de la construction d'un objet MQQueueManager ou transmettre ce pool à MQEnvironment pour qu'il soit utilisé comme pool de connexions par défaut.

Pour éviter que les connexions n'utilisent trop de ressources, vous pouvez limiter le nombre total de connexions pouvant être gérées par un objet MQSimpleConnectionManager, ainsi que la taille du pool de connexions. La définition de limites est utile en cas de demandes de connexions conflictuelles au sein d'une machine virtuelle Java.

Par défaut, la méthode getMaxConnections () renvoie la valeur zéro, ce qui signifie qu'il n'y a pas de limite au nombre de connexions que l'objet MQSimpleConnectionManager peut traiter. Vous pouvez définir une limite à l'aide de la méthode setMaxConnections (). Si vous définissez une limite et que la limite est atteinte, une demande de connexion supplémentaire peut entraîner l'émission d'une exception MQException, avec le code anomalie MQRC_MAX_CONNS_LIMIT_ATTEINTES.

Contrôle du pool de connexions par défaut dans IBM MQ classes for Java

Cet exemple montre comment utiliser le pool de connexions par défaut.

Prenons l'exemple d'application suivant, 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 prend la liste des gestionnaires de files d'attente locaux à partir de la ligne de commande, se connecte à chacun d'eux et effectue certaines opérations. Toutefois, lorsque la ligne de commande répertorie plusieurs fois le même gestionnaire de files d'attente, il est plus efficace de ne se connecter qu'une seule fois et de réutiliser cette connexion plusieurs fois.

IBM MQ classes for Java fournit un pool de connexions par défaut que vous pouvez utiliser pour cela. Pour activer le pool, utilisez l'une des méthodes MQEnvironment.addConnectionPoolToken(). Pour désactiver le pool, utilisez MQEnvironment.removeConnectionPoolToken().

L'exemple d'application suivant, MQApp2, est fonctionnellement identique à MQApp1, mais se connecte une seule fois à chaque gestionnaire de files d'attente.

```
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 première ligne en gras active le pool de connexions par défaut en enregistrant un objet MQPoolToken avec MQEnvironment.

Le constructeur MQQueueManager recherche désormais dans ce pool une connexion appropriée et ne crée une connexion au gestionnaire de files d'attente que s'il ne parvient pas à en trouver une existante. L'appel qmgr.disconnect() renvoie la connexion au pool pour une réutilisation ultérieure. Ces appels d'API sont identiques à l'exemple d'application MQApp1.

La deuxième ligne mise en évidence désactive le pool de connexions par défaut, ce qui détruit toutes les connexions de gestionnaire de files d'attente stockées dans le pool. Cela est important car sinon, l'application s'arrêterait avec un certain nombre de connexions de gestionnaire de files d'attente actives dans le pool. Cette situation peut entraîner des erreurs qui apparaîtraient dans les journaux du gestionnaire de files d'attente.

Si une application utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, le constructeur MQQueueManager recherche d'abord dans la table une définition de canal de connexion client appropriée. S'il en trouve une, le constructeur recherche dans le pool de connexions par défaut une connexion pouvant être utilisée pour le canal. Si le constructeur ne parvient pas à trouver une connexion appropriée dans le pool, il recherche ensuite dans la table de définition de canal du client la définition de canal de connexion client appropriée suivante et procède comme décrit précédemment. Si le constructeur termine sa recherche dans la table de définition de canal du client et ne parvient pas à trouver une connexion appropriée dans le pool, il lance une deuxième recherche dans la table. Lors de cette recherche, le constructeur tente de créer une nouvelle connexion pour chaque définition de canal de connexion client appropriée et utilise la première connexion qu'il parvient à créer.

Le pool de connexions par défaut stocke un maximum de dix connexions inutilisées et maintient les connexions inutilisées actives pendant un maximum de cinq minutes. L'application peut modifier cela (pour plus de détails, voir «Fourniture d'un autre pool de connexions dans IBM MQ classes for Java», à la page 409).

Au lieu d'utiliser MQEnvironment pour fournir un jeton MQPoolToken, l'application peut construire ses propres éléments:

```
   MQPoolToken token=new MQPoolToken();
   MQEnvironment.addConnectionPoolToken(token);
```

Certains fournisseurs d'applications ou de middlewares fournissent des sous-classes de MQPoolToken afin de transmettre des informations à un pool de connexions personnalisé. Ils peuvent être construits et transmis à addConnectionPoolToken() de cette manière afin que des informations supplémentaires puissent être transmises au pool de connexions.

Le pool de connexions par défaut et plusieurs composants dans IBM MQ classes for Java
Cet exemple montre comment ajouter ou supprimer des MQPoolTokens dans un ensemble statique d'objets MQPoolToken enregistrés.

MQEnvironment contient un ensemble statique d'objets MQPoolToken enregistrés. Pour ajouter ou supprimer des MQPoolTokens de cet ensemble, utilisez les méthodes suivantes:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Une application peut être constituée de nombreux composants qui existent indépendamment et qui exécutent des tâches à l'aide d'un gestionnaire de files d'attente. Dans une telle application, chaque composant doit ajouter un jeton MQPoolToken à l'ensemble MQEnvironment pour sa durée de vie.

Par exemple, l'exemple d'application MQApp3 crée dix unités d'exécution et démarre chacune d'elles. Chaque unité d'exécution enregistre son propre jeton MQPoolToken, attend un certain temps, puis se connecte au gestionnaire de files d'attente. Une fois l'unité d'exécution déconnectée, elle supprime son propre jeton MQPoolToken.

Le pool de connexions par défaut reste actif tant qu'il existe au moins un jeton dans l'ensemble de MQPoolTokens. Il reste donc actif pendant toute la durée de cette application. L'application n'a pas besoin de conserver un objet maître dans le contrôle global des unités d'exécution.

```
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);
      }
}
```

Fourniture d'un autre pool de connexions dans IBM MQ classes for Java
Cet exemple montre comment utiliser la classe **com.ibm.mq.MQSimpleConnectionManager** pour fournir un autre pool de connexions.

Cette classe fournit des fonctions de base pour le regroupement de connexions et les applications peuvent utiliser cette classe pour personnaliser le comportement du pool.

Une fois instancié, un gestionnaire MQSimpleConnectionpeut être spécifié sur le constructeur MQQueueManager . Le gestionnaire MQSimpleConnectiongère ensuite la connexion sous-jacente au gestionnaire MQQueueManagerconstruit. Si le gestionnaire MQSimpleConnectioncontient une connexion en pool adaptée, cette connexion est réutilisée et renvoyée au gestionnaire MQSimpleConnectionaprès un appel MQQueueManager.disconnect ().

Le fragment de code suivant illustre ce comportement:

```
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 connexion qui est établie lors du premier constructeur MQQueueManager est stockée dans myConnMan après l'appel qmgr.disconnect(). La connexion est ensuite réutilisée lors du deuxième appel au constructeur MQQueueManager .

La deuxième ligne active le gestionnaire MQSimpleConnection. La dernière ligne désactive MQSimpleConnectionManager, détruisant les connexions contenues dans le pool. Un gestionnaire MQSimpleConnectionest, par défaut, dans MODE_AUTO, qui est décrit plus loin dans cette section.

Un gestionnaire MQSimpleConnectionalloue les connexions les plus récemment utilisées et détruit les connexions les moins récemment utilisées. Par défaut, une connexion est détruite si elle n'a pas été utilisée pendant cinq minutes ou s'il y a plus de dix connexions inutilisées dans le pool. Vous pouvez modifier ces valeurs en appelant MQSimpleConnectionManager.setTimeout().

Vous pouvez également configurer un gestionnaire MQSimpleConnectionà utiliser comme pool de connexions par défaut, à utiliser lorsqu'aucun gestionnaire de connexions n'est fourni sur le constructeur MQQueueManager .

L'application suivante le démontre:

```
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);
      }
}
```

Les lignes en gras créent et configurent un objet MQSimpleConnectionManager. La configuration effectue les opérations suivantes:

- Arrête les connexions qui ne sont pas utilisées pendant une heure
- Limite le nombre de connexions gérées par myConnMan à 75
- Limite le nombre de connexions inutilisées dans le pool à 50

- Définit MODE_AUTO, qui est la valeur par défaut. Cela signifie que le pool est actif uniquement s'il s'agit du gestionnaire de connexions par défaut et qu'il existe au moins un jeton dans l'ensemble de MQPoolTokens détenu par MQEnvironment.

Le nouveau gestionnaire MQSimpleConnectionest ensuite défini comme gestionnaire de connexions par défaut.

Dans la dernière ligne, l'application appelle MQApp3.main(). Cela exécute un certain nombre d'unités d'exécution, où chaque unité d'exécution utilise IBM MQ de manière indépendante. Ces unités d'exécution utilisent myConnMan lorsqu'elles forgent des connexions.

Coordination JTA/JDBC à l'aide de IBM MQ classes for Java

IBM MQ classes for Java prend en charge la méthode MQQueueManager.begin (), qui permet à IBM MQ d'agir en tant que coordinateur pour une base de données qui fournit un pilote compatible JDBC de type 2 ou JDBC de type 4.

Cette prise en charge n'est pas disponible sur toutes les plateformes. Pour savoir quelles plateformes prennent en charge la coordination JDBC , voir Configuration système requise pour IBM MQ.

Pour utiliser la prise en charge XA-JTA, vous devez utiliser la bibliothèque de commutateurs JTA spéciale. La méthode d'utilisation de cette bibliothèque varie selon que vous utilisez Windows ou l'une des autres plateformes.

Configuration de la coordination JTA/JDBC sur Windows

La bibliothèque XA est fournie en tant que DLL avec un nom au format `jdbcxxx.dll`.

Le `jdbcora12.dll` fourni fournit la compatibilité avec Oracle 12C, pour une installation de serveur IBM MQ for Windows .

Sur les systèmes Windows , la bibliothèque XA est fournie en tant que DLL complète. Le nom de cette DLL est `jdbcxxx.dll` où `xxx` indique la base de données pour laquelle la bibliothèque de commutateurs a été compilée. Cette bibliothèque se trouve dans le répertoire `java\lib\jdbc` ou `java\lib64\jdbc` de votre installation IBM MQ classes for Java . Vous devez déclarer la bibliothèque XA, également décrite comme fichier de commutation de chargement, au gestionnaire de files d'attente. Utilisez la console IBM MQ Explorer. Indiquez les détails du fichier de commutation de chargement dans le panneau de propriétés du gestionnaire de files d'attente, sous Gestionnaires de ressources XA. Vous devez uniquement indiquer le nom de la bibliothèque. Exemple :

Pour une base de données Db2 , définissez la zone SwitchFile sur: `dbcdb2`

Pour une base de données Oracle , définissez la zone SwitchFile sur: `jdbcora`

Remarques :

1. Oracle 12C est pris en charge par IBM MQ classes for Java, uniquement sous IBM MQ for Windows.
2. La version prise en charge d' Oracle 12C est 12.1.0.1.0 Enterprise Edition et les groupes de correctifs ultérieurs.
3. Les bases de données Oracle 64 bits sur 64 bits Windows requièrent le client Oracle 32 bits.
4. A l'aide de IBM MQ classes for Java, IBM MQ peut agir en tant que coordinateur de transaction. Toutefois, il n'est pas possible de participer à une transaction de type JTA.

Configuration de la coordination JTA/JDBC sur des plateformes autres que Windows

Des fichiers objet sont fournis. Liez le fichier approprié à l'aide du fichier makefile fourni et déclarez-le au gestionnaire de files d'attente à l'aide du fichier de configuration.

Pour chaque système de gestion de base de données, IBM MQ fournit deux fichiers objet. Vous devez lier un fichier objet pour créer une bibliothèque de commutation 32 bits et lier l'autre fichier objet pour créer une bibliothèque de commutation 64 bits. Pour Db2, le nom de chaque fichier objet est `jdbcdb2.o`et pour Oracle , le nom de chaque fichier objet est `jdbcora.o`.

Vous devez lier chaque fichier objet à l'aide du fichier makefile approprié fourni avec IBM MQ.
Une bibliothèque de commutateurs requiert d'autres bibliothèques, qui peuvent être stockées à des emplacements différents sur des systèmes différents. Toutefois, une bibliothèque de commutation ne

peut pas utiliser la variable d'environnement de chemin de bibliothèque pour localiser ces bibliothèques car la bibliothèque de commutation est chargée par le gestionnaire de files d'attente, qui s'exécute dans un environnement `setuid` . Le fichier makefile fourni garantit donc qu'une bibliothèque de commutation contient les chemins d'accès complets de ces bibliothèques.

Pour créer une bibliothèque de commutation, entrez une commande **make** au format suivant. Pour créer une bibliothèque de commutation 32 bits, entrez la commande dans le répertoire `/java/lib/jdbc` de votre installation IBM MQ . Pour créer une bibliothèque de commutation 64 bits, entrez la commande dans le répertoire `/java/lib64/jdbc` .

```
make DBMS
```

où *SGBD* est le système de gestion de base de données pour lequel vous créez la bibliothèque de commutateurs. Les valeurs valides sont db2 pour Db2 et `oracle` pour Oracle.

Remarque :

- Pour exécuter des applications 32 bits, vous devez créer une bibliothèque de commutation 32 bits et 64 bits pour chaque système de gestion de base de données que vous utilisez. Pour exécuter des applications 64 bits, il vous suffit de créer une bibliothèque de commutation 64 bits. Pour Db2, le nom de chaque bibliothèque de commutateurs est `jdbcdb2` et, pour Oracle, le nom de chaque bibliothèque de commutateurs est `jdbcora`. Les fichiers makefile garantissent que les bibliothèques de commutation 32 bits et 64 bits sont stockées dans des répertoires IBM MQ différents. Une bibliothèque de commutateurs 32 bits est stockée dans le répertoire `/java/lib/jdbc` et une bibliothèque de commutateurs 64 bits est stockée dans le répertoire `/java/lib64/jdbc` .
- Etant donné que vous pouvez installer Oracle n'importe où sur un système, les fichiers makefile utilisent la variable d'environnement **ORACLE_HOME** pour localiser l'emplacement où Oracle est installé.
- Si IBM MQ est installé dans un emplacement autre que l'emplacement par défaut, modifiez la valeur de **MQ_INSTALLATION_PATH** dans le fichier makefile.

Une fois que vous avez créé les bibliothèques de commutateurs pour Db2et/ou Oracle, vous devez les déclarer dans votre gestionnaire de files d'attente. Si le fichier de configuration du gestionnaire de files d'attente (qm.ini) contient déjà des sections `XAResourceManager` pour les bases de données Db2 ou Oracle , vous devez remplacer l'entrée `SwitchFile` dans chaque section par l'une des valeurs suivantes:

Pour une base de données Db2

```
SwitchFile=jdbcdb2
```

Pour une base de données Oracle

```
SwitchFile=jdbcora
```

N'indiquez pas le nom de chemin qualifié complet de la bibliothèque de commutation 32 bits ou 64 bits. Indiquez uniquement le nom de la bibliothèque.

Si le fichier de configuration du gestionnaire de files d'attente ne contient pas encore de strophes `XAResourceManager` pour les bases de données Db2 ou Oracle , ou si vous souhaitez ajouter des strophes `XAResourceManager` supplémentaires, voir Administration de IBM MQ pour plus d'informations sur la construction d'une strophe `XAResourceManager` . Toutefois, chaque entrée `SwitchFile` d'une nouvelle section `XAResourceManager` doit être exactement comme décrit précédemment pour une base de données Db2 ou Oracle . Vous devez également inclure l'entrée `ThreadOfControl=PROCESS`.

Après avoir mis à jour le fichier de configuration du gestionnaire de files d'attente et vous être assuré que toutes les variables d'environnement de base de données appropriées ont été définies, vous pouvez redémarrer le gestionnaire de files d'attente.

Utilisation de la coordination JTA/JDBC
Codez vos appels d'API comme dans l'exemple fourni.

La séquence de base des appels d'API pour une application utilisateur est la suivante:

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >

qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

xads dans l'appel getJDBCConnection est une implémentation spécifique à la base de données de l'interface XADataSource , qui définit les détails de la base de données à laquelle se connecter. Consultez la documentation de votre base de données pour déterminer comment créer un objet XADataSource approprié à transmettre à getJDBCConnection.

Vous devez également mettre à jour votre chemin d'accès aux classes avec les fichiers jar spécifiques à la base de données appropriés pour exécuter le travail JDBC .

Si vous devez vous connecter à plusieurs bases de données, vous devez appeler getJDBCConnection plusieurs fois pour effectuer la transaction sur plusieurs connexions différentes.

Il existe deux formes de getJDBCConnection, reflétant les deux formes 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
```

Ces méthodes déclarent une exception dans leurs clauses throws afin d'éviter les problèmes avec le vérificateur JVM pour les clients qui n'utilisent pas les fonctions JTA. L'exception réelle émise est javax.transaction.xa.XAException qui requiert que le fichier jta.jar soit ajouté au chemin d'accès aux classes pour les programmes qui n'en avaient pas besoin auparavant.

Pour utiliser le support JTA/JDBC , vous devez inclure l'instruction suivante dans votre application:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Problèmes connus et limitations de la coordination JTA/JDBC
Certains problèmes et limitations de la prise en charge de JTA/JDBC dépendent du système de gestion de base de données utilisé. Par exemple, les pilotes JDBC testés se comportent différemment lorsque la base de données est arrêtée alors qu'une application est en cours d'exécution. Si la connexion à la base de données utilisée par une application est interrompue, il existe des étapes que l'application peut effectuer pour rétablir une nouvelle connexion au gestionnaire de files d'attente et à la base de données afin qu'elle puisse ensuite utiliser ces nouvelles connexions pour effectuer le travail transactionnel requis.

Etant donné que le support JTA/JDBC effectue des appels aux pilotes JDBC , l'implémentation de ces pilotes JDBC peut avoir un effet significatif sur le comportement du système. En particulier, les pilotes JDBC testés se comportent différemment lorsque la base de données est arrêtée alors qu'une application est en cours d'exécution.

Important : Evitez toujours d'arrêter brutalement une base de données alors que des applications y détiennent des connexions ouvertes.

Remarque : Une application IBM MQ classes for Java doit se connecter à l'aide du mode de liaison pour que IBM MQ agisse en tant que coordinateur de base de données.

Plusieurs sections XAResourceManager
L'utilisation de plusieurs strophes XAResourceManager dans un fichier de configuration de gestionnaire de files d'attente, `qm.ini`, n'est pas prise en charge. Toute strophe XAResourceManager autre que la première est ignorée.

Db2

Parfois, Db2 renvoie une erreur SQL0805N . Ce problème peut être résolu à l'aide de la commande CLP suivante:

```
DB2 bind @db2cli.lst blocking all grant public
```

Pour plus d'informations, reportez-vous à la documentation de Db2.

La section XAResourceManager doit être configurée pour utiliser ThreadOfControl=PROCESS. Pour Db2 8.1 et les versions ultérieures, cela ne correspond pas à l'unité d'exécution par défaut du paramètre de contrôle pour Db2, de sorte que toc=p doit être spécifié dans la chaîne d'ouverture XA. Voici un exemple de section XAResourceManager pour Db2 avec la coordination JTA/JDBC :

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Cela n'empêche pas les applications Java qui utilisent la coordination JTA/JDBC d'être elles-mêmes multiprocessus.

Oracle
L'appel de la méthode JDBC Connection.close() après MQQueueManager.disconnect () génère une exception SQLException. Appelez Connection.close() avant MQQueueManager.disconnect () ou omettez l'appel à Connection.close().

Traitement des problèmes liés aux connexions de base de données

Lorsqu'une application IBM MQ classes for Java utilise le support JTA/JDBC fourni par IBM MQ, elle effectue généralement les étapes suivantes:

1. Crée un objet MQQueueManager pour représenter une connexion au gestionnaire de files d'attente qui fera office de gestionnaire de transactions.
2. Construit un objet XADataSource qui contient des détails sur la connexion à la base de données qui sera répertoriée dans la transaction.
3. Appelle la méthode MQQueueManager.getJDBCConnection(XADataSource) en transmettant la source XADataSource créée précédemment. Ainsi, IBM MQ classes for Java établit une connexion à la base de données.
4. Appelle la méthode MQQueueManager.begin () pour démarrer la transaction XA.
5. Effectue le travail de messagerie et de base de données.
6. Lorsque tout le travail requis est terminé, appelle la méthode MQQueueManager.commit (). La transaction XA est terminée.
7. Si une nouvelle transaction XA est requise à ce stade, l'application peut répéter les étapes 4, 5 et 6.
8. Une fois l'application terminée, elle doit fermer la connexion de base de données créée à l'étape 3, puis appeler la méthode MQQueueManager.disconnect () pour la déconnecter du gestionnaire de files d'attente.

Le IBM MQ classes for Java gère une liste interne de toutes les connexions de base de données qui ont été créées lorsqu'une application appelle MQQueueManager.getJDBCConnection(XADataSource). Si un gestionnaire de files d'attente doit communiquer avec la base de données lors du traitement de la transaction XA, le traitement suivant a lieu:

1. Les appels du gestionnaire de files d'attente dans IBM MQ classes for Java, en transmettant les détails de l'appel XA qui doit être transmis à la base de données.
2. Le IBM MQ classes for Java recherche ensuite la connexion appropriée dans la liste, puis utilise cette connexion pour transmettre l'appel XA à la base de données.

Si la connexion à la base de données est perdue à un moment quelconque de ce traitement, l'application doit:

1. Annuler tout travail existant effectué sous la transaction, en appelant la méthode MQQueueManager.backout ().
2. Fermez la connexion à la base de données. Cela devrait entraîner le IBM MQ classes for Java à supprimer les détails de la connexion à la base de données interrompue de sa liste interne.
3. Déconnectez-vous du gestionnaire de files d'attente en appelant la méthode MQQueueManager.disconnect ().
4. Etablissez une nouvelle connexion au gestionnaire de files d'attente en construisant un nouvel objet MQQueueManager .
5. Créez une nouvelle connexion de base de données en appelant la méthode MQQueueManager.getJDBCConnection(XADataSource).
6. Réexécutez le travail transactionnel.

Cela permet à l'application de rétablir une nouvelle connexion au gestionnaire de files d'attente et à la base de données, puis d'utiliser ces connexions pour effectuer le travail transactionnel requis.

Prise en charge de TLS (Transport Layer Security) dans IBM MQ classes for Java

Les applications client IBM MQ classes for Java prennent en charge le chiffrement TLS. Vous avez besoin d'un fournisseur JSSE pour utiliser le chiffrement TLS.

Les applications client IBM MQ classes for Java utilisant TRANSPORT (CLIENT) prennent en charge le chiffrement TLS. TLS fournit le chiffrement des communications, l'authentification et l'intégrité des messages. Il est généralement utilisé pour sécuriser les communications entre deux homologues sur Internet ou dans un intranet.

IBM MQ classes for Java utilise Java Secure Socket Extension (JSSE) pour gérer le chiffrement TLS et requiert donc un fournisseur JSSE. Les machines virtuelles Java JSE v1.4 ont un fournisseur JSSE intégré. Les détails de la gestion et du stockage des certificats peuvent varier d'un fournisseur à l'autre. Pour plus d'informations à ce sujet, reportez-vous à la documentation de votre fournisseur JSSE.

Cette section suppose que votre fournisseur JSSE est correctement installé et configuré, et que des certificats appropriés ont été installés et mis à la disposition de votre fournisseur JSSE.

Si votre application client IBM MQ classes for Java utilise une table de définition de canal du client (CCDT) pour se connecter à un gestionnaire de files d'attente, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for Java», à la page 386.

Activation de TLS dans IBM MQ classes for Java

Pour activer TLS, spécifiez une CipherSuite. Il existe deux façons de spécifier une CipherSuite.

TLS est pris en charge uniquement pour les connexions client. Pour activer TLS, vous devez spécifier la CipherSuite à utiliser lors de la communication avec le gestionnaire de files d'attente et cette CipherSuite doit correspondre à la spécification CipherSpec définie sur le canal cible. De plus, la suite de chiffrement nommée CipherSuite doit être prise en charge par votre fournisseur JSSE. Toutefois, les CipherSuites sont distinctes des CipherSpecs et ont donc des noms différents. «CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for Java», à la page 420 contient une table mappant les CipherSpecs pris en charge par IBM MQ à leurs CipherSuites équivalentes connues de JSSE.

Pour activer TLS, spécifiez CipherSuite à l'aide de la variable de membre statique sslCipherSuite de MQEnvironment. L'exemple suivant se connecte à un canal SVRCONN nommé SECURE.SVRCONN.CHANNEL, qui a été configuré pour exiger TLS avec un CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA256:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_WITH_AES_128_CBC_SHA256";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Bien que le CipherSpec du canal soit TLS_RSA_WITH_AES_128_CBC_SHA256, l'application Java doit spécifier une CipherSuite de SSL_RSA_WITH_AES_128_CBC_SHA256. Voir «CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for Java», à la page 420 pour la liste des mappages entre les CipherSpecs et les CipherSuites.

Une application peut également spécifier une CipherSuite en définissant la propriété d'environnement CMQC.SSL_CIPHER_SUITE_PROPERTY.

Vous pouvez également utiliser la table de définition de canal du client (CCDT). Pour plus d'informations, voir «Utilisation d'une table de définition de canal du client avec IBM MQ classes for Java», à la page 386

Si vous avez besoin d'une connexion client pour utiliser une CipherSuite prise en charge par le fournisseur IBM Java JSSE FIPS (IBMJSSEFIPS), une application peut définir la zone sslFipsRequired dans la classe MQEnvironment sur `true`. L'application peut également définir la propriété d'environnement CMQC.SSL_FIPS_REQUIRED_PROPERTY. La valeur par défaut est `false`, ce qui signifie qu'une connexion client peut utiliser n'importe quelle CipherSuite prise en charge par IBM MQ.

Si une application utilise plusieurs connexions client, la valeur de la zone obligatoire sslFipsutilisée lorsque l'application crée la première connexion client détermine la valeur utilisée lorsque l'application crée une connexion client ultérieure. Par conséquent, lorsque l'application crée une connexion client ultérieure, la valeur de la zone sslFipsRequired est ignorée. Vous devez redémarrer l'application si vous souhaitez utiliser une valeur différente pour la zone sslFipsRequired.

Pour que la connexion à l'aide de TLS aboutisse, le magasin de clés de confiance JSSE doit être configuré avec des certificats racine d'autorité de certification à partir desquels le certificat présenté par le gestionnaire de files d'attente peut être authentifié. De même, si SSLClientAuth sur le canal SVRCONN a été défini sur MQSSL_CLIENT_AUTH_REQUIRED, le magasin de clés JSSE doit contenir un certificat d'identification sécurisé par le gestionnaire de files d'attente.

Référence associée

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

Utilisation du nom distinctif du gestionnaire de files d'attente dans IBM MQ classes for Java

Le gestionnaire de files d'attente s'identifie lui-même à l'aide d'un certificat TLS, qui contient un nom distinctif (DN). Une application client IBM MQ classes for Java peut utiliser ce nom distinctif pour s'assurer qu'elle communique avec le gestionnaire de files d'attente approprié.

Un modèle de nom distinctif est spécifié à l'aide de la variable de nom sslPeerde MQEnvironment. Par exemple, en définissant:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permet à la connexion d'aboutir uniquement si le gestionnaire de files d'attente présente un certificat dont le nom usuel commence par QMGR., et au moins deux noms d'unité organisationnelle, le premier devant être IBM et le second WebSphere.

Si le nom sslPeerest défini, les connexions aboutissent uniquement si un modèle valide est défini et que le gestionnaire de files d'attente présente un certificat correspondant.

Une application peut également spécifier le nom distinctif du gestionnaire de files d'attente en définissant la propriété d'environnement CMQC.SSL_PEER_NAME_PROPERTY. Pour plus d'informations sur les noms distinctifs, voir Noms distinctifs.

Utilisation de listes de révocation de certificat dans IBM MQ classes for Java

Indiquez les listes de révocation de certificat à utiliser via la classe java.security.cert.CertStore . IBM MQ classes for Java vérifie ensuite les certificats par rapport à la liste de révocation de certificat spécifiée.

Une liste de révocation de certificat (CRL) est un ensemble de certificats qui ont été révoqués, soit par l'autorité de certification émettrice, soit par l'organisation locale. Les listes de révocation de certificat sont généralement hébergées sur des serveurs LDAP. Avec Java 2 v1.4, un serveur de liste de révocation de certificat peut être spécifié lors de la connexion et le certificat présenté par le gestionnaire de files d'attente est vérifié par rapport à la liste de révocation de certificat avant que la connexion ne soit autorisée. Pour plus d'informations sur les listes de révocation de certificat et IBM MQ, voir Utilisation des listes de révocation de certificat et des listes de révocation d'autorité et Accès aux listes de révocation de certificat et aux listes de révocation de certificat avec IBM MQ classes for Java et IBM MQ classes for JMS.

Remarque : Pour utiliser un CertStore avec une CRL hébergée sur un serveur LDAP, assurez-vous que votre kit de développement de logiciels (SDK) Java est compatible avec la CRL. Certains logiciels SDK exigent que la CRL soit conforme à la RFC 2587, qui définit un schéma pour LDAP v2. La plupart des serveurs LDAP v3 utilisent RFC 2256 à la place.

Les CRL à utiliser sont spécifiées via la classe java.security.cert.CertStore . Pour plus de détails sur l'obtention d'instances de CertStore, reportez-vous à la documentation relative à cette classe. Pour créer un CertStore basé sur un serveur LDAP, créez d'abord une instance de paramètres LDAPCertStore, initialisée avec les paramètres de serveur et de port à utiliser. Exemple :

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Après avoir créé une instance CertStoreParameters, utilisez le constructeur statique sur CertStore pour créer un CertStore de type LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

D'autres types de CertStore (par exemple, Collection) sont également pris en charge. Généralement, plusieurs serveurs de liste de révocation de certificat sont configurés avec des informations de liste de révocation de certificat identiques pour assurer la redondance. Lorsque vous disposez d'un objet CertStore pour chacun de ces serveurs CRL, placez-les tous dans une collection appropriée. L'exemple suivant illustre les objets CertStore placés dans une ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

Cette collection peut être définie dans la variable statique MQEnvironment, sslCertStores, avant la connexion pour activer la vérification CRL:

```
MQEnvironment.sslCertStores = crls;
```

Le certificat présenté par le gestionnaire de files d'attente lors de la configuration d'une connexion est validé comme suit:

1. Le premier objet CertStore de la collection identifiée par les magasins sslCertest utilisé pour identifier un serveur CRL.
2. Une tentative est effectuée pour contacter le serveur CRL.
3. Si la tentative aboutit, le serveur recherche une correspondance pour le certificat.
   a. Si le certificat est révoqué, le processus de recherche est terminé et la demande de connexion échoue avec le code anomalie MQRC_SSL_CERTIFICATE_RÉVOQUÉ.
   b. Si le certificat est introuvable, le processus de recherche est terminé et la connexion est autorisée à continuer.
4. Si la tentative de contact du serveur échoue, l'objet CertStore suivant est utilisé pour identifier un serveur CRL et le processus se répète à l'étape 2.

S'il s'agit du dernier CertStore de la collection, ou si la collection ne contient aucun objet CertStore , le processus de recherche a échoué et la demande de connexion a échoué avec le code anomalie MQRC_SSL_CERT_STORE_ERROR.

L'objet Collection détermine l'ordre dans lequel les CertStores sont utilisés.

La collection de CertStores peut également être définie à l'aide de la propriété CMQC.SSL_CERT_STORE_PROPERTY. Par commodité, cette propriété permet également de spécifier un seul CertStore sans être membre d'une collection.

Si sslCertStores est défini sur null, aucune vérification de la liste de révocation de certificat n'est effectuée. Cette propriété est ignorée si la suite sslCiphern'est pas définie.

Renégociation de la clé secrète dans IBM MQ classes for Java
Une application client IBM MQ classes for Java peut contrôler à quel moment la clé secrète utilisée pour le chiffrement sur une connexion client est renégociée, en termes de nombre total d'octets envoyés et reçus.

L'application peut effectuer cette opération de l'une des manières suivantes: si l'application utilise plusieurs de ces méthodes, les règles de priorité habituelles s'appliquent.

- En définissant la zone sslResetCount dans la classe MQEnvironment.
- En définissant la propriété d'environnement MQC.SSL_RESET_COUNT_PROPERTY dans un objet Hashtable. L'application affecte ensuite la table de hachage à la zone `properties` de la classe MQEnvironment ou transmet la table de hachage à un objet MQQueueManager sur son constructeur.

La valeur de la zone sslResetCount ou de la propriété d'environnement MQC.SSL_RESET_COUNT_PROPERTY représente le nombre total d'octets envoyés et reçus par le code client IBM MQ classes for Java avant la renégociation de la clé secrète. Le nombre d'octets envoyés est le nombre avant chiffrement et le nombre d'octets reçus est le nombre après déchiffrement. Le nombre d'octets inclut également les informations de contrôle envoyées et reçues par le client IBM MQ classes for Java .

Si le nombre de réinitialisations est égal à zéro, ce qui correspond à la valeur par défaut, la clé secrète n'est jamais renégociée. Le nombre de réinitialisations est ignoré si CipherSuite n'est pas spécifié.

Fourniture d'une fabrique SSLSocketFactory personnalisée dans IBM MQ classes for Java
Si vous utilisez une fabrique de sockets JSSE personnalisée, définissez MQEnvironment.sslSocketFactory sur l'objet de fabrique personnalisé. Les détails varient entre les différentes implémentations JSSE.

Différentes implémentations JSSE peuvent fournir des fonctionnalités différentes. Par exemple, une implémentation JSSE spécialisée peut permettre la configuration d'un modèle particulier de matériel de chiffrement. En outre, certains fournisseurs JSSE permettent la personnalisation des magasins de clés et des magasins de clés de confiance par programme ou permettent de modifier le choix du certificat d'identité du magasin de clés. Dans JSSE, toutes ces personnalisations sont abstraites dans une classe de fabrique, javax.net.ssl.SSLSocketFactory.

Consultez la documentation JSSE pour plus de détails sur la création d'une implémentation SSLSocketFactory personnalisée. Les détails varient d'un fournisseur à l'autre, mais une séquence typique d'étapes peut être:

1. Créer un objet SSLContext à l'aide d'une méthode statique sur SSLContext
2. Initialisez ce contexte SSL avec les implémentations KeyManager et TrustManager appropriées (créées à partir de leurs propres classes de fabrique)
3. Créez une fabrique SSLSocketFactory à partir de SSLContext

Lorsque vous disposez d'un objet SSLSocketFactory , définissez MQEnvironment.sslSocketFactory sur l'objet de fabrique personnalisé. Exemple :

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java Utilisez cette fabrique SSLSocketFactory pour vous connecter au gestionnaire de files d'attente IBM MQ . Cette propriété peut également être définie à l'aide de la propriété CMQC.SSL_SOCKET_FACTORY_PROPERTY. Si sslSocketFactory est défini sur null, la valeur par défaut SSLSocketFactory de la machine virtuelle Java est utilisée. Cette propriété est ignorée si la suite sslCiphern'est pas définie.

Lorsque vous utilisez des SSLSocketFactoriespersonnalisées, tenez compte de l'effet du partage de connexion TCP/IP. Si le partage de connexion est possible, un nouveau socket n'est pas demandé pour SSLSocketFactory fourni, même si le socket produit est différent d'une manière ou d'une autre dans le contexte d'une demande de connexion ultérieure. Par exemple, si un certificat client différent doit être présenté sur une connexion ultérieure, le partage de connexion ne doit pas être autorisé.

Modification du magasin de clés JSSE ou du magasin de clés de confiance dans IBM MQ classes for Java
Si vous modifiez le magasin de clés JSSE ou le magasin de clés de confiance, vous devez effectuer certaines actions pour que les modifications prennent effet.

Si vous modifiez le contenu du magasin de clés ou du magasin de clés de confiance JSSE ou que vous modifiez l'emplacement du magasin de clés ou du magasin de clés de confiance, les applications IBM MQ classes for Java qui s'exécutent à ce moment-là ne prennent pas automatiquement en compte les modifications. Pour que les modifications prennent effet, les actions suivantes doivent être effectuées:

- Les applications doivent fermer toutes leurs connexions et détruire toutes les connexions inutilisées dans les pools de connexions.
- Si votre fournisseur JSSE met en cache les informations du magasin de clés et du magasin de clés de confiance, ces informations doivent être actualisées.

Une fois ces actions effectuées, les applications peuvent recréer leurs connexions.

En fonction de la façon dont vous concevez vos applications et de la fonction fournie par votre fournisseur JSSE, il peut être possible d'effectuer ces actions sans arrêter ni redémarrer vos applications. Toutefois, l'arrêt et le redémarrage des applications peuvent être la solution la plus simple.

Traitement des erreurs lors de l'utilisation de TLS avec IBM MQ classes for Java
Un certain nombre de codes raison peuvent être émis par IBM MQ classes for Java lors de la connexion à un gestionnaire de files d'attente à l'aide de TLS.

Ils sont décrits dans la liste suivante:

MQRC_SSL_NOT_ALLOWED
La propriété sslCipherde la suite a été définie, mais la connexion des liaisons a été utilisée. Seule la connexion client prend en charge TLS.

MQRC_JSSE_ERREUR
Le fournisseur JSSE a signalé une erreur qui n'a pas pu être traitée par IBM MQ. Cela peut être dû à un problème de configuration avec JSSE ou au fait que le certificat présenté par le gestionnaire de files d'attente n'a pas pu être validé. L'exception générée par JSSE peut être extraite à l'aide de la méthode getCause() sur MQException.

MQRC_SSL_INITIALIZATION_ERROR
Un appel MQCONN ou MQCONNX a été émis avec les options de configuration TLS spécifiées, mais une erreur s'est produite lors de l'initialisation de l'environnement TLS.

MQRC_SSL_PEER_NAME_MISMATCH
Le modèle de nom distinctif spécifié dans la propriété sslPeerName ne correspond pas au nom distinctif présenté par le gestionnaire de files d'attente.

MQRC_SSL_PEER_NAME_ERROR
Le modèle de nom distinctif spécifié dans la propriété sslPeerName n'est pas valide.

MQRC_UNSUPPORTED_CIPHER_SUITE
La suite CipherSuite nommée dans sslCipherSuite n'a pas été reconnue par le fournisseur JSSE. Une liste complète des CipherSuites prises en charge par le fournisseur JSSE peut être obtenue par un programme à l'aide de la méthode SSLSocketFactory.getSupportedCipherSuites(). La liste des

CipherSuites qui peuvent être utilisées pour communiquer avec IBM MQ se trouve dans «CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for Java», à la page 420.

MQRC_SSL_CERTIFICATE_RÉVOQUÉ
Le certificat présenté par le gestionnaire de files d'attente a été trouvé dans une CRL spécifiée avec la propriété sslCertStores. Mettez à jour le gestionnaire de files d'attente pour qu'il utilise des certificats de confiance.

MQRC_SSL_CERT_STORE_ERREUR
Aucun des CertStores fournis n'a pu être recherché pour le certificat présenté par le gestionnaire de files d'attente. La méthode MQException.getCause() renvoie l'erreur qui s'est produite lors de la recherche du premier CertStore tenté. Si l'exception causale est NoSuchElementException, ClassCastException ou NullPointerException, vérifiez que la collection spécifiée dans la propriété sslCertStores contient au moins un objet CertStore valide.

CipherSpecs TLS et suites de chiffrement dans IBM MQ classes for Java
La capacité des applications IBM MQ classes for Java à établir des connexions à un gestionnaire de files d'attente dépend du CipherSpec spécifié à l'extrémité serveur du canal MQI et de la CipherSuite spécifiée à l'extrémité client.

Le tableau suivant répertorie les CipherSpecs pris en charge par IBM MQ et leurs CipherSuiteséquivalentes.

Deprecated Consultez la rubrique CipherSpecs CipherSpecs pour voir si des CipherSpecs, répertoriés dans le tableau suivant, ont été dépréciés par IBM MQ et, si tel est le cas, dans quelle mesure la mise à jour de CipherSpec a été dépréciée.

Important : Les CipherSuites répertoriées sont celles prises en charge par l'environnement d'exécution IBM Java (JRE) fourni avec IBM MQ. Les CipherSuites répertoriées incluent celles prises en charge par l'environnement d'exécution Java Oracle Java . Pour plus d'informations sur la configuration de votre application pour l'utilisation d'un environnement d'exécution Java Oracle Java , voir Configuration de votre application pour l'utilisation des mappages IBM Java ou Oracle Java CipherSuite.

Le tableau indique également le protocole utilisé pour la communication et indique si la CipherSuite est conforme à la norme FIPS 140-2.

Remarque : Sous AIX, Linux, and Windows, IBM MQ fournit la conformité à la norme FIPS 140-2 via le module cryptographique IBM Crypto for C (ICC) . Le certificat de ce module a été déplacé vers le statut Historique. Les clients doivent afficher le certificatIBM Crypto for C (ICC) et prendre connaissance des conseils fournis par le NIST. Un module FIPS 140-3 de remplacement est actuellement en cours et son statut peut être affiché en le recherchant dans la liste des modules NIST CMVP en cours de traitement.

IBM MQ Operator 3.2.0 et l'image de conteneur du gestionnaire de files d'attente à partir de la version 9.4.0.0 sont basés sur UBI 9. La conformité à la norme FIPS 140-3 est actuellement en attente et son statut peut être affiché en recherchant "Red Hat Enterprise Linux 9- OpenSSL FIPS Provider" dans la liste de processus des modules CMVP NIST.

Les suites de chiffrement désignées comme conformes à la norme FIPS 140-2 peuvent être utilisées si l'application n'a pas été configurée pour appliquer la conformité à la norme FIPS 140-2, mais si la conformité à la norme FIPS 140-2 a été configurée pour l'application (voir les remarques suivantes sur la configuration), seules les CipherSuites marquées comme compatibles à la norme FIPS 140-2 peuvent être configurées. La tentative d'utilisation d'autres CipherSuites génère une erreur.

Remarque : Chaque environnement d'exécution Java peut disposer de plusieurs fournisseurs de sécurité cryptographique, chacun d'eux pouvant fournir une implémentation de la même CipherSuite. Cependant, tous les fournisseurs de sécurité ne sont pas certifiés FIPS 140-2. Si la conformité à la norme FIPS 140-2 n'est pas appliquée pour une application, il est possible qu'une implémentation non certifiée de CipherSuite soit utilisée. Les implémentations non certifiées peuvent ne pas fonctionner conformément à la norme FIPS 140-2, même si CipherSuite répond théoriquement au niveau de sécurité minimal requis par la norme. Pour plus d'informations sur la configuration de la mise en application de la norme FIPS 140-2 dans les applications IBM MQ Java , voir les remarques suivantes.

Pour plus d'informations sur la conformité FIPS 140-2 et Suite-B pour CipherSpecs et CipherSuites, voir Spécification de CipherSpecs. Vous devrez peut-être également prendre connaissance des informations concernant les Federal Information Processing Standardsaméricaines.

Pour utiliser l'ensemble complet des CipherSuites et pour utiliser la conformité FIPS 140-2 et/ou Suite-B certifiée, un environnement d'exécution Java approprié est requis. IBM Java 7 Service Refresh 4 Fix Pack 2 ou un niveau supérieur de IBM JRE fournit la prise en charge appropriée pour les suites de chiffrement TLS 1.2 CipherSuites répertoriées dans Tableau 59, à la page 421.

Pour pouvoir utiliser les chiffrements TLS 1.3 , l'environnement d'exécution Java qui exécute votre application doit prendre en charge TLS 1.3.

Remarque : Pour utiliser des CipherSuites, les fichiers de règles "sans restriction" doivent être configurés dans l'environnement d'exécution Java. Pour plus de détails sur la configuration des fichiers de règles dans un SDK ou un JRE, voir la rubrique *IBM SDK Policy files* dans le document *Security Reference for IBM SDK, Java Technology Edition* correspondant à la version que vous utilisez.

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes

CipherSpec **«1», à la page 439**	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_128_CBC_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_128_GCM_SHA256	SSL_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_256_CBC_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_AES_256_GCM_SHA384	SSL_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS 1.2	oui
ECDHE_ECDSA_NULL_SHA256	SSL_ECDHE_ECDSA_WITH_NULL_SHA	TLS_ECDHE_ECDSA_WITH_NULL_SHA	TLS 1.2	non

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	non
ECDHE_RSA_3DES_EDE_CBC_SHA256	SSL_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_128_CBC_SHA256	SSL_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_128_GCM_SHA256	SSL_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_256_CBC_SHA384	SSL_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS 1.2	oui

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_AES_256_GCM_SHA384	SSL_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	oui
ECDHE_RSA_NULL_SHA256	SSL_ECDHE_RSA_WITH_NULL_SHA	TLS_ECDHE_RSA_WITH_NULL_SHA	TLS 1.2	non

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	non
TLS_RSA_WITH_3DES_EDE_CBC_SHA «2», à la page 439	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	non «4», à la page 439

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	non «4», à la page 439
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2	non «4», à la page 439

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	non «4», à la page 439
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0	non «4», à la page 439

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	non «4», à la page 439
TLS_RSA_WITH_AES_256_GCM_SHA384	SSL_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2	non «4», à la page 439

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	non
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	TLS_RSA_WITH_NULL_SHA256	TLS 1.2	non
TLS_RSA_WITH_RC4_128_SHA256	SSL_RSA_WITH_RC4_128_SHA	SSL_RSA_WITH_RC4_128_SHA	TLS 1.2	non

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
ANY_TLS12	*TLS12	*TLS12	TLS 1.2	oui
TLS_AES_128_GCM_SHA256 «3», à la page 439	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS v1.3	non
TLS_AES_256_GCM_SHA384 «3», à la page 439	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS v1.3	non

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_CHACHA20_POLY1305_SHA256 «3», à la page 439	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS v1.3	non
TLS_AES_128_CCM_SHA256 «3», à la page 439	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS v1.3	non

Tableau 59. CipherSpecs pris en charge par IBM MQ et leurs CipherSuites équivalentes (suite)

CipherSpec «1», à la page 439	**Equivalent CipherSuite (IBM JRE)**	**Equivalent CipherSuite (Oracle JRE)**	**Protocole**	**Compatible FIPS 140-2**
TLS_AES_128_CCM_8_SHA256 «3», à la page 439	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS v1.3	non
Tous «3», à la page 439	*ANY	*ANY	Multiple	non
ANY_TLS13 «3», à la page 439	*TLS13	*TLS13	TLS V13	non
ANY_TLS12_OR_HIGHER «3», à la page 439	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS 1.2 et versions ultérieures	non
ANY_TLS13_OR_HIGHER «3», à la page 439	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS 1.3 et versions ultérieures	non

Remarques :

1. Il s'agit de la valeur configurée sur un canal dans IBM MQ, y compris dans une table de définition de canal du client (CCDT) (binaire ou JSON).
2. Deprecated CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA est obsolète. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.
3. Pour pouvoir utiliser les chiffrements TLS v1.3 , l'environnement d'exécution Java (Java runtime environment) exécutant votre application doit prendre en charge TLS v1.3.
4. V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, l'environnement d'exécution Java de IBM Java 8 supprime la prise en charge de l'échange de clés RSA en mode FIPS.

Configuration des suites de chiffrement et de la conformité FIPS dans une application IBM MQ classes for Java

- Une application qui utilise IBM MQ classes for Java peut utiliser l'une des deux méthodes suivantes pour définir CipherSuite pour une connexion:
  - Définissez la zone sslCipherSuite dans la classe MQEnvironment sur le nom CipherSuite .
  - Définissez la propriété CMQC.SSL_CIPHER_SUITE_PROPERTY dans la table de hachage des propriétés transmise au constructeur MQQueueManager pour le nom CipherSuite .
- Une application qui utilise IBM MQ classes for Java peut utiliser l'une des deux méthodes suivantes pour appliquer la conformité à la norme FIPS 140-2:
  - Définissez la zone sslFipsRequired sur true dans la classe MQEnvironment.
  - Affectez à la propriété CMQC.SSL_FIPS_REQUIRED_PROPERTYin la valeur true dans la table de hachage des propriétés transmise au constructeur MQQueueManager .

Configuration de votre application pour l'utilisation des mappages IBM Java ou Oracle Java CipherSuite

V 9.4.0 Depuis la IBM MQ 9.4.0, un chiffrement peut être défini en tant que nom CipherSpec ou CipherSuite et est géré correctement par IBM MQ.

Remarque : Removed La Java propriété système `com.ibm.mq.cfg.useIBMCipherMappings`, qui contrôlait les mappages utilisés dans les versions antérieures de IBM MQ, n'est plus nécessaire et est supprimée du produit à l'adresse IBM MQ 9.4.0.

Limitations de l'interopérabilité

Certaines CipherSuites peuvent être compatibles avec plusieurs IBM MQ CipherSpec, selon le protocole utilisé. Toutefois, seule la combinaison CipherSuite/CipherSpec qui utilise la version TLS spécifiée dans le tableau 1 est prise en charge. La tentative d'utilisation des combinaisons non prises en charge de CipherSuites et CipherSpecs échouera avec une exception appropriée. Les installations utilisant l'une de ces combinaisons CipherSuite/CipherSpec doivent être déplacées vers une combinaison prise en charge.

Le tableau suivant présente les CipherSuites auxquelles cette limitation s'applique.

Tableau 60. CipherSuites et leurs CipherSpecs pris en charge et non pris en charge

CipherSuite	TLS CipherSpec pris en charge	CipherSpec SSL non pris en charge
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA «1», à la page 440	TRIPLE_DES_SHA_US

Tableau 60. CipherSuites et leurs CipherSpecs pris en charge et non pris en charge (suite)		
CipherSuite	**TLS CipherSpec pris en charge**	**CipherSpec SSL non pris en charge**
SSL_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA	DES_SHA_EXPORT
SSL_RSA_WITH_RC4_128_SHA	TLS_RSA_WITH_RC4_128_SHA256	RC4_SHA_US

Remarque :

1. Deprecated Ce CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA est obsolète. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.

Exécution d'applications IBM MQ classes for Java

Si vous écrivez une application (une classe contenant une méthode main ()), à l'aide du client ou du mode de liaison, exécutez votre programme à l'aide de l'interpréteur Java .

Entrez la commande :

```
java -Djava.library.path= library_path MyClass
```

où *chemin_bibliothèque* est le chemin d'accès aux bibliothèques IBM MQ classes for Java . Pour plus d'informations, voir «IBM MQ classes for Java bibliothèques», à la page 367.

Tâches associées

Traçage des applications IBM MQ classes for Java

Traçage de l'adaptateur de ressources IBM MQ

Comportement dépendant de l'environnement IBM MQ classes for Java

IBM MQ classes for Java vous permet de créer des applications qui peuvent s'exécuter sur différentes versions d' IBM MQ. Cette collection de rubriques décrit le comportement des classes Java qui dépendent de ces différentes versions.

IBM MQ classes for Java fournit un noyau de classes, qui fournissent une fonction et un comportement cohérents dans tous les environnements. Les fonctions en dehors de ce coeur dépendent de la capacité du gestionnaire de files d'attente auquel l'application est connectée.

Sauf indication contraire, le comportement affiché est celui décrit dans la référence d'application MQI appropriée au gestionnaire de files d'attente.

Classes de base dans IBM MQ classes for Java

IBM MQ classes for Java contient un ensemble de classes de base, qui peut être utilisé dans tous les environnements.

L'ensemble de classes suivant est considéré comme des classes de base et peut être utilisé dans tous les environnements avec uniquement les variations mineures répertoriées dans le «Restrictions et variantes pour les classes de base de IBM MQ classes for Java», à la page 441.

- Environnement MQ
- Exception MQException
- Options de MQGetMessage

  Exclusion de:

  – MatchOptions
  – GroupStatus

  - SegmentStatus
  - Segmentation
- MQManagedObject

  Exclusion de:
  - consulter ()
  - set ()
- Message MQ

  Exclusion de:
  - groupId
  - messageFlags
  - messageSequenceNuméro
  - décalage
  - originalLength
- MQPoolServices
- MQPoolServicesEvénement
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessage-Options

  Exclusion de:
  - KnownDestCount
  - UnknownDestCount
  - InvalidDestCount
  - recordFields
- Processus MQ
- MQQUEUE
- MQQueueManager

  Exclusion de:
  - begin ()
  - Liste accessDistribution()
- Gestionnaire MQSimpleConnection
- Sujet MQ
- MQC

Remarque :

1. Certaines constantes ne sont pas incluses dans le noyau (voir «Restrictions et variantes pour les classes de base de IBM MQ classes for Java», à la page 441 pour plus de détails) ; ne les utilisez pas dans des programmes entièrement portables.
2. Certaines plateformes ne prennent pas en charge tous les modes de connexion. Sur ces plateformes, vous ne pouvez utiliser que les classes principales et les options liées aux modes pris en charge.

Restrictions et variantes pour les classes de base de IBM MQ classes for Java

Les classes centrales se comportent généralement de manière cohérente dans tous les environnements, même si les appels MQI équivalents présentent normalement des différences d'environnement. Le comportement est comme si un gestionnaire de files d'attente AIX, Linuxou Windows est utilisé, à l'exception des restrictions et variations mineures suivantes.

*Restrictions pour les valeurs MQGMO_ * dans IBM MQ classes for Java*
Certaines valeurs MQGMO_ * ne sont pas prises en charge par tous les gestionnaires de files d'attente.

L'utilisation des valeurs MQGMO_ * suivantes peut entraîner l'émission d'une exception MQException à partir de MQQueue.get():

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_MESSAGE TERMINE
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

De plus, MQGMO_SET_SIGNAL n'est pas pris en charge lorsqu'il est utilisé à partir de Java.

*Restrictions pour les valeurs MQPMRF_ * dans IBM MQ classes for Java*
Ils sont utilisés uniquement lors de l'insertion de messages dans une liste de distribution et ne sont pris en charge que par les gestionnaires de files d'attente prenant en charge les listes de distribution. Par exemple, les gestionnaires de files d'attente z/OS ne prennent pas en charge les listes de distribution.

*Restrictions pour les valeurs MQPMO_ * dans IBM MQ classes for Java*
Certaines valeurs MQPMO_ * ne sont pas prises en charge par tous les gestionnaires de files d'attente

L'utilisation des valeurs MQPMO_ * suivantes peut entraîner l'émission d'une exception MQException à partir de MQQueue.put() ou de MQQueueManager.put ():

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NOUVEAU_ID_MESSAGE
MQPMO_RESOLVE_LOCAL_Q

*Restrictions et variations pour les valeurs MQCNO_ * dans IBM MQ classes for Java*
Certaines valeurs `MQCNO_*` ne sont pas prises en charge.

- La reconnexion automatique du client n'est pas prise en charge par IBM MQ classes for Java. Quelle que soit la valeur `MQCNO_RECONNECT_*` que vous avez définie, la connexion continue de se comporter comme si vous aviez défini `MQCNO_RECONNECT_DISABLED`.
- `MQCNO_FASTPATH` est ignoré sur les gestionnaires de files d'attente qui ne prennent pas en charge `MQCNO_FASTPATH`. Elle est également ignorée par les connexions client.

*Restrictions pour les valeurs MQRO_ * dans IBM MQ classes for Java*
Les options de rapport suivantes peuvent être définies.

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_ET_EXPIRATION

Pour plus d'informations, voir Rapport.

Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms
IBM MQ for z/OS behaves differently from IBM MQ on other platforms in some areas.

BackoutCount
A z/OS queue manager returns a maximum BackoutCount of 255, even if the message has been backed out more than 255 times.

Default dynamic queue prefix
When connected to a z/OS queue manager using a bindings connection, the default dynamic queue prefix is CSQ.*. Otherwise, the default dynamic queue prefix is AMQ.*.

MQQueueManager constructor
Client connect is not supported on z/OS. Attempting to connect with client options results in an MQException with MQCC_FAILED and MQRC_ENVIRONMENT_ERROR.

The MQQueueManager constructor might also fail with MQRC_CHAR_CONVERSION_ERROR (if it fails to initialize conversion between the IBM-1047 and ISO8859-1 code pages), or MQRC_UCS2_CONVERSION_ERROR (if it fails to initialize conversion between the queue manager's code page and Unicode). If your application fails with one of these reason codes, ensure that the National Language Resources component of Language Environment is installed, and ensure that the correct conversion tables are available.

Conversion tables for Unicode are installed as part of the z/OS C/C++ optional feature. See the *z/OS C/C++ Programming Guide*, SC09-4765, for more information about enabling UCS-2 conversions.

Fonctions en dehors des classes de base de IBM MQ classes for Java
Les IBM MQ classes for Java contiennent certaines fonctions spécifiquement conçues pour utiliser des extensions d'API qui ne sont pas prises en charge par tous les gestionnaires de files d'attente. Cette collection de rubriques décrit leur comportement lors de l'utilisation d'un gestionnaire de files d'attente qui ne les prend pas en charge.

Variantes de l'option de constructeur MQQueueManager
Certains des constructeurs MQQueueManager incluent un argument entier facultatif. Certaines valeurs de cet argument ne sont pas acceptées sur toutes les plateformes.

Lorsqu'un constructeur MQQueueManager inclut un argument entier facultatif, il est mappé à la zone d'options MQCNO de l'interface MQI et est utilisé pour passer d'une connexion normale à une connexion Fast Path. Cette forme étendue du constructeur est acceptée dans tous les environnements, si les seules options utilisées sont MQCNO_STANDARD_BINDING ou MQCNO_FASTPATH_BINDING. Toute autre option entraîne l'échec du constructeur avec MQRC_OPTIONS_ERROR. Option de raccourci CMQC.MQCNO_FASTPATH_BINDING est honoré uniquement avec une connexion de liaisons à un gestionnaire de files d'attente qui la prend en charge. Dans d'autres environnements, il est ignoré.

Restrictions sur la méthode MQQueueManager.begin ()
Cette méthode ne peut être utilisée que sur un gestionnaire de files d'attente IBM MQ sur des systèmes AIX, Linux, and Windows en mode liaisons. Sinon, il échoue avec MQRC_ENVIRONMENT_ERROR.

Pour plus d'informations, voir «Coordination JTA/JDBC à l'aide de IBM MQ classes for Java», à la page 411.

Variations dans les zones d'options MQGetMessage
Certains gestionnaires de files d'attente ne prenant pas en charge la structure MQGMO version 2, vous devez définir certaines zones sur leurs valeurs par défaut.

Lorsque vous utilisez un gestionnaire de files d'attente qui ne prend pas en charge la structure MQGMO version 2, conservez les valeurs par défaut des zones suivantes:

GroupStatus
SegmentStatus
Segmentation

En outre, la zone MatchOptions prend en charge uniquement MQMO_MATCH_MSG_ID et MQMO_MATCH_CORREL_ID. Si vous placez des valeurs non prises en charge dans ces zones, la fonction

MQDestination.get() suivante échoue avec MQRC_GMO_ERROR. Si le gestionnaire de files d'attente ne prend pas en charge la structure MQGMO version 2, ces zones ne sont pas mises à jour après la réussite de la commande MQDestination.get().

Restrictions dans les listes de distribution dans IBM MQ classes for Java
Tous les gestionnaires de files d'attente ne vous permettent pas d'ouvrir une MQDistributionList.

Les classes suivantes sont utilisées pour créer des listes de distribution:

MQDistributionList
Elément MQDistributionList
MQMessageTracker

Vous pouvez créer et remplir des éléments MQDistributionLists et MQDistributionListdans n'importe quel environnement, mais tous les gestionnaires de files d'attente ne vous permettent pas d'ouvrir une MQDistributionList. En particulier, les gestionnaires de files d'attente z/OS ne prennent pas en charge les listes de distribution. La tentative d'ouverture d'une liste MQDistributionList lors de l'utilisation d'un gestionnaire de files d'attente de ce type génère une erreur MQRC_OD_ERROR.

Variations dans les zones d'options MQPutMessage
Si un gestionnaire de files d'attente ne prend pas en charge les listes de distribution, certaines zones MQPMO sont traitées différemment.

Quatre zones du MQPMO sont affichées sous la forme des variables de membre suivantes dans la classe MQPutMessageOptions:

KnownDestCount
UnknownDestCount
InvalidDestCount
recordFields

Ces champs sont principalement destinés à être utilisés avec les listes de distribution. Toutefois, un gestionnaire de files d'attente qui prend en charge les listes de distribution remplit également les zones DestCount après une opération MQPUT sur une seule file d'attente. Par exemple, si la file d'attente se résout en file d'attente locale, knownDestCount est défini sur 1 et les deux autres zones de comptage sont définies sur 0.

Si le gestionnaire de files d'attente ne prend pas en charge les listes de distribution, ces valeurs sont simulées comme suit:

- Si la fonction put () aboutit, unknownDestCount est défini sur 1 et les autres sur 0.
- Si la fonction put () échoue, le nombre invalidDestest défini sur 1 et les autres sur 0.

La variable recordFields est utilisée avec les listes de distribution. Une valeur peut être écrite dans recordFields à tout moment, quel que soit l'environnement. Elle est ignorée si l'objet d'options MQPutMessageest utilisé sur un objet MQDestination.put() ou MQQueueManager.put () suivant, plutôt que sur MQDistributionList.put ().

Restrictions dans les zones MQMD avec IBM MQ classes for Java
Certaines zones MQMD concernées par la segmentation de message doivent être laissées à leur valeur par défaut lors de l'utilisation d'un gestionnaire de files d'attente qui ne prend pas en charge la segmentation.

Les zones MQMD suivantes sont principalement concernées par la segmentation de message:

GroupId
MsgSeqNumber
Décalage
MsgFlags
OriginalLength

Si une application définit l'une de ces zones MQMD sur des valeurs autres que leurs valeurs par défaut, puis effectue une opération put () ou get () sur un gestionnaire de files d'attente qui ne les prend pas

en charge, la commande put () ou get () émet une exception MQException avec MQRC_MD_ERROR. Une insertion () ou une extraction () réussie avec un gestionnaire de files d'attente de ce type laisse toujours les zones MQMD définies sur leurs valeurs par défaut. N'envoyez pas de message groupé ou segmenté à une application Java qui s'exécute sur un gestionnaire de files d'attente qui ne prend pas en charge le regroupement et la segmentation de messages.

Si une application Java tente d'extraire () un message d'un gestionnaire de files d'attente qui ne prend pas en charge ces zones et que le message physique à extraire fait partie d'un groupe de messages segmentés (c'est-à-dire qu'il comporte des valeurs autres que celles par défaut pour les zones MQMD), il est extrait sans erreur. Toutefois, les zones MQMD du MQMessage ne sont pas mises à jour, la propriété de format MQMessage est définie sur MQFMT_MD_EXTENSION et les données de message vraies sont préfixées avec une structure MQMDE qui contient les valeurs des nouvelles zones.

Restrictions pour IBM MQ classes for Java sous CICS Transaction Server

Dans l'environnement CICS Transaction Server for z/OS , seule l'unité d'exécution principale (première) est autorisée à émettre des appels CICS ou IBM MQ .

Notez que les classes IBM MQ JMS ne sont pas prises en charge pour une utilisation dans une application CICS Java .

Il n'est donc pas possible de partager des objets MQQueueManager ou MQQueue entre les unités d'exécution de cet environnement, ni de créer un MQQueueManager sur une unité d'exécution enfant.

z/OS «Miscellaneous differences between IBM MQ classes for Java on z/OS and other platforms», à la page 443 identifie certaines restrictions et variantes qui s'appliquent à IBM MQ classes for Java lors de l'exécution sur un gestionnaire de files d'attente z/OS . De plus, lors de l'exécution sous CICS, les méthodes de contrôle des transactions sur MQQueueManager ne sont pas prises en charge. Au lieu d'émettre MQQueueManager.commit () ou MQQueueManager.backout (), les applications utilisent les méthodes de synchronisation de tâche JCICS , Task.commit() et Task.rollback(). La classe Task est fournie par JCICS dans le package com.ibm.cics.server .

Utilisation de l'adaptateur de ressources IBM MQ

L'adaptateur de ressources permet aux applications qui s'exécutent dans un serveur d'applications d'accéder aux ressources IBM MQ . Il prend en charge les communications entrantes et sortantes.

Contenu de l'adaptateur de ressources

Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. Outre l'adaptateur de ressources qui prend en charge Java EE et JMS 2.0, IBM MQ 9.3.0 et les versions ultérieures fournissent un adaptateur de ressources qui prend en charge Jakarta Messaging.

JM 3.0 **Adaptateur de ressources IBM MQ pour Jakarta Messaging**

Jakarta Connectors Architecture fournit un moyen standard de connecter des applications qui s'exécutent dans un environnement Jakarta EE à un système d'information d'entreprise (EIS) tel que IBM MQ ou Db2. L'adaptateur de ressources IBM MQ pour Jakarta Messaging implémente les interfaces Jakarta Connectors 2.0.0 et contient le fichier IBM MQ classes for Jakarta Messaging. Il permet aux applications Jakarta Messaging et aux beans gérés par message (MDB) exécutés sur un serveur d'applications d'accéder aux ressources d'un gestionnaire de files d'attente IBM MQ . L'adaptateur de ressources prend en charge à la fois le domaine point à point et le domaine de publication / abonnement.

JMS 2.0 **Adaptateur de ressources IBM MQ pour JMS 2.0**

Java Platform, Enterprise Edition Connector Architecture (JCA) fournit un moyen standard de connecter les applications qui s'exécutent dans un environnement Java EE à un système d'information d'entreprise (EIS) tel que IBM MQ ou Db2. L'adaptateur de ressources IBM MQ pour JMS 2.0 implémente les interfaces JCA 1.7 et contient le fichier IBM MQ classes for JMS. Il permet aux applications JMS et aux beans gérés par message (MDB) exécutés sur un serveur d'applications

d'accéder aux ressources d'un gestionnaire de files d'attente IBM MQ . L'adaptateur de ressources prend en charge à la fois le domaine point à point et le domaine de publication / abonnement.

L'adaptateur de ressources IBM MQ prend en charge deux types de communication entre une application et un gestionnaire de files d'attente:

Communication sortante
Une application démarre une connexion à un gestionnaire de files d'attente, puis envoie des messages JMS aux destinations JMS et reçoit des messages JMS des destinations JMS de manière synchrone.

Communication entrante
Un message JMS qui arrive à une destination JMS est distribué à un bean géré par message, qui traite le message de manière asynchrone.

L'adaptateur de ressources contient également le IBM MQ classes for Java. Les classes sont automatiquement disponibles pour les applications qui s'exécutent sur un serveur d'applications dans lequel l'adaptateur de ressources a été déployé et permettent aux applications qui s'exécutent sur ce serveur d'applications d'utiliser l'API IBM MQ classes for Java lorsqu'elles accèdent aux ressources d'un gestionnaire de files d'attente IBM MQ .

L'utilisation de IBM MQ classes for Java dans un environnement Java EE est prise en charge avec des restrictions. Pour plus d'informations sur ces restrictions, voir «Exécution d'applications IBM MQ classes for Java dans Java EE», à la page 359.

Version de l'adaptateur de ressources à utiliser

La version de l'adaptateur de ressources que vous utilisez varie selon que vous le déployez sur un serveur d'applications prenant en charge Jakarta EE ou Java EE:

JM 3.0 **Jakarta EE**
Depuis la IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge. L'adaptateur de ressources IBM MQ pour Jakarta Messaging doit être déployé dans un serveur d'applications prenant en charge Jakarta EE.

Java EE 7
L'adaptateur de ressources IBM MQ 8.0 et versions ultérieures prend en charge JCA v1.7 et fournit la prise en charge de JMS 2.0 . Cet adaptateur de ressources doit être déployé dans un serveur d'applications Java EE 7 et ultérieur (voir «Déclaration de prise en charge de l'adaptateur de ressources IBM MQ», à la page 447).

Vous pouvez installer l'adaptateur de ressources IBM MQ 8.0 ou version ultérieure sur tout serveur d'applications certifié conforme à la spécification Java Platform, Enterprise Edition 7 . A l'aide de l'adaptateur de ressources IBM MQ 8.0 ou d'une version ultérieure, une application peut se connecter à un gestionnaire de files d'attente à l'aide du transport BINDINGS ou CLIENT.

Important : L'adaptateur de ressources IBM MQ 8.0 ou version ultérieure peut être déployé uniquement dans un serveur d'applications prenant en charge JMS 2.0.

Utilisation de l'adaptateur de ressources avec WebSphere Application Server traditional

L'adaptateur de ressources IBM MQ est préinstallé dans WebSphere Application Server traditional 9.0 ou version ultérieure. Par conséquent, il n'est pas nécessaire d'installer un nouvel adaptateur de ressources.

JM 3.0 WebSphere Application Server traditional ne prend actuellement pas en charge Jakarta EE. Voir la déclaration de prise en charge de l'adaptateur de ressources IBM MQ.

Remarque : Un adaptateur de ressources IBM MQ 9.0 ou version ultérieure peut se connecter en mode transport CLIENT ou BINDINGS à n'importe quel gestionnaire de files d'attente IBM MQ en service.

Utilisation de l'adaptateur de ressources avec WebSphere Liberty

Pour vous connecter à IBM MQ à partir de WebSphere Liberty, vous devez utiliser l'adaptateur de ressources IBM MQ . Etant donné que Liberty ne contient pas l'adaptateur de ressources IBM MQ , vous devez l'obtenir séparément de Fix Central.

La version de l'adaptateur de ressources que vous utilisez varie selon que vous le déployez dans une version de Liberty prenant en charge Jakarta EE ou Java EE.

Pour plus d'informations sur le téléchargement et l'installation de l'adaptateur de ressources, voir «Installation de l'adaptateur de ressources dans Liberty», à la page 455.

Concepts associés

«Configuration de l'adaptateur de ressources pour les communications entrantes», à la page 463
Pour configurer les communications entrantes, définissez les propriétés d'un ou plusieurs objets ActivationSpec.

«Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 482
Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

«Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85
IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie Java fournis avec IBM MQ. Outre l'implémentation des interfaces définies dans les spécifications JMS et Jakarta Messaging , ces fournisseurs de messagerie ajoutent deux ensembles d'extensions à l'API de messagerie Java .

«Utilisation IBM MQ classes for Java», à la page 356
Utilisez IBM MQ dans un environnement Java . IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

Référence associée

Configuration du serveur d'applications pour utiliser le dernier niveau de maintenance de l'adaptateur de ressources

Identification des incidents pour l'adaptateur de ressources IBM MQ

WebSphere Application Server rubriques

Gestion de l'adaptateur de ressources IBM MQ

Déploiement d'applications JMS dans Liberty pour utiliser le fournisseur de messagerie IBM MQ

Déclaration de prise en charge de l'adaptateur de ressources IBM MQ

L'adaptateur de ressources IBM MQ que vous devez utiliser pour la communication entre une application et un gestionnaire de files d'attente varie selon que vous utilisez l'API Jakarta Messaging 3.0 ou l'API JMS 2.0 .

JMS 2.0 IBM MQ 8.0 ou version ultérieure est fourni avec un adaptateur de ressources qui implémente la spécification JMS 2.0 . Il ne peut être déployé que sur un serveur d'applications compatible avec Java Platform, Enterprise Edition 7 (Java EE 7) et qui prend donc en charge JMS 2.0. La liste des serveurs d'applications certifiés pour Java Platform, Enterprise Edition est disponible sur le site Web d' Oracle.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. Outre l'adaptateur de ressources qui prend en charge Java EE et JMS 2.0, IBM MQ 9.3.0 et les versions ultérieures fournissent un adaptateur de ressources qui prend en charge Jakarta Messaging. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Using IBM MQ classes for JMS.

Déploiement dans WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 et versions ultérieures, et WebSphere Application Server Liberty 9.0 et versions ultérieures, sont des serveurs d'applications certifiés Java EE 7 pour que l'adaptateur de ressources IBM MQ 9.0 puisse y être déployé.

Pour utiliser l'adaptateur de ressources IBM MQ pour Jakarta Messaging avec Liberty, vous devez utiliser une version de Liberty qui prend en charge Jakarta EE.

WebSphere Liberty dispose des fonctions suivantes pour utiliser les adaptateurs de ressources:

- JM 3.0 La fonction messaging-3.0 permet d'utiliser les adaptateurs de ressources Jakarta Messaging 3.0 .
- La fonction wmqJmsClient-2.0 permet d'utiliser les adaptateurs de ressources JMS 2.0 .
- La fonction wmqJmsClient-1.1 permet d'utiliser les adaptateurs de ressources JMS 1.1 .

Important :

- JM 3.0 L'adaptateur de ressources IBM MQ pour Jakarta Messaging doit être déployé dans une version de Liberty qui prend en charge Jakarta EE. Cet adaptateur de ressources ne peut pas être utilisé avec les versions de Liberty qui prennent en charge l'ancienne spécification Java EE et non Jakarta EE.
- JMS 2.0 L'adaptateur de ressources IBM MQ 8.0 ou version ultérieure qui prend en charge JMS 2.0 doit être déployé avec la fonction wmqJmsClient-2.0 .

Déploiement dans WebSphere Application Server traditional

WebSphere Application Server traditional 9.0 est fourni avec un adaptateur de ressources IBM MQ 9.0 déjà installé. Par conséquent, il n'est pas nécessaire d'installer un nouvel adaptateur de ressources. L'adaptateur de ressources installé peut se connecter en mode transport CLIENT ou BINDINGS à tous les gestionnaires de files d'attente qui s'exécutent sur une version prise en charge de IBM MQ. Pour plus d'informations, voir «Connectivité aux gestionnaires de files d'attente IBM MQ 8.0 ou version ultérieure», à la page 449.

Important :

- L'adaptateur de ressources IBM MQ 9.0 ne peut pas être déployé dans les versions d' WebSphere Application Server traditional antérieures à IBM MQ 9.0, car ces versions ne sont pas certifiées Java EE 7 .
- JM 3.0 WebSphere Application Server traditional ne prend actuellement pas en charge Jakarta EE.

Pour plus d'informations sur les versions de l'adaptateur de ressources fournies avec WebSphere Application Server, voir la note technique Quelle version de l'adaptateur de ressources WebSphere MQ (RA) est fournie avec WebSphere Application Server?

Utilisation de l'adaptateur de ressources avec d'autres serveurs d'applications

Pour tous les autres serveurs d'applications compatibles avec Java EE 7 ou Jakarta EE , les problèmes qui se produisent après la réussite du test de vérification de l'installation (IVT) de l'adaptateur de ressources IBM MQ peuvent être signalés à IBM pour l'examen de la trace du produit IBM MQ et d'autres informations de diagnostic IBM MQ . Si l'IVT de l'adaptateur de ressources IBM MQ ne peut pas être exécuté correctement, les problèmes rencontrés sont probablement dus à un déploiement incorrect ou à des définitions de ressources incorrectes spécifiques au serveur d'applications et les problèmes doivent être examinés à l'aide de la documentation du serveur d'applications et de l'organisation de support de ce serveur d'applications.

Java Temps d'exécution

L'environnement d'exécution Java (JRE) utilisé pour exécuter le serveur d'applications doit être pris en charge avec le client IBM MQ 9.0 ou version ultérieure. Pour plus d'informations, voir Configuration système requise pour IBM MQ. (Sélectionnez la version et le système d'exploitation ou le rapport de composant que vous souhaitez voir, puis suivez le lien **Java** répertorié sous l'onglet **Supported Software** .)

Connectivité aux gestionnaires de files d'attente IBM MQ 8.0 ou version ultérieure

La gamme complète de fonctionnalités JMS 2.0 est disponible lors de la connexion à un gestionnaire de files d'attente IBM MQ 8.0 ou version ultérieure à l'aide de l'adaptateur de ressources qui a été déployé sur un serveur d'applications certifié Java EE 7 . Pour utiliser la fonctionnalité JMS 2.0 , l'adaptateur de ressources doit se connecter au gestionnaire de files d'attente à l'aide du mode normal du fournisseur de messagerie IBM MQ . Pour plus d'informations, voir Configuration de la propriété JMS **PROVIDERVERSION**.

JM 3.0 La gamme complète de fonctionnalités Jakarta Messaging 3.0 est disponible lors de la connexion à un gestionnaire de files d'attente IBM MQ 9.3 ou version ultérieure à l'aide de l'adaptateur de ressources qui a été déployé sur un serveur d'applications certifié Jakarta EE .

Extensions MQ

La spécification JMS 2.0 introduit des modifications dans le fonctionnement de certains comportements. Etant donné que IBM MQ 8.0 ou une version ultérieure implémente cette spécification, il y a des changements de comportement entre IBM MQ 8.0 et les versions ultérieures et les versions antérieures du produit. Dans IBM MQ 8.0 ou version ultérieure, IBM MQ classes for JMS inclut la prise en charge de la propriété système Java `com.ibm.mq.jms.SupportMQExtensions` qui, lorsqu'elle est définie sur TRUE, permet à ces versions de IBM MQ de rétablir ces comportements à ceux de IBM WebSphere MQ 7.5 ou version antérieure. La valeur par défaut de la propriété est `false`.

L'adaptateur de ressources IBM MQ 9.0 ou version ultérieure inclut également une propriété d'adaptateur de ressources appelée `supportMQExtensions` qui a le même effet et la même valeur par défaut que la propriété système `com.ibm.mq.jms.SupportMQExtensions` Java . Par défaut, cette propriété d'adaptateur de ressources est définie sur false dans le fichier `ra.xml` .

Si la propriété d'adaptateur de ressources et la propriété système Java sont définies, la propriété système est prioritaire.

Notez que dans l'adaptateur de ressources déjà déployé dans WebSphere Application Server traditional 9.0, cette propriété est automatiquement définie sur TRUE pour faciliter la migration.

Pour plus d'informations, voir «Propriété SupportMQExtensions», à la page 336.

Problèmes généraux

L'entrelacement de session n'est pas pris en charge

Certains serveurs d'applications fournissent une fonction appelée entrelacement de session, dans laquelle la même session JMS peut être utilisée dans plusieurs transactions, bien qu'elle ne soit répertoriée que dans une seule session à la fois. L'adaptateur de ressources IBM MQ ne prend pas en charge cette fonction, ce qui peut entraîner les problèmes suivants:

```
Une tentative d'insertion d'un message dans une file d'attente MQ a échoué avec le code
anomalie 2072 (MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Les appels à xa_close () échouent avec le code anomalie -3 (XAER_PROTO) et un FDC avec l'ID sonde AT040010 est généré sur le gestionnaire de files d'attente IBM MQ accessible à partir du serveur d'applications. Pour plus d'informations sur la désactivation de cette fonction, consultez la documentation de votre serveur d'applications.

Spécification Java Transaction API (JTA) sur la façon dont les ressources XA sont récupérées pour la récupération des transactions XA

La section 3.4.8 de la spécification JTA ne définit pas de mécanisme spécifique par lequel les ressources XA sont recréées pour effectuer une reprise transactionnelle XA. Il appartient donc à chaque gestionnaire de transactions individuel (et, par conséquent, au serveur d'applications) de savoir comment les ressources XA impliquées dans une transaction XA sont récupérées. Il est possible que, pour certains serveurs d'applications, l'adaptateur de ressources IBM MQ 9.0 n'implémente pas les mécanismes spécifiques au serveur d'applications qui sont utilisés pour effectuer une reprise transactionnelle XA.

Mise en correspondance des connexions dans une fabrique ManagedConnection

Un serveur d'applications peut appeler la méthode matchManagedConnections sur une instance de fabrique ManagedConnectionfournie par l'adaptateur de ressources IBM MQ . Une ManagedConnection est renvoyée uniquement si la méthode en trouve une qui correspond aux arguments **`javax.security.auth.Subject`** et **`javax.resource.spi.ConnectionRequestInfo`** transmis à la méthode par le serveur d'applications.

Limitations de l'adaptateur de ressources IBM MQ

L'adaptateur de ressources IBM MQ est pris en charge sur toutes les plateformes IBM MQ . Toutefois, lorsque vous utilisez l'adaptateur de ressources IBM MQ , certaines fonctions de IBM MQ ne sont pas disponibles ou sont limitées.

L'adaptateur de ressources IBM MQ présente les limitations suivantes:

- Depuis la IBM MQ 8.0, l'adaptateur de ressources est un adaptateur de ressources Java Platform, Enterprise Edition 7 (Java EE 7) qui fournit la fonction JMS 2.0 . Par conséquent, l'adaptateur de ressources IBM MQ 8.0 ou version ultérieure doit être installé sur un serveur d'applications certifié Java EE 7 ou version ultérieure. Il peut se connecter en mode de transport client ou liaisons à n'importe quel gestionnaire de files d'attente en service.
- Lors de l'exécution dans le serveur d'applications WebSphere Liberty , les IBM MQ classes for Java stabilisées ne sont pas prises en charge. Dans d'autres serveurs d'applications, l'utilisation de IBM MQ classes for Java n'est pas recommandée. Voir la note technique IBM Using WebSphere MQ Java Interfaces in J2EE/JEE Environments pour plus de détails sur les remarques relatives à IBM MQ classes for Java dans Java EE.
- Lors de l'exécution dans le serveur d'applications WebSphere Liberty sous z/OS, la fonction wmqJmsClient-2.0 doit être utilisée. La prise en charge de JCA générique n'est pas possible pour z/OS.
- L'adaptateur de ressources IBM MQ ne prend pas en charge les programmes d'exit de canal écrits dans des langages autres que Java.
- Lorsqu'un serveur d'applications est en cours d'exécution, la valeur de la propriété sslFipsRequired doit être true pour toutes les ressources JCA ou false pour toutes les ressources JCA . Il s'agit d'une exigence même si les ressources JCA ne sont pas utilisées simultanément. Si la propriété sslFipsRequired a des valeurs différentes pour différentes ressources JCA , IBM MQ émet le code anomalie MQRC_UNSUPPORTED_CIPHER_SUITE, même si une connexion TLS n'est pas utilisée.
- Vous ne pouvez pas spécifier plusieurs magasins de clés pour un serveur d'applications. Si des connexions sont établies à plusieurs gestionnaires de files d'attente, toutes les connexions doivent utiliser le même magasin de clés. Cette limitation ne s'applique pas à WebSphere Application Server.
- Si vous utilisez une table de définition de canal du client (CCDT) avec plusieurs définitions de canal de connexion client appropriées, en cas d'incident, l'adaptateur de ressources peut sélectionner une autre définition de canal et donc un gestionnaire de files d'attente différent de la CCDT, ce qui entraînerait des problèmes pour la reprise des transactions. L'adaptateur de ressources ne prend aucune mesure pour empêcher l'utilisation d'une telle configuration et il est de votre responsabilité d'éviter les configurations qui peuvent entraîner des problèmes pour la reprise des transactions.
- La fonctionnalité de relance de connexion n'est pas prise en charge pour les connexions sortantes lors de l'exécution dans un conteneur Java EE (EJB/Servlet). La nouvelle tentative de connexion n'est pas du tout prise en charge pour les connexions sortantes JMS lorsque l'adaptateur est utilisé dans un

contexte de conteneur JEE , quelle que soit la configuration de la transaction ou pour une utilisation non transactionnelle.

- La réauthentification, telle que définie dans la section 9.1.9 de la spécification Java EE Connector Architecture version 1.7 , des connexions JMS n'est pas prise en charge. La propriété appelée **reauthentication-support** doit être définie sur la valeur `false`dans le fichier `ra.xml` de l'évaluateur de ressources IBM MQ . Si le serveur d'applications tente de réauthentifier une connexion JMS , l'adaptateur de ressources IBM MQ émet une exception javax.resource.spi.SecurityException avec le code de message MQJCA1028 .

Tâches associées

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client

Référence associée

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

WebSphere Application Server et l'adaptateur de ressources IBM MQ

L'adaptateur de ressources IBM MQ est utilisé par les applications qui exécutent la messagerie JMS avec le fournisseur de messagerie IBM MQ dans WebSphere Application Server.

Important : N'utilisez pas l'adaptateur de ressources IBM MQ avec WebSphere Application Server 6.0 ou WebSphere Application Server 6.1.

WebSphere Application Server traditional 9.0 inclut une version de l'adaptateur de ressources IBM MQ 9.0 . L'adaptateur de ressources IBM MQ 9.0 ou version ultérieure ne peut pas être déployé dans des versions antérieures d' WebSphere Application Server, car ces versions ne sont pas certifiées Java EE 7 .

JM 3.0 WebSphere Application Server traditional ne prend actuellement pas en charge Jakarta EE. Voir la déclaration de prise en charge de l'adaptateur de ressources IBM MQ.

Si vous souhaitez utiliser une application JMS pour accéder aux ressources d'un gestionnaire de files d'attente IBM MQ depuis WebSphere Application Server, utilisez le fournisseur de messagerie IBM MQ dans WebSphere Application Server. Le fournisseur de messagerie IBM MQ contient une version de IBM MQ classes for JMS. Pour plus d'informations, voir la note technique Quelle version de WebSphere MQ Resource Adapter (RA) est fournie avec WebSphere Application Server?.

Important : N'incluez aucun des fichiers JAR IBM MQ classes for JMS ou IBM MQ classes for Java dans votre application. Cela peut entraîner des exceptions ClassCastet peut être difficile à gérer.

Liberty et l'adaptateur de ressources IBM MQ

L'adaptateur de ressources IBM MQ peut être installé dans WebSphere Liberty à l'aide d'une fonction Liberty . La fonction que vous utilisez dépend de la version de l'adaptateur de ressources que vous installez. Vous pouvez également, sous réserve de certaines restrictions, installer l'adaptateur de ressources à l'aide du support Java Platform, Enterprise Edition Connector Architecture (Java EE JCA) générique.

Restrictions générales lors de l'installation de l'adaptateur de ressources dans Liberty

Les restrictions suivantes s'appliquent à l'adaptateur de ressources lors de l'utilisation de la fonction wmqJmsClient-1.1 ou wmqJmsClient-2.0 et lors de l'utilisation du support JCA générique:

- Les IBM MQ classes for Java ne sont pas pris en charge dans Liberty. Ils ne doivent pas être utilisés avec la fonction de messagerie IBM MQ Liberty ni avec la prise en charge JCA générique. Pour plus d'informations, voir Utilisation des interfaces WebSphere MQ Java dans les environnements J2EE/JEE.
- Le type de transport de l'adaptateur de ressources IBM MQ est BINDINGS_THEN_CLIENT. Ce type de transport n'est pas pris en charge dans la fonction de messagerie IBM MQ Liberty .

- Avant IBM MQ 9.0, la fonction Advanced Message Security (AMS) n'était pas incluse dans la fonction de messagerie IBM MQ Liberty . Toutefois, AMS est pris en charge avec un adaptateur de ressources IBM MQ 9.0 ou version ultérieure.

Remarque : Sur les versions IBM MQ ultérieures à IBM MQ 9.0.0.6 et IBM MQ 9.1.0.1 , vous devez utiliser la fonction transportSecurity-1.0 au lieu de la fonction ssl-1.0 .

Pour plus d'informations, voir :

Activation de la communication SSL dans Liberty
Valeurs SSL par défaut dans Liberty
Transport Security 1.0

Restrictions relatives à l'utilisation des fonctions Liberty

Avec WebSphere Liberty 8.5.5 Fix Pack 2 to WebSphere Liberty 8.5.5 Fix Pack 5 inclus, seule la fonction wmqJmsClient-1.1 était disponible et seul JMS 1.1 pouvait être utilisé. WebSphere Liberty 8.5.5 Fix Pack 6 a ajouté la fonction wmqJmsClient-2.0 pour que JMS 2.0 puisse être utilisé.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge. Pour utiliser l'adaptateur de ressources IBM MQ pour Jakarta Messaging avec Liberty, vous devez utiliser une version de Liberty qui prend en charge Jakarta EE. Vous devez utiliser l'adaptateur de ressources pour Jakarta Messaging avec la fonction Liberty generic messaging-3.0 .

La fonction que vous devez utiliser dépend de la version de l'adaptateur de ressources que vous utilisez:

- L'adaptateur de ressources IBM MQ 8.0 IBM MQ 8.0.0 Fix Pack 3 et versions ultérieures ne peut être utilisé qu'avec la fonction wmqJmsClient-2.0 .
- L'adaptateur de ressources IBM MQ 9.0 ne peut être utilisé qu'avec la fonction wmqJmsClient-2.0 .
- JM 3.0 La fonction messaging-3.0 permet d'utiliser les adaptateurs de ressources Jakarta Messaging 3.0 .

Restrictions liées à l'utilisation de la prise en charge JCA générique

Si vous utilisez la prise en charge JCA générique, les restrictions suivantes s'appliquent:

- Vous devez spécifier le niveau de JMS lorsque vous utilisez le support JCA générique. JMS 2.0 et JCA 1.7 doivent être utilisés uniquement avec les adaptateurs de ressources IBM MQ 8.0 IBM MQ 8.0.0 Fix Pack 3 et versions ultérieures.
- Il n'est pas possible d'exécuter l'adaptateur de ressources IBM MQ sur z/OS à l'aide de la prise en charge JCA générique. Pour exécuter l'adaptateur de ressources IBM MQ sous z/OS, il doit être exécuté avec la fonction wmqJmsClient-1.1 ou wmqJmsClient-2.0 .
- L'emplacement de l'adaptateur de ressources est spécifié à l'aide de l'élément xml suivant:

JM 3.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/
wmq.jakarta.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

JMS 2.0
```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Important : La valeur de la balise ID peut être n'importe quelle valeur EXCEPT pour `wmqJms`. Si vous utilisez `wmqJms` comme ID, Liberty ne peut pas charger correctement l'adaptateur de ressources. En effet, `wmqJms` est l'ID utilisé en interne pour faire référence à la fonction spécifique de IBM MQ. Il crée en fait une exception NullPointer.

Les exemples suivants illustrent des fragments provenant d'un fichier `server.xml` lors de l'exécution de JMS 2.0:

```
<!-- Enable features -->
  <featureManager>
    <feature>servlet-3.1</feature>
    <feature>jndi-1.0</feature>
    <feature>jca-1.7</feature>
    <feature>jms-2.0</feature>
  </featureManager>
```

Conseil : Notez l'utilisation des fonctions jca-1.7 et jms-2.0 et l'absence de la fonction wmqJmsClient-2.0 .

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Conseil : Notez l'utilisation de `mqJms` pour l'ID, ce qui est préférable. N'utilisez pas `wmqJms`.

```
<application id="WMQHTTP" location="${server.config.dir}/apps/WMQHTTP.war"
name="WMQHTTP" type="war">
    <classloader apiTypeVisibility="spec, ibm-api, api, third-party"
classProviderRef="mqJms"/>
</application>
```

Conseil : Notez la référence classloaderProviderà l'adaptateur de ressources via l'ID `mqJms`; il s'agit d'autoriser le chargement de classes spécifiques à IBM MQ.

Restrictions lors de la fonction de trace utilisant la prise en charge JCA générique

La fonction de trace et la consignation ne sont pas intégrées au système de trace Liberty . A la place, la trace de l'adaptateur de ressources IBM MQ doit être activée à l'aide des propriétés système Java ou d'un fichier de configuration IBM MQ classes for JMS , comme décrit dans Traçage des applications IBM MQ classes for JMS. Pour plus d'informations sur la définition des propriétés système Java dans Liberty, voir la documentationWebSphere Liberty.

Par exemple, pour activer la trace de l'adaptateur de ressources IBM MQ dans Liberty 19.0.0.9, ajoutez une entrée au fichier Liberty `jvm.options`:

1. Créez un fichier texte nommé `jvm.options`.
2. Insérez les options JVM suivantes pour activer le traçage, une par ligne, dans ce fichier:

   ```
   -Dcom.ibm.msg.client.commonservices.trace.status=ON
   -Dcom.ibm.msg.client.commonservices.trace.outputName=C:\Trace\MQRA-WLP_%PID%.trc
   ```

3. Pour appliquer ces paramètres à un serveur unique, sauvegardez `jvm.options` dans:

   ```
   ${server.config.dir}/jvm.options
   ```

   Pour appliquer ces modifications à tous les Liberty, sauvegardez `jvm.options` dans:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Cette opération s'applique à toutes les machines virtuelles Java qui n'ont pas de fichier `jvm.options` défini en local.
4. Redémarrez le serveur pour activer les modifications.

La trace est alors écrite dans un fichier de trace appelé `MQRA-WLP_<process identifier>.trc` dans le répertoire `<path_to_trace_to>`.

Prise en charge complète de Liberty XA avec les tables de définition de canal du client

Lorsque vous utilisez WebSphere Liberty 18.0.0.2 ou version ultérieure, vous pouvez utiliser des groupes de gestionnaires de files d'attente au sein de la table de définition de canal du client (CCDT) avec des transactions XA. Cela signifie qu'il est désormais possible d'utiliser la disponibilité et la distribution de la charge de travail fournies par les groupes de gestionnaires de files d'attente tout en assurant l'intégrité des transactions.

En cas d'erreurs de connectivité à un gestionnaire de files d'attente, le gestionnaire de files d'attente doit être à nouveau disponible pour que la transaction puisse être résolue. La récupération des transactions est gérée par Libertyet vous devrez peut-être configurer le gestionnaire de transactions de sorte qu'une période de temps appropriée soit autorisée pour que les gestionnaires de files d'attente redeviennent disponibles. Pour plus d'informations, voir Gestionnaire de transactions (transaction) dans la documentation du produit WebSphere Liberty .

Il s'agit d'une fonction côté client, c'est-à-dire que vous avez besoin d'un adaptateur de ressources et non d'un gestionnaire de files d'attente.

Installation de l'adaptateur de ressources IBM MQ

L'adaptateur de ressources IBM MQ est fourni en tant que fichier d'archive de ressources (RAR). Installez le fichier RAR dans votre serveur d'applications. Vous devrez peut-être ajouter des répertoires au chemin du système.

Pourquoi et quand exécuter cette tâche

L'adaptateur de ressources IBM MQ est fourni en tant que fichier d'archive de ressources (RAR):

- JM 3.0 Pour Jakarta Messaging 3.0, ce fichier est appelé `wmq.jakarta.jmsra.rar`. Le fichier RAR contient IBM MQ classes for Jakarta Messaging et l'implémentation IBM MQ des interfaces Jakarta Connectors Architecture (JCA).
- JMS 2.0 Pour JMS 2.0, ce fichier est appelé `wmq.jmsra.rar`. Le fichier RAR contient IBM MQ classes for JMS et l'implémentation IBM MQ des interfaces Java EE Connector Architecture (JCA).

Lorsque vous installez l'adaptateur de ressources dans le cadre de l'installation du produit IBM MQ , le fichier RAR est installé avec IBM MQ classes for JMS dans le répertoire indiqué dans Tableau 61, à la page 454.

Tableau 61. Répertoire IBM MQ contenant le fichier RAR pour chaque plateforme

Plateforme	**Répertoire**
AIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Vous devez utiliser l'adaptateur de ressources IBM MQ pour vous connecter à IBM MQ à partir d'un serveur d'applications. Selon le serveur d'applications que vous utilisez, l'adaptateur de ressources peut être préinstallé ou vous devrez peut-être l'installer vous-même.

Tableau 62. Installation de l'adaptateur de ressources dans un serveur d'applications	
Serveur d'applications	**Préinstallé ou à installer?**
WebSphere Application Server traditional 9.0	L'adaptateur de ressources IBM MQ 9.0 est pré-installé dans WebSphere Application Server traditional 9.0. Par conséquent, vous n'avez pas besoin d'installer un nouvel adaptateur de ressources dans WebSphere Application Server traditional 9.0.
WebSphere Liberty	WebSphere Liberty ne contenant pas l'adaptateur de ressources IBM MQ , vous devez l'obtenir séparément de Fix Central.
Autre serveur d'applications Java EE ou Jakarta EE	Procurez-vous l'adaptateur de ressources séparément de Fix Central, comme pour WebSphere Liberty.

Procédure

- Si vous vous connectez à IBM MQ à partir de WebSphere Libertyou d'un autre serveur d'applications Java EE ou Jakarta EE , téléchargez et installez l'adaptateur de ressources IBM MQ comme décrit dans «Installation de l'adaptateur de ressources dans Liberty», à la page 455.
- Linux AIX

  Pour les connexions de liaisons sur les systèmes AIX and Linux , vérifiez que le répertoire contenant les bibliothèques JNI (Java Native Interface) se trouve dans le chemin système.

  Pour connaître l'emplacement de ce répertoire, qui contient également les bibliothèques IBM MQ classes for JMS , voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 100.

  Windows Sous Windows, ce répertoire est automatiquement ajouté au chemin du système lors de l'installation de IBM MQ classes for JMS.

  Conseil : Comme alternative à la définition du chemin système, l'adaptateur de ressources IBM MQ possède une propriété appelée nativeLibraryPath qui peut être utilisée pour spécifier l'emplacement de la bibliothèque JNI. Par exemple, dans WebSphere Liberty , cela serait configuré comme illustré dans l'exemple suivant:

  ```
  <wmqJmsClient nativeLibraryPath="/opt/mqm/java/lib64"/>
  ```

  Les transactions sont prises en charge à la fois en mode client et en mode liaisons.

Installation de l'adaptateur de ressources dans Liberty

Pour vous connecter à IBM MQ à partir de WebSphere Libertyou d'autres serveurs d'applications Java EE ou Jakarta EE , vous devez utiliser l'adaptateur de ressources IBM MQ . Etant donné que Liberty ne contient pas l'adaptateur de ressources IBM MQ , vous devez l'obtenir séparément de Fix Central.

Avant de commencer

Remarque : Les informations de cette rubrique ne s'appliquent pas à WebSphere Application Server traditional 9.0. L'adaptateur de ressources IBM MQ 9.0 est préinstallé dans WebSphere Application Server traditional 9.0. Par conséquent, il n'est pas nécessaire d'installer un nouvel adaptateur de ressources dans ce cas.

Avant de commencer cette tâche, assurez-vous qu'un environnement d'exécution Java (JRE) Java runtime environment est installé sur votre machine et que ce dernier a été ajouté au chemin du système.

Le programme d'installation de Java utilisé dans ce processus d'installation ne nécessite pas d'exécution en tant que superutilisateur ou utilisateur spécifique. La seule condition requise est que l'utilisateur dans lequel il est exécuté ait accès en écriture au répertoire dans lequel vous souhaitez placer les fichiers.

Pour les versions Liberty jusqu'à WebSphere Liberty 8.5.5 Fix Pack 1, si un EJB est déployé en utilisant uniquement la configuration dans `ejb-jar.xml`, l'APAR PM89890 doit être appliqué à la version de WebSphere Application Server utilisée par le profil Liberty . Cette méthode de configuration est utilisée pour le programme de vérification de l'installation (IVT) de l'adaptateur de ressources. Par conséquent, cet APAR est requis pour que le IVT s'exécute.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge. Pour utiliser l'adaptateur de ressources IBM MQ pour Jakarta Messaging avec Liberty, vous devez utiliser une version de Liberty qui prend en charge Jakarta EE. Par exemple, vous pouvez utiliser la fonction Liberty generic messaging-3.0 .

Pourquoi et quand exécuter cette tâche

Le fichier JAR de l'adaptateur de ressources que vous pouvez télécharger depuis Fix Central est exécutable. Lorsque vous exécutez ce fichier exécutable, il affiche le contrat de licence IBM MQ , qui doit être accepté. Il demande un répertoire dans lequel installer l'adaptateur de ressources IBM MQ . Le fichier RAR de l'adaptateur de ressources et le programme de test de vérification de l'installation (IVT) sont ensuite installés dans ce répertoire. Vous pouvez accepter la valeur par défaut ou spécifier un autre répertoire, qui peut être le répertoire des adaptateurs de ressources d'un serveur d'applications ou tout autre répertoire de votre système. Le répertoire est créé dans le cadre de l'installation s'il n'existe pas.

Avant IBM MQ 9.0, le nom du fichier à télécharger était au format *V.R.M.F*`-WS-MQ-Java-InstallRA.jar`, par exemple `8.0.0.6-WS-MQ-Java-InstallRA.jar`. Depuis la IBM MQ 9.0, le format du nom de fichier est *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar`, par exemple `9.0.0.0-IBM-MQ-Java-InstallRA.jar`.

Une fois que vous avez téléchargé et installé l'adaptateur de ressources, vous êtes prêt à le configurer dans WebSphere Liberty.

Procédure

1. Téléchargez l'adaptateur de ressources IBM MQ à partir de Fix Central.
   a) Cliquez sur ce lien: IBM MQ Adaptateur de ressources.
   b) Recherchez l'adaptateur de ressources correspondant à votre version d' IBM MQ dans la liste affichée des correctifs disponibles.
   Exemple :

   ```
   release level: 9.1.4.0-IBM-MQ-Java-InstallRA
   Continuous Delivery Release: 9.1.4 IBM MQ Resource Adapter for use with Application
   Servers
   ```

   Cliquez ensuite sur le nom du fichier de l'adaptateur de ressources et suivez le processus de téléchargement.
2. Démarrez l'installation en entrant la commande suivante à partir du répertoire dans lequel vous avez téléchargé le fichier.
   Depuis la IBM MQ 9.0, le format de la commande est le suivant:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   où *V.R.M.F* est le numéro de version, d'édition, de modification et de groupe de correctifs et *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar` est le nom du fichier qui a été téléchargé à partir de Fix Central.

Par exemple, pour installer l'adaptateur de ressources IBM MQ pour l'édition IBM MQ 9.1.4 , utilisez la commande suivante:

```
java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
```

Remarque : Pour effectuer cette installation, vous devez disposer d'un environnement d'exécution Java (JRE) installé sur votre machine et ajouté au chemin système.

Lorsque vous entrez la commande, les informations suivantes s'affichent:

```
Avant de pouvoir utiliser, extraire ou installer IBM MQ 9.1, vous devez accepter
les termes de 1. IBM International License Agreement for Evaluation of
Programmes 2. IBM -Conditions Internationales d'Utilisation de Logiciels IBM et autres
license information. Veuillez lire attentivement les contrats de licence suivants.

The license agreement is separately viewable using the
--viewLicenseAgreement option.
Press Enter to display the license terms now, or 'x' to skip.
```

3. Lisez et acceptez les dispositions du contrat de licence:
   a) Pour afficher la licence, appuyez sur Entrée.

      Vous pouvez également appuyer sur x pour ignorer l'affichage de la licence.

      Après l'affichage de la licence ou immédiatement après la sélection de x, le message suivant s'affiche pour vous indiquer que vous pouvez choisir d'afficher des termes de licence supplémentaires:

      ```
      Des informations supplémentaires sur les licences peuvent être visualisées séparément à
      l'aide du
      -- Option d'informationviewLicense.
      Appuyez sur Entrée pour afficher des informations supplémentaires sur la licence ou
      sur'x'pour les ignorer.
      ```

   b) Pour afficher les dispositions de licence supplémentaires, appuyez sur Entrée.

      Vous pouvez également cliquer sur x pour ignorer l'affichage des dispositions de licence supplémentaires.

      Après l'affichage des dispositions de la licence supplémentaire ou immédiatement après la sélection de x, le message suivant s'affiche pour vous demander d'accepter le contrat de licence:

      ```
      By choosing the "I Agree" option below, you agree to the terms of the
      license agreement and non-IBM terms, if applicable. Si vous ne
      agree, select "I do not Agree".

      Select [1] I Agree, or [2] I do not Agree:
      ```

   c) Pour accepter le contrat de licence et continuer à sélectionner le répertoire d'installation, sélectionnez 1.

      Sinon, si vous sélectionnez 2, l'installation s'arrête immédiatement.

      Si vous avez sélectionné 1, le message suivant s'affiche, vous demandant de sélectionner un répertoire d'installation cible:

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

4. Indiquez le répertoire d'installation de l'adaptateur de ressources:
   - Si vous souhaitez installer l'adaptateur de ressources à l'emplacement par défaut, appuyez sur Entrée sans spécifier de valeur.
   - Si vous souhaitez installer l'adaptateur de ressources à un emplacement différent de l'emplacement par défaut, indiquez le nom du répertoire dans lequel vous souhaitez installer l'adaptateur de ressources, puis appuyez sur Entrée.

   Une fois les fichiers installés à l'emplacement sélectionné, un message de confirmation s'affiche, comme illustré dans l'exemple suivant:

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

   Lors de l'installation, un nouveau répertoire nommé `wmq` est créé dans le répertoire d'installation sélectionné, puis les fichiers suivants sont installés dans le répertoire `wmq` :

- Le programme de test de vérification de l'installation, `wmq.jakarta.jmsra.ivt` (Jakarta Messaging 3.0) ou `wmq.jmsra.ivt` (JMS 2.0).
- Le fichier RAR IBM MQ ,`wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0 ou `wmq.jmsra.rar` (JMS 2.0).

5. JMS 2.0

Facultatif : Configurez l'adaptateur de ressources Java EE 7 (JMS 2.0) dans WebSphere Liberty Profile.

Les étapes à suivre pour configurer l'adaptateur de ressources dans Liberty sont les suivantes. Pour plus d'informations, voir Documentation produit WebSphere Application Server.

a) Ajoutez la fonction wmqJmsClient-2.0 au fichier `server.xml` pour pouvoir utiliser l'adaptateur de ressources IBM MQ .

Pour plus d'informations, voir «Version de l'adaptateur de ressources à utiliser», à la page 446.

b) Ajoutez une référence au fichier `wmq.jmsra.rar` (JMS 2.0) que vous avez installé.

Voici un exemple de configuration permettant de prendre en charge les servlets et les beans gérés par message avec JNDI :

```
 <featureManager>
    <feature>wmqJmsClient-2.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

6. JM 3.0

Facultatif : Configurez l'adaptateur de ressources Jakarta EE 9 (Jakarta Messaging 3.0) dans WebSphere Liberty Profile.

Les étapes à suivre pour configurer l'adaptateur de ressources dans Liberty sont les suivantes. Pour plus d'informations, voir Documentation produit WebSphere Application Server.

a) Ajoutez la fonction wmqJmsClient-3.0 au fichier `server.xml` pour permettre l'utilisation de l'adaptateur de ressources IBM MQ .

Pour plus d'informations, voir «Version de l'adaptateur de ressources à utiliser», à la page 446.

b) Ajoutez une référence au fichier `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) que vous avez installé.

Voici un exemple de configuration permettant de prendre en charge les servlets et les beans gérés par message avec JNDI :

```
 <featureManager>
    <feature>wmqJmsClient-3.0</feature>
    <feature>servlet-3.0</feature>
    <feature>jmsMdb-3.1</feature>
    <feature>jndi-1.0</feature>
    </featureManager>

<variable name="wmqJmsClient.rar.location"
      value="H:\Liberty\WMQ\wmq\wmq.jmsra.rar"/>
```

Remarque : Si vous utilisez Open Libertyau lieu de WebSphere Liberty Profile, vous devrez utiliser la fonction de prise en charge de l'adaptateur de ressources générique "messagingClient-3.0" à la place de "wmqJmsClient-3.0" et les autres aspects de la configuration seront différents. Pour plus de détails, reportez-vous à la documentation Open Liberty .

Configuration de l'adaptateur de ressources IBM MQ

Pour configurer l'adaptateur de ressources IBM MQ , vous devez définir diverses ressources Java Platform, Enterprise Edition Connector Architecture (JCA) et, en option, des propriétés système. Vous devez également configurer l'adaptateur de ressources pour exécuter le programme de test de

vérification de l'installation (IVT). Cela est important car le service IBM peut nécessiter l'exécution de ce programme pour indiquer que tout serveur d'applications nonIBM a été correctement configuré.

Avant de commencer

Cette tâche suppose que vous êtes déjà familiarisé avec JMS et IBM MQ classes for JMS. La plupart des propriétés utilisées pour configurer l'adaptateur de ressources IBM MQ sont équivalentes aux propriétés des objets IBM MQ classes for JMS et ont la même fonction.

Pourquoi et quand exécuter cette tâche

Chaque serveur d'applications fournit son propre ensemble d'interfaces d'administration. Certains serveurs d'applications fournissent des interfaces graphiques permettant de définir des ressources JCA , mais d'autres nécessitent que l'administrateur écrive des plans de déploiement XML. Il est donc hors de la portée de cette documentation de fournir des informations sur la configuration de l'adaptateur de ressources IBM MQ pour chaque serveur d'applications.

Les étapes suivantes se concentrent donc uniquement sur ce que vous devez configurer. Pour plus d'informations sur la configuration d'un adaptateur de ressources JCA , reportez-vous à la documentation fournie avec votre serveur d'applications.

Procédure

Définissez les ressources JCA dans les catégories suivantes:

- Définissez les propriétés de l'objet ResourceAdapter .

  Ces propriétés, qui représentent les propriétés globales de l'adaptateur de ressources, telles que le niveau de trace de diagnostic, sont décrites dans «Configuration des propriétés de l'objet ResourceAdapter», à la page 460.
- Définissez les propriétés d'un objet ActivationSpec .

  Ces propriétés déterminent comment un bean géré par message est activé pour la communication entrante. Pour plus d'informations, voir «Configuration de l'adaptateur de ressources pour les communications entrantes», à la page 463.
- Définissez les propriétés d'un objet ConnectionFactory .

  Le serveur d'applications utilise ces propriétés pour créer un objet JMS ConnectionFactory pour la communication sortante. Pour plus d'informations, voir «Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 482.
- Définissez les propriétés d'un objet de destination géré.

  Le serveur d'applications utilise ces propriétés pour créer un objet JMS Queue ou JMS Topic pour la communication sortante. Pour plus d'informations, voir «Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 482.
- Facultatif : Définissez un plan de déploiement pour l'adaptateur de ressources.

  Le fichier RAR de l'adaptateur de ressources IBM MQ contient un fichier appelé `META-INF/ra.xml`, qui contient un descripteur de déploiement pour l'adaptateur de ressources. Ce descripteur de déploiement est défini par le schéma XML dans le fichier https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd et contient des informations sur l'adaptateur de ressources et les services qu'il fournit. Un serveur d'applications peut également nécessiter un plan de déploiement pour l'adaptateur de ressources. Ce plan de déploiement est spécifique au serveur d'applications.

Spécifiez les propriétés système de la machine virtuelle Java selon les besoins:

- Si vous utilisez le protocole TLS (Transport Layer Security), indiquez les emplacements du fichier de clés et du fichier de clés certifiées en tant que propriétés système JVM, comme dans l'exemple suivant:

```
java ... -Djavax.net.ssl.keyStore=
key_store_location
```

```
-Djavax.net.ssl.trustStore=trust_store_location
-Djavax.net.ssl.keyStorePassword=key_store_password
```

Ces propriétés ne peuvent pas être des propriétés d'un objet ActivationSpec ou ConnectionFactory et vous ne pouvez pas spécifier plus d'un magasin de clés pour un serveur d'applications. Les propriétés s'appliquent à l'ensemble de la machine virtuelle Java et peuvent donc affecter le serveur d'applications si d'autres applications, exécutées sur le serveur d'applications, utilisent des connexions TLS. Le serveur d'applications peut également réinitialiser ces propriétés à des valeurs différentes. Pour plus d'informations sur l'utilisation de TLS avec IBM MQ classes for JMS, voir «Utilisation de TLS avec IBM MQ classes for JMS», à la page 262.

- Facultatif : Si nécessaire, configurez l'adaptateur de ressources pour consigner les messages d'avertissement dans le journal de sortie standard de votre serveur d'applications.

  Les journaux de l'adaptateur de ressources, les messages d'avertissement et d'erreur utilisent le même mécanisme que le IBM MQ classes for JMS. Pour plus d'informations, voir Erreurs de journalisation pour IBM MQ classes for JMS. Cela signifie que, par défaut, les messages sont envoyés dans un fichier appelé `mqjms.log`. Pour configurer l'adaptateur de ressources afin qu'il consigne en plus les messages d'avertissement dans le journal de sortie standard de votre serveur d'applications, définissez la propriété système JVM suivante pour votre serveur d'applications:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  Il s'agit de la même propriété que celle utilisée pour contrôler la trace pour IBM MQ classes for JMS. Comme avec IBM MQ classes for JMS, il est possible d'utiliser une propriété système pointant vers le fichier `jms.config` (voir «Le fichier de configuration IBM MQ classes for JMS/Jakarta Messaging», à la page 102). Pour plus d'informations sur la définition d'une propriété système JVM, consultez la documentation de votre serveur d'applications.

Configurez l'adaptateur de ressources pour exécuter le test de vérification de l'installation

- Configurez l'adaptateur de ressources pour exécuter le programme de test de vérification de l'installation (IVT) fourni avec l'adaptateur de ressources IBM MQ .

  Pour plus d'informations sur les éléments à configurer pour exécuter le programme IVT, voir «Vérification de l'installation de l'adaptateur de ressources», à la page 503.

  Ceci est important car le service IBM peut nécessiter l'exécution de ce programme pour indiquer que tout serveur d'applications nonIBM a été correctement configuré.

  Important : Vous devez configurer l'adaptateur de ressources avant de pouvoir exécuter le programme.

Configuration des propriétés de l'objet ResourceAdapter

L'objet ResourceAdapter encapsule les propriétés globales de l'adaptateur de ressources IBM MQ , telles que le niveau de trace de diagnostic. Pour définir ces propriétés, utilisez les fonctions de votre adaptateur de ressources, comme décrit dans la documentation fournie avec votre serveur d'applications.

L'objet ResourceAdapter possède deux ensembles de propriétés:

- Propriétés associées à la fonction de trace de diagnostic
- Propriétés associées au pool de connexions géré par l'adaptateur de ressources

La manière dont vous définissez ces propriétés dépend des interfaces d'administration fournies par votre serveur d'applications. Si vous utilisez WebSphere Application Server traditional, voir «Configuration de WebSphere Application Server traditional», à la page 462 ou si vous utilisez WebSphere Liberty, voir «Configuration de WebSphere Liberty», à la page 462. Pour les autres serveurs d'applications, consultez la documentation du produit correspondant à votre serveur d'applications.

Pour plus d'informations sur la définition des propriétés associées à la trace de diagnostic, voir Traçage de l'adaptateur de ressources IBM MQ

L'adaptateur de ressources gère un pool de connexions internes JMS qui sont utilisées pour distribuer des messages aux beans gérés par message. Le Tableau 63, à la page 461 répertorie les propriétés de l'objet ResourceAdapter qui sont associées au pool de connexions.

Tableau 63. Propriétés de l'objet ResourceAdapter associées au pool de connexions

Nom de la propriété	Tapez	Valeur par défaut	Description
maxConnections	String	50	Nombre maximal de connexions à un gestionnaire de files d'attente IBM MQ et nombre maximal de beans gérés par message déployés.
connectionConcurrency	String	1	Nombre maximal de beans gérés par message qui partagent une connexion JMS. Le partage des connexions n'est pas possible et cette propriété a toujours la valeur 1.
reconnectionRetryNombre	String	5	Nombre maximal de tentatives effectuées par l'adaptateur de ressources pour se reconnecter à un gestionnaire de files d'attente IBM MQ en cas d'échec d'une connexion.
reconnectionRetryIntervalle	String	300 000	Durée, en millisecondes, pendant laquelle l'adaptateur de ressources attend avant de tenter de se reconnecter à un gestionnaire de files d'attente IBM MQ .
startupRetryNombre	String	0	Nombre par défaut de tentatives de connexion à un bean géré par message au démarrage, si le gestionnaire de files d'attente n'est pas en cours d'exécution au démarrage du serveur d'applications.
startupRetryIntervalle	String	30 000	Temps de veille par défaut entre les tentatives de connexion au démarrage (en millisecondes).
supportMQExtensions	String	false	Rétablit le comportement IBM MQ JMS à un comportement antérieur àJMS 2.0 . Pour plus d'informations, voir «Propriété SupportMQExtensions», à la page 336.
nativeLibraryChemin	String	<vide>	Chemin à utiliser pour charger la bibliothèque JNI IBM MQ afin d'autoriser les connexions en mode liaisons. Windows Sous Windows , le chemin du système doit également contenir l'emplacement de l'installation IBM MQ correspondante.

Lorsqu'un bean géré par message est déployé dans le serveur d'applications, une nouvelle connexion JMS est créée et une conversation est démarrée avec le gestionnaire de files d'attente, à condition que le nombre maximal de connexions spécifié par la propriété maxConnection ne soit pas dépassé. Le nombre maximal de beans gérés par message est donc égal au nombre maximal de connexions. Si le nombre de beans gérés par message déployés atteint ce maximum, toute tentative de déploiement d'un autre bean géré par message échoue. Si un bean géré par message est arrêté, sa connexion peut être utilisée par un autre bean géré par message.

En général, si plusieurs beans gérés par message doivent être déployés, vous devez augmenter la valeur de la propriété maxConnections .

Les propriétés d'intervalle reconnectionRetryCount et reconnectionRetryrégissent le comportement de l'adaptateur de ressources lorsque les connexions à un gestionnaire de files d'attente IBM MQ échouent en raison d'une défaillance du réseau, par exemple. Lorsqu'une connexion échoue, l'adaptateur de ressources interrompt la distribution des messages à tous les beans gérés par message fournis par cette connexion pendant un intervalle spécifié par la propriété d'intervalle reconnectionRetry. L'adaptateur de ressources tente ensuite de se reconnecter au gestionnaire de files d'attente. Si la tentative échoue, l'adaptateur de ressources effectue d'autres tentatives de reconnexion à des intervalles spécifiés par la propriété reconnectionRetryInterval jusqu'à ce que la limite imposée par la propriété reconnectionRetryCount soit atteinte. Si toutes les tentatives échouent, la distribution est arrêtée définitivement jusqu'à ce que les beans gérés par message soient redémarrés manuellement.

En général, l'objet ResourceAdapter ne nécessite aucune administration. Toutefois, pour activer le traçage des diagnostics sur les systèmes AIX and Linux , par exemple, vous pouvez définir les propriétés suivantes:

```
traceEnabled:    true
traceLevel:      10
```

Ces propriétés n'ont aucun effet si l'adaptateur de ressources n'a pas été démarré, ce qui est le cas, par exemple, lorsque des applications utilisant des ressources IBM MQ s'exécutent uniquement dans le conteneur client. Dans cette situation, vous pouvez définir les propriétés du traçage de diagnostic en tant que propriétés système de la machine virtuelle Java (JVM). Vous pouvez définir les propriétés à l'aide de l'indicateur -D de la commande **java** , comme dans l'exemple suivant:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Il n'est pas nécessaire de définir toutes les propriétés de l'objet ResourceAdapter . Toutes les propriétés non spécifiées prennent leurs valeurs par défaut. Dans un environnement géré, il est préférable de ne pas mélanger les deux méthodes de spécification des propriétés. Si vous les mélangez, les propriétés système JVM sont prioritaires sur les propriétés de l'objet ResourceAdapter .

Configuration de WebSphere Application Server traditional

Les mêmes propriétés sont disponibles pour l'adaptateur de ressources dans WebSphere Application Server traditional, mais elles doivent être définies dans le panneau des propriétés de l'adaptateur de ressources (voir Paramètres du fournisseur JMS dans la documentation du produit WebSphere Application Server traditional . La trace est contrôlée par la section des diagnostics de la configuration WebSphere Application Server traditional . Pour plus d'informations, voir Utilisation des fournisseurs de diagnostic dans la documentation du produit WebSphere Application Server traditional .

Configuration de WebSphere Liberty

L'adaptateur de ressources est configuré à l'aide d'éléments XML dans le fichier `server.xml` , comme illustré dans l'exemple suivant:

JM 3.0

```
<featureManager>
...
    <feature>messaging-3.0</feature>
...
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jakarta.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

JMS 2.0

```
<featureManager>
...
    <feature>wmqJmsClient-2.0</feature>
...
```

```
</featureManager>
    <variable name="wmqJmsClient.rar.location"
        value="F:/_rtc_wmq8005/_build/ship/lib/jca/wmq.jmsra.rar"/>
...
        <wmqJmsClient supportMQExtensions="true" logWriterEnabled="true"/>
```

La trace est activée en ajoutant cet élément XML:

```
<logging traceSpecification="JMSApi=all:WAS.j2c=all:"/>
```

Configuration de l'adaptateur de ressources pour les communications entrantes

Pour configurer les communications entrantes, définissez les propriétés d'un ou plusieurs objets ActivationSpec.

Les propriétés d'un objet ActivationSpec déterminent comment un bean géré par message (MDB) reçoit des messages JMS d'une file d'attente IBM MQ . Le comportement transactionnel du bean géré par message est défini dans son descripteur de déploiement.

Un objet ActivationSpec possède deux ensembles de propriétés:

- Propriétés utilisées pour créer une connexion JMS à un gestionnaire de files d'attente IBM MQ
- Propriétés utilisées pour créer un consommateur de connexion JMS qui distribue les messages de manière asynchrone lorsqu'ils arrivent dans une file d'attente spécifiée

La manière dont vous définissez les propriétés d'un objet ActivationSpec dépend des interfaces d'administration fournies par votre serveur d'applications.

Propriétés utilisées pour créer une connexion JMS à un gestionnaire de files d'attente IBM MQ

Toutes les propriétés de Tableau 64, à la page 463 sont facultatives.

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
applicationName	String	• Le nom de la classe appelante, s'il est disponible, est ajusté pour ne pas dépasser 28 caractères. S'il n'est pas disponible, la chaîne `WebSphere MQ Client for Java` est utilisée.	Nom sous lequel une application est enregistrée auprès du gestionnaire de files d'attente. Ce nom d'application est affiché par la commande **DISPLAY CONN MQSC/PCF** (où la zone est appelée **APPLTAG**) ou dans l'écran IBM MQ Explorer Connexions d'application (où la zone est appelée **App name**).
brokerCCDurSubQueue [1]	String	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de connexion reçoit des messages d'abonnement durable

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
brokerCCSubFile d'attente [1]	String	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de connexion reçoit des messages d'abonnement non durable
brokerControlFile d'attente [1]	String	• **SYSTEM.BROKER.CONTROL.QUEUE** • Un nom de file d'attente	Nom de la file d'attente de contrôle du courtier
brokerQueueGestionnaire [1]	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente sur lequel le courtier s'exécute
brokerSubFile d'attente [1]	String	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de message non durable reçoit des messages
brokerVersion [1]	String	• **unspecified** -Après la migration du courtier de V6 vers V7, définissez cette propriété de sorte que les en-têtes RFH2 ne soient plus utilisés. Après la migration, cette propriété n'est plus significative. • **V1** -Pour utiliser un courtier IBM MQ publish/subscribe broker.This est la valeur par défaut si TRANSPORT est défini sur BIND ou CLIENT. • **V2** -Pour utiliser un courtier de IBM Integration Bus en mode natif. Cette valeur est la valeur par défaut si TRANSPORT est défini sur DIRECT ou DIRECTIVE THTTP.	Version du courtier utilisé
ccdtURL	String	• **null** • Un URL (uniform resource locator)	URL qui identifie le nom et l'emplacement du fichier contenant la table de définition de canal du client (CCDT) et qui indique comment accéder au fichier
CCSID	String	• **819** • Identificateur de jeu de caractères codés pris en charge par la machine virtuelle Java (JVM)	Identificateur de jeu de caractères codés pour une connexion
canal	String	• **SYSTEM.DEF.SVRCONN** • Nom d'un canal MQI	Nom du canal MQI à utiliser

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
cleanupInterval [1]	int	• **3 600 000** • Entier positif	Intervalle, en millisecondes, entre les exécutions en arrière-plan de l'utilitaire de nettoyage de publication / abonnement
cleanupLevel [1]	String	• **SECURISEE** • Aucun • FONDATION • FORCE • NONDUR	Niveau de nettoyage d'un magasin d'abonnements basé sur un courtier
clientID	String	• **null** • Un identificateur de client	Identificateur client d'une connexion
cloneSupport	String	• **DISABLED** -Une seule instance d'un abonné durable à la rubrique peut s'exécuter à la fois. • ENABLED-Deux ou plusieurs instances du même abonné durable à la rubrique peuvent s'exécuter simultanément, mais chaque instance doit s'exécuter dans une machine virtuelle Java (JVM) distincte.	Indique si deux instances ou plus du même abonné durable à la rubrique peuvent être exécutées simultanément
connectionFactoryRecherche	String	• **null** • Nom JNDI d'un objet ConnectionFactory	Si cette propriété est définie, ActivationSpec recherche un objet JMS ConnectionFactory avec le nom JNDI spécifié dans l'espace de nom JNDI du serveur d'applications, puis utilise les propriétés de cet objet pour créer une connexion JMS à un gestionnaire de files d'attente IBM MQ , avec une exception. La seule propriété de ActivationSpec qui sera utilisée lors de la création de la connexion JMS est clientID. Pour plus d'informations, voir «Propriétés ActivationSpec connectionFactoryLookup et destinationLookup», à la page 478.

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
ConnectionNameList	String	• **localhost (1414)** • Chaîne composée d'éléments séparés par des virgules, où chaque élément prend le format suivant: `HOSTNAME(PORT)` où *HOSTNAME* est un nom DNS ou une adresse IP.	Liste des noms de connexion TCP/IP utilisés pour les communications entrantes. Lorsqu'il est spécifié, **connectionNameList** remplace les propriétés **hostname** et **port** . Cette propriété permet de se reconnecter aux gestionnaires de files d'attente multi-instance. La forme de **connectionNameList** est similaire à **localAddress**, mais ne doit pas être confondue avec celle-ci. **localAddress** indique les caractéristiques des communications locales, tandis que **connectionNameList** indique comment atteindre un gestionnaire de files d'attente éloignées.
dynamicallyBalanced [4]	Boolean	• **faux** • Oui	Indique si ce bean géré par message peut être invité à recevoir des messages d'un gestionnaire de files d'attente différent dans le cadre de l'équilibrage des applications dans un cluster uniforme.
failIfQuiesce	Boolean	• **true** • false	Indique si les appels à certaines méthodes échouent si le gestionnaire de files d'attente est à l'état de mise au repos
headerCompression	String	• **Aucun** • La compression de l'en-tête de message SYSTEM-RLE est effectuée	Liste des techniques pouvant être utilisées pour compresser les données d'en-tête sur une connexion

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
hostName	String	• **système hôte local** • Un nom d'hôte • Une adresse IP	Nom d'hôte ou adresse IP du système sur lequel réside le gestionnaire de files d'attente. Les propriétés **hostname** et **port** sont remplacées par la propriété **connectionNameList** lorsqu'elle est spécifiée.
localAddress	String	• **null** • Chaîne au format: `[ host_name ][(low_port [, high_port ])]` où *nom_hôte* est un nom d'hôte ou une adresse IP, *port_inférieur* et *port_supérieur* sont des numéros de port TCP et les crochets indiquent un composant facultatif	Pour une connexion à un gestionnaire de files d'attente, cette propriété spécifie l'un des éléments suivants ou les deux: • Interface réseau locale à utiliser • Port local, ou plage de ports locaux, à utiliser La forme de **localAddress** est similaire à **connectionNameList**, mais ne doit pas être confondue avec celle-ci. **localAddress** indique les caractéristiques des communications locales, tandis que **connectionNameList** indique comment atteindre un gestionnaire de files d'attente éloignées.
messageCompression	String	• **Aucun** • Liste d'une ou de plusieurs des valeurs suivantes, séparées par des caractères blancs: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Liste des techniques pouvant être utilisées pour compresser les données de message sur une connexion
messageRetention [1]	Boolean	• **true** -Les messages non souhaités restent dans la file d'entrée • false-Les messages non souhaités sont traités en fonction de leurs options de disposition	Indique si le destinataire de la connexion conserve les messages indésirables dans la file d'entrée

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
messageSelection [1]	String	• **client** • COURTIER	Détermine si la sélection des messages est effectuée par IBM MQ classes for JMS ou par le courtier. La sélection de messages par le courtier n'est pas prise en charge lorsque brokerVersion a la valeur 1.
Mot de passe	String	• **null** • Un mot de passe	Mot de passe par défaut à utiliser lors de la création d'une connexion au gestionnaire de files d'attente
pollingInterval [1]	int	• **5000** • Tout entier positif	Lorsque la file d'attente associée aux différents programmes d'écoute de message dans une session ne contient aucun message approprié, cette valeur est l'intervalle de temps maximal, en millisecondes, qui peut s'écouler avant que chaque programme d'écoute tente à nouveau d'extraire un message de sa file d'attente. Si l'absence de message approprié est fréquemment observée pour l'un quelconque des écouteurs de messages au sein d'une session, envisagez d'augmenter la valeur de cette propriété. Cette propriété est pertinente uniquement si TRANSPORT a la valeur BIND ou CLIENT.
port	int	• **1414** • Un numéro de port TCP	Port sur lequel le gestionnaire de files d'attente écoute. Les propriétés **hostname** et **port** sont remplacées par la propriété **connectionNameList** lorsqu'elle est spécifiée.

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
providerVersion	chaîne	• **non spécifié** • Une chaîne dans l'un des formats suivants – V.R.M.F – V.R.M – V.R – V où V, R, M et F sont des valeurs entières supérieures ou égales à zéro.	Version, édition, niveau de modification et groupe de correctifs du gestionnaire de files d'attente auquel le bean géré par message a l'intention de se connecter.
queueManager	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente auquel se connecter
receiveExit [3]	String	• **null** • Chaîne comprenant un ou plusieurs éléments séparés par des virgules, où chaque élément correspond au nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQReceiveExit	Identifie un programme d'exit de réception de canal ou une séquence de programmes d'exit de réception à exécuter successivement
receiveExitInitialisation	String	• **null** • Chaîne comprenant un ou plusieurs éléments de données utilisateur séparés par des virgules	Données utilisateur transmises aux programmes d'exit de réception de canal lorsqu'ils sont appelés

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
rescanInterval [1]	int	• **5000** • Tout entier positif	Lorsqu'un consommateur de message du domaine point à point utilise un sélecteur de message pour sélectionner les messages qu'il souhaite recevoir, IBM MQ classes for JMS recherche dans la file d'attente IBM MQ les messages appropriés dans la séquence déterminée par l'attribut **MsgDeliverySequence** de la file d'attente. Lorsque IBM MQ classes for JMS trouve un message approprié et le distribue au consommateur, IBM MQ classes for JMS reprend la recherche du message approprié suivant à partir de sa position actuelle dans la file d'attente. IBM MQ classes for JMS continue de rechercher la file d'attente de cette manière jusqu'à ce qu'elle atteigne la fin de la file d'attente ou jusqu'à ce que l'intervalle de temps en millisecondes, déterminé par la valeur de cette propriété, ait expiré. Dans chaque cas, IBM MQ classes for JMS revient au début de la file d'attente pour poursuivre sa recherche et un nouvel intervalle de temps commence.
securityExit [3]	String	• **null** • Nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQSecurityExit	Identifie un programme d'exit de sécurité de canal
securityExitInit	String	• **null** • Une chaîne de données utilisateur	Données utilisateur transmises à un programme d'exit de sécurité de canal lorsqu'il est appelé

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
sendExit [3]	String	• **null** • Chaîne comprenant un ou plusieurs éléments séparés par des virgules, où chaque élément est le nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQSendExit	Identifie un programme d'exit d'émission de canal ou une séquence de programmes d'exit d'émission à exécuter successivement
SENDEXITINIT	String	• **null** • Chaîne comprenant un ou plusieurs éléments de données utilisateur séparés par des virgules	Données utilisateur transmises aux programmes d'exit d'émission de canal lorsqu'ils sont appelés
SHARECONVALLOWED	Boolean	• NON-Une connexion client ne peut pas partager son socket. • **YES** -Une connexion client peut partager son socket.	Indique si une connexion client peut partager son socket avec d'autres connexions JMS de niveau supérieur à partir du même processus vers le même gestionnaire de files d'attente, si les définitions de canal correspondent
sparseSubscriptions [1]	Boolean	• **false** -Les abonnements reçoivent des messages de correspondance fréquents. • true-Les abonnements reçoivent des messages correspondants peu fréquents. Cette valeur nécessite que la file d'attente d'abonnement puisse être ouverte pour l'exploration.	Contrôle la stratégie d'extraction de message d'un objet TopicSubscriber
sslCertMagasins	String	• **null** • Chaîne d'une ou de plusieurs URL LDAP séparées par des blancs. Chaque URL LDAP a le format suivant: `ldap://host_name [: port ]` où *nom_hôte* est un nom d'hôte ou une adresse IP, *port* est un numéro de port TCP et les crochets indiquent un composant facultatif.	Les serveurs LDAP (Lightweight Directory Access Protocol) qui contiennent des listes de révocation de certificat (CRL) à utiliser sur une connexion TLS
SSLCIPHERSUITE	String	• **null** • Nom d'une CipherSuite	CipherSuite à utiliser pour une connexion TLS
sslFipsObligatoire [2]	Boolean	• **faux** • Oui	Indique si une connexion TLS doit utiliser une CipherSuite prise en charge par le fournisseur IBM Java JSSE FIPS (IBMJSSEFIPS)

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
SSLPEERNAME	String	• **null** • Modèle pour les noms distinctifs	Pour une connexion TLS, modèle utilisé pour vérifier le nom distinctif dans le certificat numérique fourni par le gestionnaire de files d'attente
SSLRESETCOUNT	int	• **0** • Entier compris entre 0 et 999 999 999	Nombre total d'octets envoyés et reçus par une connexion TLS avant que les clés secrètes utilisées par TLS ne soient renégociées
Fabrique sslSocket	String	Chaîne représentant le nom de classe qualifié complet d'une classe fournissant une implémentation de l'interface javax.net.ssl.SSLSocketFactory . Inclusion facultative d'un argument à transmettre à la méthode du constructeur, entre parenthèses.	Toutes les connexions établies dans la portée de l'objet administré utilisent des sockets obtenues à partir de cette implémentation de l'interface SSLSocketFactory .
statusRefreshIntervalle [1]	int	• **60000** • Tout entier positif	Intervalle, en millisecondes, entre les actualisations de la transaction à exécution longue qui détecte lorsqu'un abonné perd sa connexion au gestionnaire de files d'attente. Cette propriété est pertinente uniquement si **subscriptionStore** a la valeur QUEUE.
subscriptionStore [1]	String	• **Courtier** • MIGRATE • QUEUE	Détermine où IBM MQ classes for JMS stocke les données persistantes sur les abonnements actifs

Tableau 64. Propriétés d'un objet ActivationSpec utilisées pour créer une connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
transportType	String	• **client** • LIAISONS • BINDINGS_THEN_CLIENT	Indique si une connexion à un gestionnaire de files d'attente utilise le mode client ou le mode liaisons. Si la valeur `BINDINGS_THEN_CLIENT` est spécifiée, l'adaptateur de ressources tente d'abord d'établir une connexion en mode liaisons. Si cette tentative de connexion échoue, l'adaptateur de ressources tente d'établir une connexion en mode client. z/OS Si une spécification d'activation qui s'exécute sur un système WebSphere Application Server for z/OS a été configurée pour utiliser le mode de transport `BINDINGS_THEN_CLIENT` et qu'une connexion établie précédemment est interrompue, toutes les tentatives de reconnexion effectuées par la spécification d'activation tentent d'abord d'utiliser le mode de transport `BINDINGS` . Si la tentative de connexion en mode transport `BINDINGS` échoue, la spécification d'activation tente ensuite une connexion en mode transport `CLIENT` .
username	String	• **null** • Nom d'utilisateur	Nom d'utilisateur par défaut à utiliser lors de la création d'une connexion à un gestionnaire de files d'attente
wildcardFormat	String	• CHAR-Reconnaissance des caractères génériques uniquement, tels qu'ils sont utilisés dans la version 1 du courtier • **TOPIC** -Reconnaît uniquement les caractères génériques de niveau rubrique, tels qu'ils sont utilisés dans la version 2 du courtier	Version de la syntaxe générique à utiliser

Remarques :

1. Cette propriété peut être utilisée avec la version 70 de IBM MQ classes for JMS.
2. Pour des informations importantes sur l'utilisation de la propriété sslFipsRequired, voir «Limitations de l'adaptateur de ressources IBM MQ», à la page 450.
3. Pour plus d'informations sur la configuration de l'adaptateur de ressources afin qu'il puisse localiser un exit, voir «Configuration de IBM MQ classes for JMS pour l'utilisation des exits de canal», à la page 287.
4. La propriété dynamicallyBalanced n'est pas prise en charge conjointement avec la prise en charge des transactions XA. Si dynamicallyBalanced a la valeur "true", le bean géré par message doit être configuré pour désactiver les transactions XA.

Propriétés utilisées pour créer un consommateur de connexion JMS

Remarque : Les paramètres **destination** et **destinationType** doivent être définis explicitement. Toutes les autres propriétés de Tableau 65, à la page 474 sont facultatives.

Tableau 65. Propriétés d'un objet ActivationSpec utilisées pour créer un consommateur de connexion JMS

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
destination	String	Un nom de destination	Destination à partir de laquelle les messages doivent être reçus. La propriété **useJNDI** détermine comment la valeur de cette propriété est interprétée.
destinationLookup	String	• **null** • Nom JNDI d'un objet Destination	Si cette propriété est définie, ActivationSpec recherche un objet JMS Destination avec le nom JNDI spécifié dans l'espace de nom JNDI du serveur d'applications, puis utilise les propriétés de cet objet pour créer un consommateur de connexion JMS , de préférence aux autres propriétés spécifiées dans ActivationSpec. Pour plus d'informations, voir «Propriétés ActivationSpec connectionFactoryLookup et destinationLookup», à la page 478.
destinationType	String	• jakarta.jms.Queue (Jakarta Messaging 3.0) • jakarta.jms.Topic (Jakarta Messaging 3.0) • javax.jms.Queue (JMS 2.0) • javax.jms.Topic (JMS 2.0)	Type de destination, file d'attente ou rubrique
maxMessages	int	• **1** • Entier positif	Nombre maximal de messages pouvant être affectés simultanément à une session de serveur. Si la spécification d'activation distribue des messages à un bean géré par message dans une transaction XA, la valeur 1 est utilisée quel que soit le paramètre de cette propriété.

Tableau 65. Propriétés d'un objet ActivationSpec utilisées pour créer un consommateur de connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
Profondeur maxPool	int	• **10** • Entier positif	Nombre maximal de sessions de serveur dans le pool de sessions de serveur utilisé par le consommateur de connexion
messageSelector	String	• **null** • Une expression de sélecteur de message SQL92	Une expression de sélecteur de message spécifiant les messages à distribuer
nonASFTimeout	int	• **0** • Entier positif	Une valeur positive indique que la distribution non ASF est utilisée. La valeur correspond à la durée, en millisecondes, pendant laquelle une demande d'obtention attend les messages qui ne sont peut-être pas encore arrivés (appel d'obtention avec attente). La valeur par défaut, 0, indique que la distribution ASF est utilisée. Ce paramètre est valide si: • L'application s'exécute sous WebSphere Application Server 7.0 ou version ultérieure. • L'application s'exécute dans WebSphere Liberty à l'aide du niveau approprié de la fonction client wmqJms. Pour plus d'informations, voir «Liberty et l'adaptateur de ressources IBM MQ», à la page 451.
nonASFRollbackactivé	Boolean	• **false** -Le message est consommé même si le bean géré par message échoue • true-L'échec au sein du bean géré par message entraîne l'annulation du message dans la file d'attente.	Indique si la distribution des messages se trouve dans un point de synchronisation IBM MQ si le bean géré par message n'est pas traité. Ignoré si le bean géré par message est traité ou si **nonASFTimeout** est défini sur 0.
poolTimeout	int	• **300000** • Entier positif	Durée, en millisecondes, pendant laquelle une session de serveur inutilisée est maintenue ouverte dans le pool de sessions de serveur avant d'être fermée en raison d'une inactivité

Tableau 65. Propriétés d'un objet ActivationSpec utilisées pour créer un consommateur de connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
READAHEADALLOWED	int	• **DESTINATION** -Déterminez si la lecture anticipée est autorisée en faisant référence à la définition de la file d'attente ou de la rubrique. • DISABLED-La lecture anticipée n'est pas autorisée. • ENABLED-La lecture anticipée est autorisée. • QUEUE-Déterminer si la lecture anticipée est autorisée en faisant référence à la définition de file d'attente. • SUJET-Déterminer si la lecture anticipée est autorisée en faisant référence à la définition de rubrique.	Indique si l'unité d'exécution de navigation de la spécification d'activation est autorisée à utiliser la lecture anticipée pour parcourir plusieurs messages de la destination dans une mémoire tampon interne, avant de les transmettre aux sessions du serveur pour une consommation destructive. **Remarque :** L'activation de la lecture anticipée peut entraîner une augmentation des messages JMSCC0108 et / ou une diminution des performances si le débit de traitement du bean géré par message ne peut pas suivre le débit de consultation des messages à partir de la destination.
readAheadClosePolicy	int	• **ALL** -Tous les messages de la mémoire tampon de lecture anticipée interne sont distribués au bean géré par message avant son arrêt. • CURRENT-Seul l'appel du bean géré par message en cours se termine, laissant potentiellement des messages dans la mémoire tampon de lecture anticipée interne, qui sont ensuite supprimés.	Qu'arrive-t-il aux messages de la mémoire tampon de lecture anticipée interne lorsque le bean géré par message est arrêté par l'administrateur?
receiveCCSID	int	• **0** -Utiliser la machine virtuelle Java `Charset.defaultCharset` • 1208- UTF-8 • Identificateur de jeu de caractères codés pris en charge	Propriété de destination qui définit le CCSID cible pour la conversion des messages du gestionnaire de files d'attente. La valeur est ignorée sauf si **`receiveConversion`** est défini sur `QMGR`.
receiveConversion	String	• **CLIENT_MSG** • QMGR	Propriété de destination qui détermine si la conversion de données sera effectuée par le gestionnaire de files d'attente.

Tableau 65. Propriétés d'un objet ActivationSpec utilisées pour créer un consommateur de connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
sharedSubscription	Boolean	• **False** -Le bean géré par message ne doit pas ouvrir l'abonnement en tant qu'abonnement partagé. • True-Le bean géré par message doit ouvrir l'abonnement en tant qu'abonnement partagé (avec les règles que JMS 2.0 implique, voir la spécification JMS 2.0 à l'adresse Java.net).	Contrôle la façon dont un bean géré par message est géré à partir d'un abonnement partagé. Pour plus d'informations sur l'utilisation de cette propriété, voir «Exemples de définition de la propriété sharedSubscription», à la page 481.
startTimeout	int	• **10 000** • Entier positif	Durée, en millisecondes, pendant laquelle la distribution d'un message à un bean géré par message doit commencer après que le travail de distribution du message a été planifié. Si ce délai est écoulé, le message est annulé dans la file d'attente.
subscriptionDurability	String	• **NonDurable** -Un abonnement non durable est utilisé pour distribuer des messages à un bean géré par message s'abonnant à la rubrique. • Durable-Un abonnement durable est utilisé pour distribuer des messages à un bean géré par message s'abonnant à la rubrique.	Indique si un abonnement durable ou non durable est utilisé pour distribuer des messages à un bean géré par message s'abonnant à la rubrique
subscriptionName	String	• **"" (chaîne vide)** • Un nom d'abonnement	Nom de l'abonnement durable

Tableau 65. Propriétés d'un objet ActivationSpec utilisées pour créer un consommateur de connexion JMS (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
useJNDI	Boolean	• **false** -La propriété appelée destination est interprétée comme le nom d'une file d'attente IBM MQ ou d'une rubrique. • true-La propriété appelée destination est interprétée comme le nom de l'un des objets suivants dans l'espace de nom JNDI du serveur d'applications: – jakarta.jms.Queue (Jakarta Messaging 3.0) – jakarta.jms.Topic (Jakarta Messaging 3.0) – javax.jms.Queue (JMS 2.0) – javax.jms.Topic (JMS 2.0)	Deprecated Détermine comment la valeur de la propriété appelée destination est interprétée. **Remarque :** Cette propriété est obsolète dans IBM MQ 9.0. La propriété destinationLookup doit être utilisée à la place.

Conflits de propriété et dépendances

Un objet ActivationSpec peut avoir des propriétés en conflit. Par exemple, vous pouvez spécifier des propriétés TLS pour une connexion en mode liaisons. Dans ce cas, le comportement est déterminé par le type de transport et le domaine de messagerie, qui est soit point à point, soit publication / abonnement, comme déterminé par la propriété **destinationType** . Toutes les propriétés qui ne sont pas applicables au type de transport ou au domaine de messagerie spécifié sont ignorées.

Si vous définissez une propriété qui requiert la définition d'autres propriétés, mais que vous ne définissez pas ces autres propriétés, l'objet ActivationSpec émet une exception InvalidPropertylorsque sa méthode validate () est appelée lors du déploiement d'un bean géré par message. L'exception est signalée à l'administrateur du serveur d'applications d'une manière qui dépend du serveur d'applications. Par exemple, si vous définissez la propriété subscriptionDurability sur Durable, indiquant que vous souhaitez utiliser des abonnements durables, vous devez également définir la propriété **subscriptionName** .

Si les propriétés appelées **ccdtURL** et **channel** sont toutes deux définies, une exception InvalidPropertyest émise. Toutefois, si vous définissez uniquement la propriété **ccdtURL** , la propriété appelée **channel** reste avec sa valeur par défaut SYSTEM.DEF.SVRCONN, aucune exception n'est émise et la table de définition de canal du client identifiée par la propriété **ccdtURL** est utilisée pour démarrer une connexion JMS .

Propriétés ActivationSpec connectionFactoryLookup et destinationLookup

La spécification JMS 2.0 a introduit deux nouvelles propriétés ActivationSpec . Les propriétés connectionFactoryLookup et destinationLookup peuvent être fournies avec le nom JNDI d'un objet géré à utiliser de préférence aux autres propriétés ActivationSpec .

Par exemple, si une fabrique de connexions est définie dans JNDI et que le nom JNDI de cet objet est spécifié dans la propriété de recherche connectionFactorypour une spécification d'activation, toutes les propriétés de la fabrique de connexions qui sont définies dans JNDI sont utilisées de préférence aux propriétés dans Tableau 64, à la page 463.

Si une destination est définie dans JNDI et que le nom JNDI est défini dans la propriété destinationLookup de ActivationSpec, les valeurs de cette dernière sont utilisées de préférence aux

valeurs de Tableau 65, à la page 474. Pour plus d'informations sur l'utilisation de ces deux propriétés, voir «Propriétés ActivationSpec connectionFactoryLookup et destinationLookup», à la page 478.

Ces deux propriétés peuvent être utilisées pour spécifier les noms JNDI des objets ConnectionFactory et Destination qui sont utilisés de préférence aux propriétés de ActivationSpec telles que définies dans Tableau 64, à la page 463 et Tableau 65, à la page 474.

Il est important de noter les points suivants qui décrivent en détail le fonctionnement de ces propriétés.

connectionFactoryRecherche
La ConnectionFactory recherchée dans JNDI est utilisée comme source des propriétés répertoriées dans Tableau 64, à la page 463. L'objet ConnectionFactory n'est pas utilisé pour créer réellement des connexions JMS , seules les propriétés de l'objet sont interrogées. Ces propriétés de la ConnectionFactory remplacent toutes les propriétés définies dans ActivationSpec. Il n'y a qu'une seule exception à cette règle. Si la propriété **ClientID** est définie pour ActivationSpec , la valeur de cette propriété remplace la valeur spécifiée dans la ConnectionFactory. En effet, un scénario courant consiste à utiliser une seule ConnectionFactory avec plusieurs ActivationSpecs. Cela simplifie l'administration. Toutefois, la spécification JMS 2.0 indique que chaque connexion JMS créée à partir d'une ConnectionFactory doit avoir un **ClientID**unique. Pour cette raison, ActivationSpecs doit avoir la possibilité de remplacer toute valeur définie dans la ConnectionFactory. Si aucun **ClientID** n'est défini sur ActivationSpec, toute valeur de la fabrique de connexions est utilisée.

destinationLookup
Une propriété **Destination** et une propriété **UseJndi** sont définies dans ActivationSpec. Si l'indicateur **UseJndi** est défini sur `true`, le texte spécifié dans la propriété de destination est considéré comme un nom JNDI et un objet de destination avec ce nom JNDI est recherché à partir de JNDI.

La propriété destinationLookup se comporte exactement de la même manière. S'il a été défini, un objet de destination avec le nom JNDI spécifié par la propriété est recherché à partir de JNDI. Cette propriété est prioritaire sur la propriété **useJNDI** .

Deprecated La propriété useJNDI est obsolète dans IBM MQ 9.0 car la propriété **destinationLookup** est la spécification JMS 2.0 ou l'équivalent ultérieur de l'exécution de la même fonction.

Propriétés ActivationSpec sans équivalent dans IBM MQ classes for JMS

La plupart des propriétés d'un objet ActivationSpec sont équivalentes aux propriétés des objets IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging ou aux paramètres des méthodes IBM MQ classes for JMS IBM MQ classes for Jakarta Messaging . Toutefois, trois propriétés d'optimisation et une propriété de mode d'utilisation n'ont pas d'équivalents dans IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging:

startTimeout
Durée, en millisecondes, pendant laquelle le gestionnaire de travaux du serveur d'applications attend que des ressources deviennent disponibles après que l'adaptateur de ressources a planifié un objet de travail pour distribuer un message à un bean géré par message. Si ce délai s'écoule avant le début de la distribution du message, l'objet de travail arrive à expiration, le message est annulé dans la file d'attente et l'adaptateur de ressources peut alors tenter à nouveau de distribuer le message. Un avertissement est consigné dans la trace de diagnostic, si elle est activée, mais n'affecte pas le processus de distribution des messages. Vous pouvez vous attendre à ce que cette condition se produise uniquement lorsque le serveur d'applications est confronté à une charge très élevée. Si la condition se produit régulièrement, envisagez d'augmenter la valeur de cette propriété pour que le gestionnaire de travaux ait plus de temps pour planifier la distribution des messages.

Profondeur maxPool
Nombre maximal de sessions de serveur dans le pool de sessions de serveur utilisé par un consommateur de connexion. Lorsqu'une session de serveur est créée, elle démarre une conversation avec un gestionnaire de files d'attente. Le consommateur de connexion utilise une session de serveur pour distribuer un message à un bean géré par message. Une plus grande profondeur de pool permet la distribution simultanée d'un plus grand nombre de messages dans des situations de volume élevé,

mais utilise davantage de ressources du serveur d'applications. Si un grand nombre de beans gérés par message doivent être déployés, envisagez de réduire la profondeur du pool afin de maintenir la charge sur le serveur d'applications à un niveau gérable. Chaque consommateur de connexion utilise son propre pool de sessions de serveur, de sorte que cette propriété ne définit pas le nombre total de sessions de serveur disponibles pour tous les consommateurs de connexion.

poolTimeout
Durée, en millisecondes, pendant laquelle une session de serveur inutilisée est maintenue ouverte dans le pool de sessions de serveur avant d'être fermée en raison d'une inactivité. Une augmentation transitoire de la charge de travail des messages entraîne la création de sessions de serveur supplémentaires afin de répartir la charge, mais une fois la charge de travail des messages redevient normale, les sessions de serveur supplémentaires restent dans le pool et ne sont pas utilisées.

Chaque fois qu'une session de serveur est utilisée, elle est marquée avec un horodatage. Une unité d'exécution scavenger vérifie périodiquement que chaque session de serveur a été utilisée au cours de la période spécifiée par cette propriété. Si une session serveur n'a pas été utilisée, elle est fermée et supprimée du pool de sessions serveur. Il se peut qu'une session de serveur ne soit pas fermée immédiatement après l'expiration de la période spécifiée. Cette propriété représente la période d'inactivité minimale avant la suppression.

useJNDI
Pour une description de cette propriété, voir Tableau 65, à la page 474.

Déploiement d'un bean géré par message

Pour déployer un bean géré par message, définissez d'abord les propriétés d'un objet ActivationSpec , en spécifiant les propriétés requises par le bean géré par message. L'exemple suivant est un ensemble typique de propriétés que vous pouvez définir explicitement:

JM 3.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   jakarta.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

JMS 2.0

```
channel:           SYSTEM.DEF.SVRCONN
destination:       SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:   javax.jms.Queue
hostName:          192.168.0.42
messageSelector:   color='red'
port:              1414
queueManager:      ExampleQM
transportType:     CLIENT
```

Le serveur d'applications utilise les propriétés pour créer un objet ActivationSpec , qui est ensuite associé à un bean géré par message. Les propriétés de l'objet ActivationSpec déterminent la manière dont les messages sont distribués au bean géré par message. Le déploiement du bean géré par message échoue si le bean géré par message requiert des transactions réparties mais que l'adaptateur de ressources ne prend pas en charge les transactions réparties. Pour plus d'informations sur l'installation de l'adaptateur de ressources afin que les transactions réparties soient prises en charge, voir «Installation de l'adaptateur de ressources IBM MQ», à la page 454.

Si plusieurs beans gérés par message reçoivent des messages de la même destination, un message envoyé dans le domaine point à point est reçu par un seul bean géré par message, même si d'autres beans gérés par message sont éligibles pour recevoir le message. En particulier, si deux beans gérés par message utilisent des sélecteurs de message différents et qu'un message entrant correspond aux deux sélecteurs de message, un seul des beans gérés par message reçoit le message. Le bean géré par message choisi pour recevoir un message n'est pas défini et vous ne pouvez pas vous fier à un bean géré

par message spécifique qui reçoit le message. Les messages envoyés dans le domaine de publication / abonnement sont reçus par tous les beans gérés par message éligibles.

Dans certaines circonstances, un message distribué à une application MDB peut être remis dans une file d'attente IBM MQ. Cette annulation peut se produire, par exemple, si un message est distribué dans une unité de travail qui est ensuite annulée. Un message annulé est à nouveau distribué, mais un message mal formaté peut entraîner à plusieurs reprises l'échec d'un bean géré par message et ne peut donc pas être distribué. Un message de ce type est appelé un message incohérent. Vous pouvez configurer IBM MQ de sorte qu' IBM MQ classes for JMS transfère automatiquement un message incohérent vers une autre file d'attente pour un examen plus approfondi ou supprime le message.

Pour plus de détails sur la gestion des messages incohérents, voir «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.

Concepts associés

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

Tâches associées

Configuration des ressources JMS dans WebSphere Application Server

Exemples de définition de la propriété sharedSubscription

Vous pouvez définir la propriété sharedSubscription d'une spécification d'activation dans un fichier WebSphere Liberty `server.xml` . Vous pouvez également définir la propriété dans un bean géré par message (MDB) à l'aide d'annotations.

Exemple: définition dans un fichier Liberty `server.xml`

Dans un fichier WebSphere Liberty `server.xml` , vous définissez une spécification d'activation comme illustré dans l'exemple suivant. Cet exemple crée un abonnement partagé durable à un gestionnaire de files d'attente sur localhost/port 1490.

```
<jmsActivationSpec id="SubApp/SubscribingEJB/SubscribingMDB" authDataRef="JMSConnectionAlias">
<properties.wmqJms hostName="localhost" port="1490" maxPoolDepth="5"
subscriptionName="MySubName"
subscriptionDurability="DURABLE" sharedSubscription="true"/>
</jmsActivationSpec>
```

Exemple: définition dans un bean géré par message

Vous pouvez également définir la propriété sharedSubscription dans le bean géré par message à l'aide d'annotations, comme illustré dans l'exemple suivant:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

L'exemple suivant illustre un fragment de code MDB qui utilise la méthode des annotations:

JM 3.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "jakarta.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {
```

```
   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

JMS 2.0

```
/**
 * Message-Driven Bean example using Annotations for configuration
 */
@MessageDriven(
  activationConfig = {
     @ActivationConfigProperty(
        propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
     @ActivationConfigProperty(
        propertyName = "sharedSubscription", propertyValue = "TRUE"),
     @ActivationConfigProperty(
        propertyName = "destination", propertyValue = "JNDI_TOPIC_NAME")
   },
   mappedName = "Stock/IBM")
public class SubscribingMDB implements MessageListener {

   // Default constructor.
   public SubscribingMDB() {
   }

   // @see MessageListener#onMessage(Message)
   public void onMessage(Message message) {
      // implement business logic here
   }

}
```

Concepts associés

Abonnés et abonnements

Durabilité des abonnements

«Abonnements clonés et partagés», à la page 335

Il existe deux méthodes pour permettre à plusieurs consommateurs d'accéder au même abonnement. Ces deux méthodes utilisent des abonnements clonés ou des abonnements partagés.

Configuration de l'adaptateur de ressources pour les communications sortantes

Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

Exemple d'utilisation de la communication sortante

Lors de l'utilisation de la communication sortante, une application exécutée dans un serveur d'applications démarre une connexion à un gestionnaire de files d'attente, puis envoie des messages à ses files d'attente et reçoit des messages de ses files d'attente de manière synchrone. Par exemple, la méthode de servlet suivante, doGet(), utilise la communication sortante:

JM 3.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (jakarta.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (jakarta.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
```

```
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

JMS 2.0

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

Lorsque le servlet reçoit une demande HTTP GET, il extrait un objet ConnectionFactory et un objet Queue de l'espace de nom JNDI et utilise les objets pour envoyer un message à une file d'attente IBM MQ . Le servlet reçoit ensuite le message qu'il a envoyé.

Ressources requises pour la communication sortante

Pour configurer la communication sortante, définissez les ressources Java EE Connector Architecture (JCA) dans les catégories suivantes:

- Propriétés d'un objet ConnectionFactory, que le serveur d'applications utilise pour créer un objet JMS ConnectionFactory .
- Propriétés d'un objet de destination géré, que le serveur d'applications utilise pour créer un objet de file d'attente JMS ou un objet de rubrique JMS .

La manière dont vous définissez ces propriétés dépend des interfaces d'administration fournies par votre serveur d'applications. Les objets ConnectionFactory, Queue et Topic créés par le serveur d'applications sont liés dans un espace de nom JNDI à partir duquel ils peuvent être extraits par une application.

En règle générale, vous définissez un objet ConnectionFactory pour chaque gestionnaire de files d'attente auquel les applications peuvent avoir besoin de se connecter. Vous définissez un objet de file d'attente pour chaque file d'attente à laquelle les applications peuvent avoir besoin d'accéder dans le domaine point à point. Vous pouvez également définir un objet de rubrique pour chaque rubrique à laquelle les applications peuvent souhaiter publier ou s'abonner. Un objet ConnectionFactory peut être indépendant du domaine. Il peut également s'agir d'un objet de fabrique QueueConnectionpour le domaine point à point ou d'un objet de fabrique TopicConnectionpour le domaine de publication / abonnement.

Conseil : Avec JMS 2.0, une fabrique de connexions peut être utilisée pour créer des connexions ainsi que des contextes. Ainsi, un pool de connexions peut être associé à une fabrique de connexions contenant un mélange de connexions et de contextes. Il est recommandé d'utiliser une fabrique de connexions pour la création de connexions ou de contextes seulement. Vous avez ainsi la garantie que le pool de connexions pour cette fabrique de connexions contient des objets d'un seul type seulement, ce qui le rend plus efficace.

Propriétés d'un objet ConnectionFactory

Tableau 66, à la page 484 répertorie les propriétés d'un objet ConnectionFactory . Le serveur d'applications utilise ces propriétés pour créer un objet JMS ConnectionFactory .

Tableau 66. Propriétés d'un objet ConnectionFactory

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
applicationName	String	• Le nom de la classe appelante, s'il est disponible, est ajusté pour ne pas dépasser 28 caractères. S'il n'est pas disponible, la chaîne `WebSphere MQ Client for Java` est utilisée.	Nom sous lequel une application est enregistrée auprès du gestionnaire de files d'attente. Ce nom d'application est affiché par la commande **DISPLAY CONN MQSC/PCF** (où la zone est appelée **APPLTAG**) ou dans l'affichage **Connexions d'application** de l'explorateur IBM MQ (où la zone est appelée **App name**).
File d'attente brokerCCSub	String	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de connexion reçoit des messages d'abonnement non durable.
File d'attente brokerControl	String	• **SYSTEM.BROKER.CONTROL.QUEUE** • Un nom de file d'attente	Nom de la file d'attente de contrôle du courtier.
File d'attente brokerPub	String	• **SYSTEM.BROKER.DEFAULT.STREAM** • Un nom de file d'attente	Nom de la file d'attente dans laquelle les messages publiés sont envoyés (file d'attente de flux).
Gestionnaire brokerQueue	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente sur lequel le courtier s'exécute.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
File d'attente brokerSub	String	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de message non durable reçoit des messages. Pour plus d'informations, voir la propriété BROKERSUBQ .
brokerVersion	String	• **unspecified** -Une fois que le courtier a été migré de V6 vers V7, définissez cette propriété de sorte que les en-têtes RFH2 ne soient plus utilisés. Après la migration, cette propriété n'est plus pertinente. • **V1** -Pour utiliser un courtier de publication / abonnement IBM MQ . Cette valeur est la valeur par défaut si TRANSPORT est défini sur BIND ou CLIENT. • **V2** -Pour utiliser un courtier de IBM Integration Bus en mode natif. Cette valeur est la valeur par défaut si TRANSPORT est défini sur DIRECT ou DIRECTIVE THTTP.	Version du courtier utilisé.
ccdtURL	String	• **null** • Un URL (uniform resource locator)	URL qui identifie le nom et l'emplacement du fichier contenant la table de définition de canal du client (CCDT) et qui indique comment accéder au fichier.
CCSID	String	• **819** • Identificateur de jeu de caractères codés pris en charge par la machine virtuelle Java (JVM)	Identificateur de jeu de caractères codés pour une connexion.
canal	String	• **SYSTEM.DEF.SVRCONN** • Nom d'un canal MQI	Nom du canal MQI à utiliser.
cleanupInterval	int	• **3 600 000** • Entier positif	Intervalle, en millisecondes, entre les exécutions en arrière-plan de l'utilitaire de nettoyage de publication / abonnement.
cleanupLevel	String	• **SECURISEE** • Aucun • FONDATION • FORCE • NONDUR	Niveau de nettoyage d'un magasin d'abonnements basé sur un courtier.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
clientID	String	• **null** • Un identificateur de client	Identificateur client pour une connexion.
cloneSupport	String	• **DISABLED** -Une seule instance d'un abonné durable à la rubrique peut s'exécuter à la fois. • ENABLED-Deux ou plusieurs instances du même abonné durable à la rubrique peuvent s'exécuter simultanément, mais chaque instance doit s'exécuter dans une machine virtuelle Java (JVM) distincte.	Indique si deux instances ou plus du même abonné durable à la rubrique peuvent s'exécuter simultanément.
ConnectionNameList	String	• **localhost (1414)** • Chaîne composée d'éléments séparés par des virgules, où chaque élément prend le format suivant: `HOSTNAME(PORT)` où *HOSTNAME* est un nom DNS ou une adresse IP.	Liste des noms de connexion TCP/IP utilisés pour les communications sortantes. **connectionNameList** remplace les propriétés **hostname** et **port** . Cette propriété permet de se reconnecter aux gestionnaires de files d'attente multi-instance. La forme de **connectionNameList** est similaire à **localAddress**, mais ne doit pas être confondue avec celle-ci. **localAddress** indique les caractéristiques des communications locales, tandis que **connectionNameList** indique comment atteindre un gestionnaire de files d'attente éloignées.
failIfQuiesce	Boolean	• **true** • false	Indique si les appels de certaines méthodes échouent si le gestionnaire de files d'attente est à l'état de mise au repos.
headerCompression	String	• **Aucun** • La compression de l'en-tête de message SYSTEM-RLE est effectuée.	Liste des techniques pouvant être utilisées pour compresser les données d'en-tête sur une connexion.
hostName	String	• **système hôte local** • Un nom d'hôte • Une adresse IP	Nom d'hôte ou adresse IP du système sur lequel réside le gestionnaire de files d'attente. Les propriétés **hostname** et **port** sont remplacées par la propriété **connectionNameList** lorsqu'elle est spécifiée.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
localAddress	String	• **null** • Chaîne au format: `[ host_name ][(low_port [, high_port ])]` où *nom_hôte* est un nom d'hôte ou une adresse IP, *port_inférieur* et *port_supérieur* sont des numéros de port TCP et les crochets indiquent un composant facultatif	Pour une connexion à un gestionnaire de files d'attente, cette propriété spécifie l'un des éléments suivants ou les deux: • Interface réseau locale à utiliser • Port local, ou plage de ports locaux, à utiliser La forme de **localAddress** est similaire à **connectionNameList**, mais ne doit pas être confondue avec celle-ci. **localAddress** indique les caractéristiques des communications locales, tandis que **connectionNameList** indique comment atteindre un gestionnaire de files d'attente éloignées.
messageCompression	String	• **Aucun** • Liste d'une ou de plusieurs des valeurs suivantes, séparées par des caractères blancs: RLE ZLIBFAST ZLIBHIGH V 9.4.0 LZ4FAST V 9.4.0 LZ4HIGH	Liste des techniques pouvant être utilisées pour compresser les données de message sur une connexion.
messageSelection	String	• **client** • COURTIER	Détermine si la sélection des messages est effectuée par IBM MQ classes for JMS ou par le courtier. La sélection de messages par le courtier n'est pas prise en charge lorsque **brokerVersion** a la valeur 1.
Mot de passe	String	• **null** • Un mot de passe	Mot de passe par défaut à utiliser lors de la création d'une connexion au gestionnaire de files d'attente.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
pollingInterval	int	• **5000** • Tout entier positif	Lorsque la file d'attente associée aux différents programmes d'écoute de message dans une session ne contient aucun message approprié, cette valeur est l'intervalle de temps maximal, en millisecondes, qui peut s'écouler avant que chaque programme d'écoute tente à nouveau d'extraire un message de sa file d'attente. Si l'absence de message approprié est fréquemment observée pour l'un quelconque des écouteurs de messages au sein d'une session, envisagez d'augmenter la valeur de cette propriété. Cette propriété est pertinente uniquement si **TRANSPORT** a la valeur `BIND` ou `CLIENT`.
port	int	• **1414** • Un numéro de port TCP	Port sur lequel le gestionnaire de files d'attente écoute. Les propriétés **hostname** et **port** sont remplacées par la propriété **connectionNameList** lorsqu'elle est spécifiée.
providerVersion	chaîne	• **non spécifié** • Une chaîne dans l'un des formats suivants – V.R.M.F – V.R.M – V.R – V où V, R, M et F sont des valeurs entières supérieures ou égales à zéro.	Version, édition, niveau de modification et groupe de correctifs du gestionnaire de files d'attente auquel l'application a l'intention de se connecter.
Intervalle pubAck	int	• **25** • Entier positif	Nombre de messages publiés par un diffuseur de publications avant que IBM MQ classes for JMS ne demande un accusé de réception au courtier.
queueManager	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente auquel établir la connexion.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
receiveExit [3]	String	• **null** • Chaîne comprenant un ou plusieurs éléments séparés par des virgules, où chaque élément correspond au nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQReceiveExit	Identifie un programme d'exit de réception de canal ou une séquence de programmes d'exit de réception à exécuter successivement.
receiveExitInitialisation	String	• **null** • Chaîne comprenant un ou plusieurs éléments de données utilisateur séparés par des virgules	Données utilisateur transmises aux programmes d'exit de réception de canal lorsqu'ils sont appelés.
rescanInterval	int	• **5000** • Tout entier positif	Lorsqu'un consommateur de message du domaine point à point utilise un sélecteur de message pour sélectionner les messages qu'il souhaite recevoir, IBM MQ classes for JMS recherche dans la file d'attente IBM MQ les messages appropriés dans la séquence déterminée par l'attribut **MsgDeliverySequence** de la file d'attente. Lorsque IBM MQ classes for JMS trouve un message approprié et le distribue au consommateur, IBM MQ classes for JMS reprend la recherche du message approprié suivant à partir de sa position actuelle dans la file d'attente. IBM MQ classes for JMS continue de rechercher la file d'attente de cette manière jusqu'à ce qu'elle atteigne la fin de la file d'attente ou jusqu'à ce que l'intervalle de temps en millisecondes, déterminé par la valeur de cette propriété, ait expiré. Dans chaque cas, IBM MQ classes for JMS revient au début de la file d'attente pour poursuivre sa recherche et un nouvel intervalle de temps commence.
securityExit [3]	String	• **null** • Nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQSecurityExit	Identifie un programme d'exit de sécurité de canal.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
securityExitInit	String	• **null** • Une chaîne de données utilisateur	Données utilisateur transmises à un programme d'exit de sécurité de canal lorsqu'il est appelé.
SENDCHECKCOUNT	int	• **0** • Tout entier positif	Nombre d'appels d'envoi à autoriser entre deux vérifications d'erreurs d'insertion asynchrone, au sein d'une même session JMS non transactionnelle.
sendExit [3]	String	• **null** • Chaîne comprenant un ou plusieurs éléments séparés par des virgules, où chaque élément est le nom qualifié complet d'une classe qui implémente l'interface IBM MQ classes for Java , MQSendExit	Identifie un programme d'exit d'émission de canal ou une séquence de programmes d'exit d'émission à exécuter successivement.
SENDEXITINIT	String	• **null** • Chaîne comprenant un ou plusieurs éléments de données utilisateur séparés par des virgules	Données utilisateur transmises aux programmes d'exit d'émission de canal lorsqu'ils sont appelés.
SHARECONVALLOWED	Boolean	• NON-Une connexion client ne peut pas partager son socket. • **YES** -Une connexion client peut partager son socket.	Indique si une connexion client peut partager son socket avec d'autres connexions JMS de niveau supérieur à partir du même processus dans un même gestionnaire de files d'attente, si les définitions de canal correspondent.
sparseSubscriptions	Boolean	• **false** -Les abonnements reçoivent des messages de correspondance fréquents. • true-Les abonnements reçoivent des messages correspondants peu fréquents. Cette valeur nécessite que la file d'attente d'abonnement puisse être ouverte pour l'exploration.	Contrôle la stratégie d'extraction de message d'un objet TopicSubscriber .
sslCertMagasins	String	• **null** • Chaîne d'une ou de plusieurs URL LDAP séparées par des blancs. Chaque URL LDAP a le format suivant: `ldap://host_name [: port ]` où *nom_hôte* est un nom d'hôte ou une adresse IP, *port* est un numéro de port TCP et les crochets indiquent un composant facultatif.	Serveurs LDAP (Lightweight Directory Access Protocol) qui contiennent des listes de révocation de certificat (CRL) à utiliser sur une connexion TLS.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
SSLCIPHERSUITE	String	• **null** • Nom d'une CipherSuite	CipherSuite à utiliser pour une connexion TLS.
sslFipsObligatoire [2]	Boolean	• **faux** • Oui	Indique si une connexion TLS doit utiliser une CipherSuite prise en charge par le fournisseur IBM Java JSSE FIPS (IBMJSSEFIPS).
SSLPEERNAME	String	• **null** • Modèle pour les noms distinctifs	Pour une connexion TLS, modèle utilisé pour vérifier le nom distinctif dans le certificat numérique fourni par le gestionnaire de files d'attente.
SSLRESETCOUNT	int	• **0** • Entier compris entre 0 et 999 999 999	Nombre total d'octets envoyés et reçus par une connexion TLS avant que les clés secrètes utilisées par TLS ne soient renégociées.
Fabrique sslSocket	String	Chaîne représentant le nom de classe qualifié complet d'une classe fournissant une implémentation de l'interface javax.net.ssl.SSLSocketFactory , incluant éventuellement un argument à transmettre à la méthode du constructeur, entre parenthèses.	Toutes les connexions établies dans la portée de l'objet de destination administré utilisent des sockets obtenus à partir de cette implémentation de l'interface SSLSocketFactory .
Intervalle statusRefresh	int	• **60000** • Tout entier positif	Intervalle, en millisecondes, entre les actualisations de la transaction à exécution longue qui détecte lorsqu'un abonné perd sa connexion au gestionnaire de files d'attente. Cette propriété est pertinente uniquement si **SUBSTORE** a la valeur QUEUE.
subscriptionStore	String	• **Courtier** • MIGRATE • QUEUE	Détermine l'emplacement où IBM MQ classes for JMS stocke les données persistantes sur les abonnements actifs.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
Correspondance targetClient	Booleen	• **true** • false	Indique si un message de réponse envoyé à la file d'attente identifiée par la zone d'en-tête JMSReplyTo d'un message entrant comporte un en-tête MQRFH2 uniquement si le message entrant comporte un en-tête MQRFH2 . Vous pouvez également configurer cette propriété pour une spécification d'activation. Pour plus d'informations, voir «Configuration de la propriété de correspondance targetClientpour une spécification d'activation», à la page 501.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
temporaryModel	String	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • N'importe quelle chaîne	Nom de la file d'attente modèle à partir de laquelle les files d'attente temporaires JMS sont créées. Utilisez SYSTEM.DEFAULT.MODEL.QUEUE si les deux instructions suivantes sont vraies: • Votre application utilise une file d'attente temporaire qui accepte les messages non persistants. • Une seule application crée une file d'attente temporaire sur le gestionnaire de files d'attente vers lequel la ConnectionFactory pointe à la fois. Notez que SYSTEM.DEFAULT.MODEL.QUEUE ne peut être ouvert que par une seule application à la fois. Utilisez SYSTEM.JMS.TEMPQ.MODEL dans les situations suivantes: • Lorsque votre application utilise une file d'attente temporaire qui accepte les messages persistants. • Si plusieurs applications peuvent se connecter au gestionnaire de files d'attente vers lequel la ConnectionFactory pointe et que ces applications doivent créer des files d'attente temporaires en même temps. Définissez une nouvelle file d'attente modèle avec l'attribut **DEFPSIST** défini sur YESet l'attribut **DEFSOPT** défini sur SHARED dans la situation suivante: • Lorsque votre application utilise une file d'attente temporaire qui accepte les messages non persistants et que plusieurs applications se connectent au gestionnaire de files d'attente vers lequel pointe ConnectionFactory , ces applications doivent créer des files d'attente temporaires en même temps. Lorsque la nouvelle file d'attente modèle est créée, définissez la propriété **temporaryModel** sur le nom de la nouvelle file d'attente modèle.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
tempQPrefix	String	• **"" (chaîne vide)** • Préfixe pouvant être utilisé pour former le nom d'une file d'attente dynamique IBM MQ . Les règles de formation du préfixe sont les mêmes que les règles de formation du contenu de la zone **DynamicQName** dans un descripteur d'objet IBM MQ , structure MQOD, mais le dernier caractère non blanc doit être un astérisque (*). Si la valeur de la propriété est la chaîne vide, IBM MQ classes for JMS utilise la valeur AMQ.* lors de la création d'une file d'attente dynamique.	Préfixe utilisé pour former le nom d'une file d'attente dynamique IBM MQ .
TEMPTOPICPREFIX	String	Toute chaîne non nulle constituée uniquement de caractères valides pour une chaîne de rubrique IBM MQ	Lors de la création de rubriques temporaires, JMS génère une chaîne de rubrique au format "TEMP/*TEMPTOPICPREFIX*/*id_unique* " ou, si cette propriété est laissée avec la valeur par défaut, uniquement "TEMP/*id_unique*". La spécification d'un **TEMPTOPICPREFIX** non vide permet de définir des files d'attente modèles spécifiques pour la création de files d'attente gérées pour les abonnés à des rubriques temporaires créées sous cette connexion.

Tableau 66. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
transportType	String	• **client** • LIAISONS • BINDINGS_THEN_CLIENT	Indique si une connexion à un gestionnaire de files d'attente utilise le mode client ou le mode liaisons. Si la valeur `BINDINGS_THEN_CLIENT` est spécifiée, l'adaptateur de ressources tente d'abord d'établir une connexion en mode liaisons. Si cette tentative de connexion échoue, l'adaptateur de ressources tente d'établir une connexion en mode client. z/OS Si une spécification d'activation qui s'exécute sur un système WebSphere Application Server for z/OS a été configurée pour utiliser le mode de transport `BINDINGS_THEN_CLIENT` et qu'une connexion établie précédemment est interrompue, toutes les tentatives de reconnexion effectuées par la spécification d'activation tentent d'abord d'utiliser le mode de transport `BINDINGS` . Si la tentative de connexion en mode transport `BINDINGS` échoue, la spécification d'activation tente ensuite une connexion en mode transport `CLIENT` .
username	String	• **null** • Nom d'utilisateur	Nom d'utilisateur par défaut à utiliser lors de la création d'une connexion à un gestionnaire de files d'attente.
wildcardFormat	int	• CHAR-Reconnaissance des caractères génériques uniquement, tels qu'ils sont utilisés dans la version 1 du courtier • SUJET-Reconnaît uniquement les caractères génériques de niveau rubrique, tels qu'ils sont utilisés dans la version 2 du courtier	Version de la syntaxe des caractères génériques à utiliser.

Remarques :

1. Pour des informations importantes sur l'utilisation de la propriété sslFipsRequired, voir «Limitations de l'adaptateur de ressources IBM MQ», à la page 450.
2. Pour plus d'informations sur la configuration de l'adaptateur de ressources afin qu'il puisse localiser un exit, voir «Configuration de IBM MQ classes for JMS pour l'utilisation des exits de canal», à la page 287.

L'exemple suivant illustre un ensemble typique de propriétés d'un objet ConnectionFactory :

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Propriétés d'un objet de destination géré

Le serveur d'applications utilise les propriétés d'un objet de destination géré pour créer un objet JMS Queue ou JMS Topic.

Le Tableau 67, à la page 496 répertorie les propriétés communes à un objet Queue et à un objet Topic.

Tableau 67. Propriétés communes à un objet Queue et à un objet Topic

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
CCSID	String	• **1208** • Identificateur de jeu de caractères codés pris en charge par la machine virtuelle Java (JVM)	Identificateur de jeu de caractères codés pour la destination.
codage	String	• **native** • Une chaîne de trois caractères: – Le premier caractère indique la représentation des entiers binaires: - *N* indique un codage normal. - *R* indique un codage inverse. – Le deuxième caractère indique la représentation des entiers décimaux condensés: - *N* indique un codage normal. - *R* indique un codage inverse. – Le troisième caractère indique la représentation des nombres en virgule flottante: - *N* indique le codage IEEE standard. - *R* indique le codage IEEE inverse. - *3* indique le codage zSeries . NATIVE est équivalent à la chaîne NNN.	Représentation des entiers binaires, des entiers décimaux condensés et des nombres à virgule flottante pour la destination.
expiration	String	• **APP** -L'heure d'expiration d'un message est déterminée par l'expéditeur de message. • UNLIM-Un message n'expire jamais. • 0-Un message n'expire jamais. • Entier positif représentant le délai d'expiration d'un message en millisecondes.	Heure d'expiration d'un message envoyé à la destination.

Tableau 67. Propriétés communes à un objet Queue et à un objet Topic (suite)

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
failIfQuiesce	String	• **true** • false	Indique si une tentative d'accès à la destination échoue si le gestionnaire de files d'attente est à l'état de mise au repos.
Style messageBody	String	• **NON SPECIFIE** • JMS • MQ	Vous pouvez définir la propriété **messageBodyStyle** sur les files d'attente et les rubriques JMS : UNSPECIFIED (par défaut) • Lors de l'envoi, IBM MQ classes for JMS génère et inclut un en-tête MQRFH2 , en fonction de la valeur de WMQ_TARGET_CLIENT. • Lors de la réception, IBM MQ classes for JMS définissez les propriétés de message JMS en fonction des valeurs dans MQRFH2, le cas échéant. MQRFH2 n'est pas présenté dans le corps du message JMS . JMS • Lors de l'envoi, IBM MQ classes for JMS génère automatiquement un en-tête MQRFH2 et l'inclut dans le message IBM MQ . • Lors de la réception, IBM MQ classes for JMS définissez les propriétés de message JMS en fonction des valeurs dans MQRFH2, le cas échéant. MQRFH2 n'est pas présenté dans le corps du message JMS . MQ • Lors de l'envoi, IBM MQ classes for JMS ne générez pas de MQRFH2. • Lors de la réception, IBM MQ classes for JMS présente MQRFH2 dans le corps du message JMS .

Tableau 67. Propriétés communes à un objet Queue et à un objet Topic (suite)

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
persistance	String	• **APP** -La persistance d'un message est déterminée par l'expéditeur de message. • QDEF-La persistance d'un message est déterminée par l'attribut **DefPersistence** de la file d'attente IBM MQ . • PERS-Un message est persistant. • NON-Un message est non persistant. • HIGH-La persistance d'un message est déterminée par l'attribut **NonPersistentMessageClass** de la file d'attente IBM MQ en fonction de l'explication fournie dans «JMS messages persistants», à la page 260.	Persistance d'un message envoyé à la destination.
priority	String	• **APP** -La priorité d'un message est déterminée par l'expéditeur de message. • QDEF-La priorité d'un message est déterminée par l'attribut **DefPriority** de la file d'attente IBM MQ . • Entier compris entre 0, priorité la plus faible et 9, priorité la plus élevée.	Priorité d'un message envoyé à la destination.
PUTASYNCALLOWED	String	• QUEUE-Déterminer si les insertions asynchrones sont autorisées en faisant référence à la définition de file d'attente. • SUJET-Déterminer si les insertions asynchrones sont autorisées en faisant référence à la définition de rubrique. • DESTINATION-Déterminez si les insertions asynchrones sont autorisées en faisant référence à la définition de file d'attente ou de rubrique. • DISABLED-Les insertions asynchrones ne sont pas autorisées. • ENABLED-Les insertions asynchrones sont autorisées.	Indique si les expéditeurs de messages sont autorisés à utiliser des insertions asynchrones pour envoyer des messages à cette destination.

Tableau 67. Propriétés communes à un objet Queue et à un objet Topic (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
READAHEADALLOWED	int	• **DESTINATION** -Déterminez si la lecture anticipée est autorisée en faisant référence à la définition de la file d'attente ou de la rubrique. • DISABLED-La lecture anticipée n'est pas autorisée. • ENABLED-La lecture anticipée est autorisée. • QUEUE-Déterminer si la lecture anticipée est autorisée en faisant référence à la définition de file d'attente. • SUJET-Déterminer si la lecture anticipée est autorisée en faisant référence à la définition de rubrique.	Indique si les consommateurs de messages et les navigateurs de files d'attente sont autorisés à utiliser la lecture anticipée pour extraire des messages non persistants de la destination dans une mémoire tampon interne avant de les recevoir.
receiveCCSID	int	• **0** -Utiliser la machine virtuelle Java `Charset.defaultCharset` • 1208- UTF-8 • Identificateur de jeu de caractères codés pris en charge	Propriété de destination qui définit le CCSID cible pour la conversion des messages du gestionnaire de files d'attente. La valeur est ignorée sauf si **`receiveConversion`** est défini sur `QMGR`.
receiveConversion	String	• **CLIENT_MSG** • QMGR	Propriété de destination qui détermine si la conversion de données sera effectuée par le gestionnaire de files d'attente.
targetClient	String	• **JMS** -La cible d'un message est une application JMS . • MQ -La cible d'un message est une application nonJMS IBM MQ .	Indique si la cible d'un message envoyé à la destination est une application JMS . Un message avec une cible qui est une application JMS contient un en-tête MQRFH2 .

Le Tableau 68, à la page 499 répertorie les propriétés spécifiques à un objet Queue.

Tableau 68. Propriétés spécifiques à un objet Queue

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
baseQueueManagerName	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente qui possède la file d'attente IBM MQ sous-jacente.
Nom baseQueue	String	• **"" (chaîne vide)** • Un nom de file d'attente	Nom de la file d'attente IBM MQ sous-jacente.

Le Tableau 69, à la page 500 répertorie les propriétés spécifiques à un objet de rubrique.

Tableau 69. Propriétés spécifiques à un objet de rubrique

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
Nom baseTopic	String	• **"" (chaîne vide)** • Un nom de rubrique	Nom de la rubrique sous-jacente.
brokerCCDurSubQueue >	String	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un consommateur de connexion reçoit des messages d'abonnement durable.
brokerDurSubQueue	String	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Un nom de file d'attente	Nom de la file d'attente à partir de laquelle un abonné de rubrique durable reçoit des messages. Pour plus d'informations, voir la propriété **BROKEDURRSUBQ** dans la documentation IBM MQ Explorer .
File d'attente brokerPub	String	• **Non définie** • Un nom de file d'attente	Nom de la file d'attente dans laquelle les messages publiés sont envoyés (file d'attente de flux). La valeur de cette propriété remplace la valeur de la propriété **brokerPubQueue** de l'objet ConnectionFactory . Toutefois, si vous ne définissez pas la valeur de cette propriété, la valeur de la propriété **brokerPubQueue** de l'objet ConnectionFactory est utilisée à la place.
brokerPubQueueManager	String	• **"" (chaîne vide)** • Un nom de gestionnaire de files d'attente	Nom du gestionnaire de files d'attente propriétaire de la file d'attente dans laquelle les messages publiés sur la rubrique sont envoyés.
brokerVersion	String	• **Non définie** • 1 • 2	Version du courtier utilisé. La valeur de cette propriété remplace la valeur de la propriété **brokerVersion** de l'objet ConnectionFactory . Toutefois, si vous ne définissez pas la valeur de cette propriété, la valeur de la propriété **brokerVersion** de l'objet ConnectionFactory est utilisée à la place.

L'exemple suivant illustre un ensemble de propriétés d'un objet Queue:

```
expiry:               UNLIM
persistence:          QDEF
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.JMS.TEMPQ.MODEL
```

L'exemple suivant illustre un ensemble de propriétés d'un objet Topic:

```
expiry:               UNLIM
persistence:          NON
baseTopicName:        myTestTopic
```

Tâches associées

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client

Configuration des ressources JMS dans WebSphere Application Server

Référence associée

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

Configuration de la propriété de correspondance targetClientpour une spécification d'activation

Vous pouvez configurer la propriété **targetClientMatching** pour une spécification d'activation de sorte qu'un en-tête MQRFH2 soit inclus dans les messages de réponse lorsque les messages de demande ne contiennent pas d'en-tête MQRFH2 . En d'autres termes, toutes les propriétés de message qu'une application définit pour un message de réponse sont incluses lorsque le message est envoyé.

Pourquoi et quand exécuter cette tâche

Si une application de bean géré par message (MDB) consomme des messages qui ne contiennent pas d'en-tête MQRFH2 , via une spécification d'activation d'adaptateur de ressources JCA IBM MQ , et envoie ensuite des messages de réponse à la destination JMS créée à partir de la zone `JMSReplyTo` du message de demande, les messages de réponse doivent inclure un en-tête MQRFH2 , même si les messages de demande ne le sont pas, sinon les propriétés de message que l'application a définies sur un message de réponse sont perdues.

La propriété **targetClientMatching** définit si un message de réponse envoyé à la file d'attente identifiée par la zone d'en-tête JMSReplyTo d'un message entrant comporte un en-tête MQRFH2 uniquement si le message entrant comporte un en-tête MQRFH2 . Vous pouvez configurer cette propriété pour une spécification d'activation, à la fois dans WebSphere Application Server traditional et dans WebSphere Liberty.

Si vous définissez la valeur de la propriété **targetClientMatching** sur `false`, un en-tête MQRFH2 peut être inclus dans un message de réponse envoyé à une destination JMS créée à partir de l'en-tête JMSReplyTo d'un message de demande entrant qui ne contient pas de MQRFH2. En effet, la propriété **targetClient** de la destination JMS est définie sur la valeur `0`, ce qui signifie que les messages contiennent un en-tête MQRFH2 . La présence de l'en-tête MQRFH2 dans le message sortant permet de stocker les propriétés de message définies par l'utilisateur dans le message lorsqu'il est envoyé à la file d'attente IBM MQ .

Si la propriété **targetClientMatching** est définie sur `true` et qu'un message de demande n'inclut pas d'en-tête MQRFH2 , un en-tête MQRFH2 n'est pas inclus dans le message de réponse.

Procédure

- Dans WebSphere Application Server traditional, utilisez la console d'administration pour définir la propriété **targetClientMatching** en tant que propriété personnalisée sur la spécification d'activation IBM MQ :
  a) Dans le panneau de navigation, cliquez sur **Ressources-> JMS-> Spécifications d'activation**.
  b) Sélectionnez le nom de la spécification d'activation que vous souhaitez afficher ou modifier.

c) Cliquez sur **Propriétés personnalisées-> Nouveau** , puis entrez les détails de la nouvelle propriété personnalisée.

Définissez le nom de la propriété sur `targetClientMatching`, le type sur `java.lang.Boolean` et la valeur sur `false`.

- Dans WebSphere Liberty, spécifiez la propriété **`targetClientMatching`** sur la définition d'une spécification d'activation dans `server.xml`.

  Exemple :

```
<jmsActivationSpec id="SimpleMDBApplication/SimpleEchoMDB/SimpleEchoMDB">
<properties.wmqJms destinationRef="MDBRequestQ"
queueManager="MY_QMGR" transportType="BINDINGS" targetClientMatching="false"/>
<authData password="********" user="tom"/>
</jmsActivationSpec>
```

Concepts associés

«Création de destinations dans une application JMS», à la page 227
Au lieu d'extraire des destinations en tant qu'objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), une application JMS peut utiliser une session pour créer des destinations de manière dynamique lors de l'exécution. Une application peut utiliser un identificateur URI (Uniform Resource Identifier) pour identifier une file d'attente IBM MQ ou une rubrique et, éventuellement, pour spécifier une ou plusieurs propriétés d'un objet de file d'attente ou de rubrique.

«Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 482
Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

Suspension du bean géré par message IBM MQ dans WebSphere Liberty

La propriété **`maxSequentialDeliveryFailures`** d'une spécification d'activation définit le nombre maximal d'échecs de distribution séquentielle de messages à une instance de bean géré par message (MDB) que l'adaptateur de ressources tolère avant de mettre en pause le MDB.

Avant de commencer

Vous devez être conscient de l'ensemble d'événements qui peuvent entraîner la mise en pause d'un bean géré par message dans WebSphere Liberty. L'adaptateur de ressources considère l'un des éléments suivants comme un échec de distribution de message:

- Une exception non vérifiée a été émise à partir de la méthode **`onMessage`** du bean géré par message.
- Une erreur `JMSException` se produit lors du traitement de l'adaptateur de ressources, avant la distribution du message au bean géré par message.
- Une erreur `JMSException` se produit lors du traitement de l'adaptateur de ressources, après la distribution du message au bean géré par message.
- Transaction XA, ou transaction locale, utilisée pour consommer le message en cours d'annulation.
- Aucune unité d'exécution n'est disponible dans le serveur d'applications pour distribuer le message au bean géré par message.

Pourquoi et quand exécuter cette tâche

La valeur par défaut de la propriété **`maxSequentialDeliveryFailures`** est *-1*, ce qui signifie que le bean géré par message n'est jamais mis en pause. Toute autre valeur négative est traitée de la même manière que *-1*. Une valeur de:

- *0* signifie que le bean géré par message s'interrompt lors de la première erreur
- *1* signifie que le bean géré par message s'arrête sur deux erreurs séquentielles
- *2* signifie que le bean géré par message s'arrête sur trois erreurs séquentielles, etc.

Vous pouvez configurer cette propriété pour une spécification d'activation, uniquement dans WebSphere Libertyet lorsque le niveau de Liberty est 18.0.0.4ou supérieur.

Avertissement : Si vous définissez l'attribut sur une valeur autre que la valeur par défaut dans un environnement de serveur d'applications autre que Liberty, la valeur est ignorée et un message d'avertissement est consigné dans le journal.

En outre, il est possible d'installer l'adaptateur de ressources IBM MQ dans WebSphere Liberty en tant qu'adaptateur de ressources générique. Cette opération désactive toutes les fonctions d'intégration IBM MQ et WebSphere Application Server et empêche l'adaptateur de ressources de détecter qu'il est en cours d'exécution dans Liberty. Par conséquent, la définition de **maxSequentialDeliveryFailures** sur une valeur supérieure ou égale à *0* n'est pas prise en charge et génère un message d'avertissement dans le journal.

Procédure

- Dans WebSphere Liberty, spécifiez la propriété **maxSequentialDeliveryFailures** sur la définition d'une spécification d'activation dans `server.xml`.

  Exemple :

```
<jmsActivationSpec>
     <properties.wmqJms destinationRef="jndi/MDBQ"
                        transportType="BINDINGS"
                        queueManager="MQ21"
                        maxSequentialDeliveryFailures="1"/>
</jmsActivationSpec>
```

Concepts associés

«Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 482
Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

Vérification de l'installation de l'adaptateur de ressources

Le programme de test de vérification de l'installation (IVT) de l'adaptateur de ressources IBM MQ est fourni sous la forme d'un fichier EAR. Pour utiliser le programme, vous devez le déployer et définir certains objets en tant que ressources JCA .

Pourquoi et quand exécuter cette tâche

Le programme de test de vérification de l'installation (IVT) est fourni sous la forme d'un fichier d'archive d'entreprise (EAR) appelé `wmq.jakarta.jmsra.ivt.ear` (Jakarta Messaging 3.0) ou `wmq.jmsra.ivt.ear` (JMS 2.0). Ce fichier est installé avec IBM MQ classes for JMS dans le même répertoire que le fichier RAR de l'adaptateur de ressources IBM MQ , `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) ou `wmq.jmsra.rar` (JMS 2.0). Pour plus d'informations sur l'emplacement d'installation de ces fichiers, voir «Installation de l'adaptateur de ressources IBM MQ», à la page 454.

Vous devez déployer le programme IVT sur votre serveur d'applications. Le programme IVT inclut un servlet et un bean géré par message qui teste qu'un message peut être envoyé et reçu d'une file d'attente IBM MQ . Vous pouvez utiliser le programme IVT pour vérifier que l'adaptateur de ressources IBM MQ a été correctement configuré pour prendre en charge les transactions réparties. Si vous déployez l'adaptateur de ressources IBM MQ dans un serveur d'applications nonIBM , le service IBM peut vous demander de démontrer que le test IVT fonctionne pour vérifier que votre serveur d'applications est correctement configuré.

Avant de pouvoir exécuter le programme IVT, vous devez définir un objet ConnectionFactory , un objet Queue et éventuellement un objet Activation Specification en tant que ressources JCA , et vous assurer que votre serveur d'applications crée des objets JMS à partir de ces définitions et les lie à un espace de nom JNDI . Vous pouvez choisir les propriétés des objets qui correspondent aux paramètres d'hôte et de port de votre propre QueueManager, mais l'ensemble de propriétés suivant est un exemple simple:

```
ConnectionFactory object:
channel:              SYSTEM.DEF.SVRCONN
hostName:             localhost
```

```
port:                 1550
queueManager:         QM1
transportType:        CLIENT
Queue object:
baseQueueManagerName: QM1
baseQueueName:        TEST.QUEUE
```

Le mécanisme utilisé pour définir les objets ConnectionFactory, Queue et Activation Specification varie en fonction de votre serveur d'applications. Par exemple, pour définir ces propriétés dans WebSphere Liberty, ajoutez les entrées suivantes au fichier `server.xml` du serveur d'applications:

JMS 3.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jakarta.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

JMS 2.0
```
<!-- IVT Connection factory -->
<jmsQueueConnectionFactory connectionManagerRef="ConMgrIVT" jndiName="IVTCF">
   <properties.wmqJms channel="SYSTEM.DEF.SVRCONN" hostname="localhost" port="1550"
transportType="CLIENT"/>
</jmsQueueConnectionFactory>
<connectionManager id="ConMgrIVT" maxPoolSize="10"/>

<!-- IVT Queues -->
<jmsQueue id="IVTQueue" jndiName="IVTQueue">
   <properties.wmqJms baseQueueName="TEST.QUEUE"/>
</jmsQueue>

<!-- IVT Activation Spec -->
<jmsActivationSpec id="wmq.jmsra.ivt/WMQ_IVT_MDB/WMQ_IVT_MDB">
   <properties.wmqJms destinationRef="IVTQueue"
transportType="CLIENT"
queueManager="QM1"
hostName="localhost"
port="1550"
maxPoolDepth="1"/>
</jmsActivationSpec>
```

Par défaut, le programme IVT attend qu'un objet ConnectionFactory soit lié dans l'espace de nom JNDI avec le nom jms / ivt/IVTCF et qu'un objet Queue soit lié avec le nom jms / ivt/IVTQueue. Vous pouvez utiliser des noms différents, mais dans ce cas, vous devez entrer les noms des objets sur la page initiale du programme IVT et modifier le fichier EAR de manière appropriée.

Une fois que vous avez déployé le programme IVT et que le serveur d'applications a créé les objets JMS et les a liés dans l'espace de nom JNDI , vous pouvez démarrer le programme IVT en procédant comme suit.

Procédure

1. Démarrez le programme IVT en entrant une URL au format suivant dans votre navigateur Web:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

où *app_server_host* est l'adresse IP ou le nom d'hôte du système sur lequel votre serveur d'applications s'exécute et *port* est le numéro du port TCP sur lequel le serveur d'applications est en mode écoute. Par exemple :

```
http://localhost:9080/WMQ_IVT/
```

Voici un exemple de la page initiale affichée par le programme IVT.

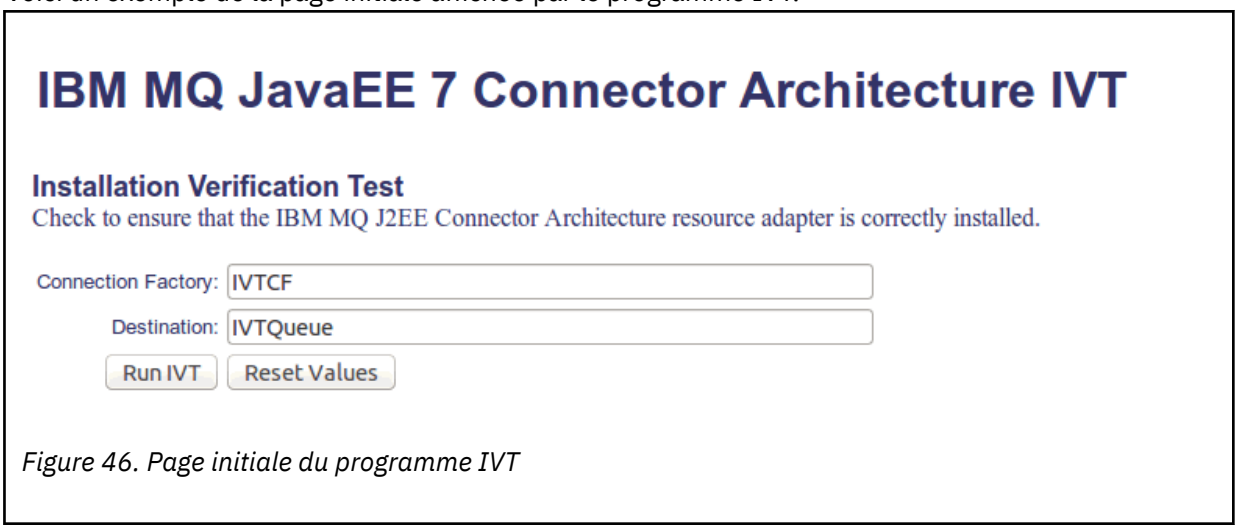


Figure 46. Page initiale du programme IVT

2. Pour exécuter le test, cliquez sur **Exécuter IVT**.

   Voici un exemple de la page qui s'affiche si l'IVT aboutit.

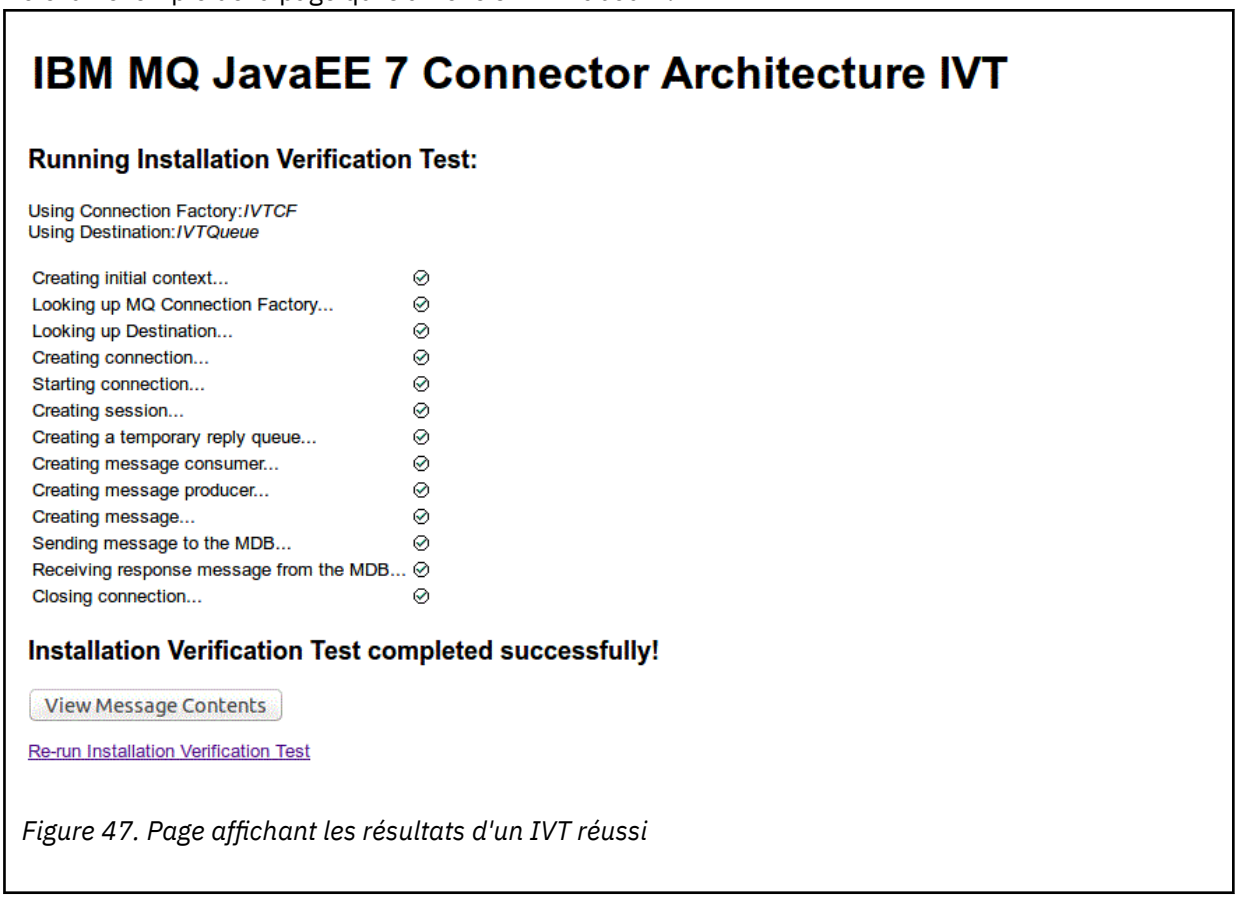


Figure 47. Page affichant les résultats d'un IVT réussi

Voici un exemple de la page qui s'affiche en cas d'échec de l'IVT. Pour obtenir plus d'informations sur la cause de l'échec, cliquez sur **Afficher la trace de pile**.

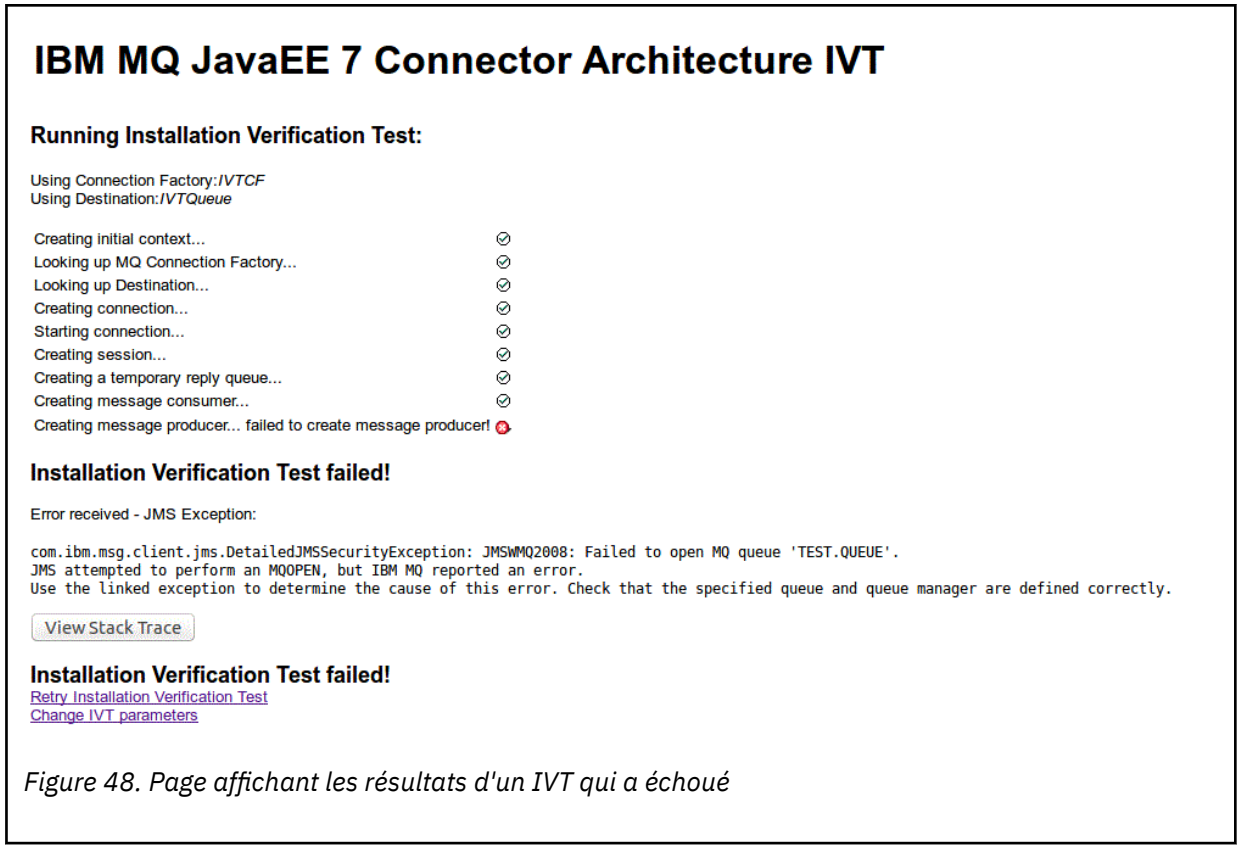


Figure 48. Page affichant les résultats d'un IVT qui a échoué

Windows Installation et test de l'adaptateur de ressources dans GlassFish Server

Pour installer l'adaptateur de ressources IBM MQ dans GlassFish Server sur un système d'exploitation Windows , vous devez d'abord créer et démarrer un domaine. Vous pouvez ensuite déployer et configurer l'adaptateur de ressources, puis déployer et exécuter l'application de test de vérification de l'installation (IVT).

Avant de commencer

- Ces instructions concernent GlassFish Server version 4.
- Cette version de GlassFish Server ne prend pas en charge Jakarta EE.

Pourquoi et quand exécuter cette tâche

Cette tâche suppose que vous disposez d'un serveur d'applications GlassFish Server en cours d'exécution et que vous êtes familiarisé avec les tâches d'administration standard pour ce serveur. Cette tâche suppose également que vous disposez d'une installation IBM MQ sur votre système local et que vous maîtrisez les tâches d'administration standard.

Remarque : Pour effectuer les étapes de tâche suivantes, vous devez disposer d'une installation IBM MQ en cours de fonctionnement, avec les objets suivants configurés:

- Un gestionnaire de files d'attente appelé QM, démarré sur le port 1414, qui utilise le canal SYSTEM.DEF.SVRCONN, et qui se connecte à l'aide du transport Client.
- Une file d'attente appelée Q1.

Procédure

1. Démarrez le programme shell GlassFish Server **asadmin** .

a) Ouvrez la ligne de commande Windows et accédez au répertoire *GlassFish*/bin , où *GlassFish* est le répertoire dans lequel GlassFish Server version 4 est installé.

b) Entrez la commande **asadmin** sur la ligne de commande.

La commande **asadmin** ouvre un programme shell dans la ligne de commande qui vous permet de créer un nouveau domaine.

GlassFish Server version 4 est démarré sur votre système.

2. Créez, puis démarrez un domaine.

a) Utilisez la commande **create-domain** , en spécifiant le port et le nom de domaine, pour créer un nouveau domaine. Entrez la commande suivante sur la ligne de commande :

```
create-domain --adminport port domain_name
```

où *port* est le numéro de port et *nom_domaine* est le nom que vous souhaitez que le domaine utilise.

Remarque : La commande **create-domain** est associée à de nombreux paramètres facultatifs. Toutefois, pour cette tâche, vous n'avez besoin que du paramètre `-- adminport` . Pour plus d'informations, voir la documentation du produit GlassFish Server version 4.

Si le port indiqué est en cours d'utilisation, le message suivant s'affiche:

`Le port pour nom_domaine port est utilisé`

Si le nom de domaine que vous avez spécifié est en cours d'utilisation, vous recevez un message vous indiquant que le nom que vous avez spécifié est déjà utilisé, ainsi qu'une liste de tous les noms de domaine actuellement indisponibles.

b) Lorsque vous êtes invité à entrer un nom d'utilisateur et un mot de passe, entrez les données d'identification à utiliser pour vous connecter au serveur d'applications via un navigateur Web.

Si la commande aboutit, un message résumant la création du domaine s'affiche sur la ligne de commande, y compris le message `Command create-domain executed successfully.`

Vous avez créé un domaine.

c) Démarrez votre domaine en entrant la commande suivante dans la ligne de commande:

```
start-domain domain_name
```

où *nom_domaine* est le nom de domaine que vous avez précédemment spécifié.

3. Utilisez un navigateur Web pour accéder au serveur d'applications GlassFish .

a) Dans la barre d'adresse d'un navigateur Web, entrez la commande suivante:

```
localhost:port
```

où *port* est le port que vous avez spécifié précédemment lors de la création de votre domaine.

La console GlassFish s'affiche.

b) Lorsque la console GlassFish a été chargée et que vous êtes invité à entrer un nom d'utilisateur et un mot de passe, entrez les données d'identification que vous avez spécifiées à l'étape 2b.

4. Téléchargez l'adaptateur de ressources sur GlassFish Server 4.

a) Dans la barre d'outils **Tâches communes** , sélectionnez l'élément de menu **Applications** pour afficher la page **Applications** .

b) Cliquez sur le bouton **Déployer** pour ouvrir la page **Déployer des applications ou des modules** .

c) Cliquez sur le bouton **Parcourir** , puis accédez à l'emplacement du fichier `wmq.jmsra.rar` . Sélectionnez le fichier, puis cliquez sur **OK**.

5. Créez un pool de connexions.

a) Dans la barre d'outils, sous **Ressources**, sélectionnez l'élément de menu **Connecteurs** .

b) Sélectionnez ensuite l'option de menu **Pools de connexions de connecteur** pour ouvrir la page **Pools de connexions de connecteur** .

c) Cliquez sur **Nouveau** pour ouvrir la page **Nouveau pool de connexions de connecteur (étape 1 sur 2)** .

d) Dans la page **Nouveau pool de connexions de connecteur (étape 1 sur 2)** , entrez le nom du pool sous la forme jms / ivt/IVTCF-Connection-Pool dans la zone **Nom du pool** .

e) Dans la zone **Adaptateur de ressources** , sélectionnez `wmq.jmsra`.

f) Dans la zone **Définition de connexion** , entrez `javax.jms.ConnectionFactory`.

g) Sélectionnez **Suivant**, puis cliquez sur **Terminer**.

6. Créez les ressources de connecteur.

a) Dans la barre d'outils, sous le menu **Connecteurs** , sélectionnez l'option **Ressource de connecteur** pour ouvrir la page **Ressources de connecteur** .

b) Sélectionnez **Nouveau**pour ouvrir la page **Nouvelle ressource de connecteur** .

c) Dans la zone **Nom JNDI** , entrez `IVTCF`.

d) Dans la zone **Nom du pool** , entrez `jms/ivt/IVTCF-Connection-Pool`.

e) Laissez toutes les autres zones vides.

f) Pour chacune des paires propriété / valeur suivantes, cliquez sur **Ajouter une propriété**et entrez le nom et la valeur de la propriété, comme illustré dans l'exemple suivant:

- nom: hôte ; valeur: localhost
- nom: port ; valeur 1414
- name: channel ; valeur: SYSTEM.DEF.SVRCONN
- nom: queueManager; valeur: QM
- nom: transportType; valeur: CLIENT

Remarque : Veillez à utiliser les valeurs correctes pour vos propres paramètres de configuration, qui peuvent être différents de ceux indiqués dans cet exemple.

g) Dans la barre d'outils, sous **Connecteurs**, sélectionnez l'option de menu **Ressources d'objet d'administration** pour ouvrir la page **Ressources d'objet d'administration** .

h) Dans la page **Ressources d'objet d'administration** , cliquez sur **Nouveau** pour ouvrir la page **Nouvelle ressource d'objet d'administration** .

i) Dans la zone **Nom JNDI** , entrez `IVTQueue`.

j) Dans la zone **Adaptateur de ressources** , entrez `wmq.jmsra`.

k) Dans la zone **Type de ressource** , entrez `javax.jms.Queue`.

l) Laissez la zone **Nom de classe** telle qu'elle est.

m) Pour chacune des paires propriété / valeur suivantes, cliquez sur **Ajouter une propriété**et entrez le nom et la valeur de la propriété, comme illustré dans l'exemple suivant:

- nom: nom ; valeur: IVTQueue
- Nom: baseQueueManagerName; value QM
- name: baseQueueNom ; valeur: Q1

Remarque : Veillez à utiliser les valeurs correctes pour vos propres paramètres de configuration, qui peuvent être différents de ceux indiqués dans cet exemple.

n) Cliquez sur **OK**.

o) Cochez la case **Activé** , puis cliquez sur **Activer**.

7. Déployez le fichier EAR `wmq.jmsra.ivt.ear` dans le serveur GlassFish .

a) Cliquez sur l'option **Applications** dans la barre d'outils pour afficher la page **Applications** .

b) Cliquez sur **Déployer** pour ajouter l'application IVT.

c) Dans la zone **Emplacement** , accédez au fichier `wmq.jmsra.ivt.ear`et sélectionnez-le.

d) Dans la zone **Virtual Servers** , sélectionnez **server**, puis cliquez sur **OK**.

8. Lancez le programme IVT.

a) Cliquez sur l'option **Applications** dans la barre d'outils pour afficher la page **Applications** .

b) Cliquez sur `wmq.jmsra.ivt` dans la table Applications déployées.

c) Cliquez sur le bouton **Lancer** dans le tableau Modules et composants.

d) Sélectionnez le lien http: link.

e) Cliquez sur **Exécuter IVT**.

Vous avez lancé le programme IVT et si vous réussissez, la sortie suivante s'affiche:

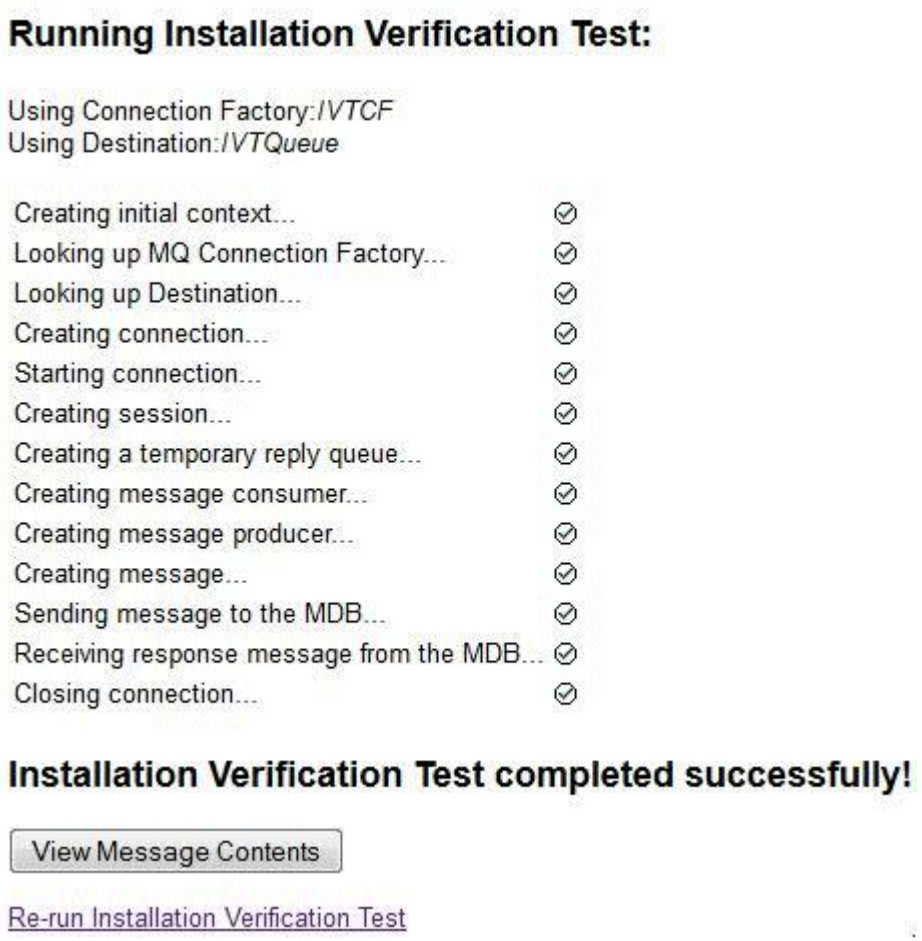


Figure 49. Sortie IVT réussie

Installation et test de l'adaptateur de ressources dans WildFly

Si vous installez l'adaptateur de ressources IBM MQ dans WildFly V10, vous devez d'abord apporter des modifications au fichier de configuration afin d'ajouter une définition de sous-système pour l'adaptateur de ressources IBM MQ . Vous pouvez ensuite déployer l'adaptateur de ressources et le tester en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Avant de commencer

- Ces instructions s'appliquent à WildFly V10.
- Cette version de WildFly ne prend pas en charge Jakarta EE.

Pourquoi et quand exécuter cette tâche

Cette tâche suppose que vous disposez d'un serveur d'applications WildFly en cours d'exécution et que vous êtes familiarisé avec les tâches d'administration standard de ce serveur. Cette tâche suppose également que vous disposez d'une installation IBM MQ et que vous êtes familiarisé avec les tâches d'administration standard.

Procédure

1. Créez un gestionnaire de files d'attente IBM MQ appelé ExampleQMet définissez-le comme décrit dans «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.

   Lors de la configuration du gestionnaire de files d'attente, notez les points suivants:

   - Le programme d'écoute doit être démarré sur le port 1414.
   - Le canal à utiliser est appelé SYSTEM.DEF.SVRCONN.
   - La file d'attente utilisée par l'application IVT est nommée TEST.QUEUE.

   File d'attente modèle SYSTEM.DEFAULT.MODEL.QUEUE doit également disposer des droits DSP et PUT pour que cette application puisse créer une file d'attente de réponses temporaire.
2. Editez le fichier de configuration *WildFly_Home*/standalone/configuration/standalone-full.xml et ajoutez le sous-système suivant:

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:4.0">
    <resource-adapters>
        <resource-adapter id="wmq.jmsra">
            <archive>
                wmq.jmsra.rar
            </archive>
            <transaction-support>NoTransaction</transaction-support>
            <connection-definitions>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/IVTCF" enabled="true"
use-java-context="true"
                                       pool-name="IVTCF">
                    <config-property name="channel">SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property
                                      name="hostName">localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
                <connection-definition class-
name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
                                       jndi-name="java:jboss/jms/ivt/JMS2CF" enabled="true"
use-java-context="true"
                                       pool-name="JMS2CF">
                    <config-property name="channel">
                        SYSTEM.DEF.SVRCONN
                    </config-property>
                    <config-property name="hostName">
                        localhost
                    </config-property>
                    <config-property name="transportType">
                        CLIENT
                    </config-property>
                    <config-property name="queueManager">
                        ExampleQM
                    </config-property>
                    <config-property name="port">
                        1414
                    </config-property>
                </connection-definition>
```

```
            </connection-definitions>
        <admin-objects>
            <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                          jndi-name="java:jboss/jms/ivt/IVTQueue" pool-name="IVTQueue">
                <config-property name="baseQueueName">
                    TEST.QUEUE
                </config-property>
                </admin-object>
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

3. Déployez l'adaptateur de ressources sur votre serveur en copiant le fichier `wmq.jmsra.rar` dans le répertoire *WildFly_Home*`/standalone/deployments`.
4. Déployez l'application IVT en faisant la copie du fichier `wmq.jmsra.ivt.ear` dans le répertoire *WildFly_Home*`/standalone/deployments`.
5. Démarrez le serveur d'applications en affichant une invite de commande, en accédant au répertoire *WildFly_Home*`/bin` et en exécutant la commande suivante:

   ```
   standalone.bat -c standalone-full.xml
   ```

6. Exécutez l'application IVT.

   Pour plus d'informations, voir «Vérification de l'installation de l'adaptateur de ressources», à la page 503. Pour WildFly, l'URL par défaut est `http://localhost:8080/WMQ_IVT/`.

Utilisation conjointe de IBM MQ et de WebSphere Application Server

Grâce au fournisseur de messagerie IBM MQ de WebSphere Application Server, les applications de messagerie Java Message Service (JMS) peuvent utiliser votre système IBM MQ comme fournisseur externe de ressources de messagerie JMS.

Pourquoi et quand exécuter cette tâche

Les applications écrites dans Java et exécutées sous WebSphere Application Server peuvent utiliser la spécification Java Message Service (JMS) pour la messagerie. La messagerie dans cet environnement peut être fournie par un gestionnaire de files d'attente IBM MQ .

Un avantage de l'utilisation d'un gestionnaire de files d'attente IBM MQ est que la connexion d'applications JMS peut participer pleinement aux fonctionnalités d'un réseau IBM MQ , ce qui permet aux applications d'échanger des messages avec des gestionnaires de files d'attente qui s'exécutent sur une multitude de plateformes.

Les applications peuvent utiliser le *transport client* ou le *transport de liaisons* pour l'objet de fabrique de connexions de file d'attente. Pour le transport de liaisons, le gestionnaire de files d'attente doit exister localement dans l'application qui requiert une connexion.

Par défaut, les messages JMS qui se trouvent dans les files d'attente IBM MQ utilisent un en-tête MQRFH2 pour contenir certaines informations d'en-tête de message JMS . La plupart des applications IBM MQ existantes ne peuvent pas traiter les messages avec ces en-tête et requièrent leurs propres en-têtes, par exemple MQCIH pour CICS Bridge, ou MQWIH pour les applications IBM MQ Workflow. Pour plus d'informations sur ces considérations spéciales, voir Mappage des messages JMS vers les messages IBM MQ.

Tâches associées

Configuration des ressources JMS dans WebSphere Application Server

Configuration du serveur d'applications pour utiliser le dernier niveau de maintenance de l'adaptateur de ressources

Utilisation d'WebSphere Application Server avec IBM MQ

IBM MQ et IBM MQ for z/OS peuvent être utilisés avec le, ou en remplacement du, fournisseur de messagerie par défaut qui est inclus avec WebSphere Application Server.

Le fournisseur de messagerie IBM MQ est installé comme faisant partie de WebSphere Application Server. Cela inclut une version de l'adaptateur de ressources IBM MQ et la fonctionnalité du client transactionnel étendu IBM MQ, ce qui permet au gestionnaire de files d'attente de participer aux transactions XA gérées par le serveur d'applications. A l'aide de l'adaptateur de ressources, les beans gérés par message peuvent être configurés pour utiliser les spécifications d'activation ou les ports d'écoute.

Pour que le serveur d'applications soit pris en charge, le programme de test de vérification de l'installation de l'adaptateur de ressourcesIBM MQ doit être déployé sur le serveur d'applications et s'exécuter correctement. Une fois que le programme de test de vérification de l'installation de l'adaptateur de ressources IBM MQ a été exécuté avec succès, l'adaptateur de ressources IBM MQ peut se connecter à n'importe quel gestionnaire de files d'attente IBM MQ pris en charge.

Connexions JMS de WebSphere Application Server à IBM MQ

Avant de prendre en compte les niveaux de IBM MQ pouvant être utilisés avec WebSphere Application Server, il est important de comprendre comment les applications Java Message Service (JMS) s'exécutant dans le serveur d'applications peuvent se connecter aux gestionnaires de files d'attente IBM MQ .

Les applications JMS qui doivent accéder aux ressources d'un gestionnaire de files d'attente IBM MQ peuvent le faire à l'aide de l'un des types de transport suivants:

LIAISONS
Ce transport peut être utilisé lorsque le serveur d'applications et le gestionnaire de files d'attente sont installés sur la même machine et la même image de système d'exploitation. En mode LIAISONS, toutes les communications entre les deux produits s'effectuent par le biais d'IPC (Inter-Process Communication).

Le fournisseur de messagerie IBM MQ n'inclut pas les bibliothèques natives requises pour la connexion à un gestionnaire de files d'attente IBM MQ en mode BINDINGS. Pour utiliser une connexion en mode LIAISONS, IBM MQ doit être installé sur la même machine que le serveur d'applications, et le chemin de la bibliothèque native de l'adaptateur de ressources doit être configuré pour pointer vers le répertoire IBM MQ dans lequel sont installées ces bibliothèques. Pour plus d'informations, voir la documentation du produit WebSphere Application Server :

- Pour WebSphere Application Server traditional, voir Configuration du fournisseur de messagerie IBM MQ avec des bibliothèques natives.
- Pour WebSphere Liberty, voir Déploiement d'applications JMS dans Liberty pour utiliser le fournisseur de messagerie IBM MQ.

z/OS Sous z/OS, si vous souhaitez connecter une fabrique de connexions WebSphere Application Server à un gestionnaire de files d'attente IBM MQ en mode liaisons, vous devez spécifier les bibliothèques IBM MQ appropriées dans la concaténation STEPLIB WebSphere Application Server . Pour plus d'informations, voir les bibliothèques IBM MQ et WebSphere Application Server for z/OS STEPLIB dans la documentation du produit WebSphere Application Server .

CLIENT
Le transport client utilise TCP/IP pour communiquer entre WebSphere Application Server et IBM MQ. Le mode CLIENT peut être utilisé lorsque le serveur d'applications et le gestionnaire de files d'attente sont installés sur des machines différentes, mais aussi lorsque les deux produits sont installés sur la même machine et la même image de système d'exploitation.

Les applications JMS peuvent également spécifier le type de transport BINDINGS_THEN_CLIENT. Lorsque ce dernier est utilisé, l'application tente initialement de se connecter au gestionnaire de files d'attente à l'aide du mode LIAISONS. En cas d'échec, elle tente d'utiliser le transport CLIENT.

Procédure de recherche de la version de l'adaptateur de ressources IBM MQ installée dans WebSphere Application Server

Pour plus d'informations sur la version de l'adaptateur de ressources IBM MQ qui est installée dans WebSphere Application Server, voir la note technique Quelle version de l'adaptateur de ressources WebSphere MQ (RA) est fournie avec WebSphere Application Server?.

Vous pouvez utiliser les commandes Jython et JACL suivantes pour déterminer le niveau de l'adaptateur de ressources actuellement utilisé par WebSphere Application Server :

Jython

```
wmqInfoMBeansUnsplit = AdminControl.queryNames("WebSphere:type=WMQInfo,*")
wmqInfoMBeansSplit = AdminUtilities.convertToList(wmqInfoMBeansUnsplit)
for wmqInfoMBean in wmqInfoMBeansSplit: print wmqInfoMBean; print
AdminControl.invoke(wmqInfoMBean, 'getInfo', '')
```

Remarque : Vous devez cliquer à deux reprises sur **Retour** après avoir saisi cette commande afin de l'exécuter.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Mise à jour de l'adaptateur de ressources

Les mises à jour apportées à l'adaptateur de ressources IBM MQ qui est installé avec le serveur d'applications sont incluses dans les groupes de correctifs WebSphere Application Server. Mise à jour de l'adaptateur de ressources IBM MQ à l'aide de l'option **Mettre à jour l'adaptateur de ressources ...** dans la console d'administration WebSphere Application Server n'est pas recommandée, car cela signifie que les mises à jour fournies dans les groupes de correctifs WebSphere Application Server n'auront aucun effet.

variable MQ_INSTALL_ROOT

Depuis la WebSphere Application Server 7.0, MQ_INSTALL_ROOT est utilisé uniquement pour localiser les bibliothèques natives et est remplacé par n'importe quel chemin de bibliothèque native configuré sur l'adaptateur de ressources.

Connexion de WebSphere Application Server à IBM MQ

Avertissement :

1. Toute version prise en charge de WebSphere Application Server peut utiliser l'adaptateur de ressources IBM MQ qui lui est fourni pour se connecter à toute version prise en charge de IBM MQ.
2. Si le mode liaisons est utilisé, certaines bibliothèques de WebSphere Application Server doivent correspondre à la version du gestionnaire de files d'attente auquel elles se connectent:
   - WebSphere Application Server doit être configuré pour charger les bibliothèques natives fournies avec IBM MQ 9.4. Pour plus d'informations, voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 100.
   - z/OS Sous z/OS, vous devez spécifier les bibliothèques IBM MQ correctes dans la concaténation STEPLIB WebSphere Application Server .

     Voir les bibliothèques IBM MQ et WebSphere Application Server for z/OS STEPLIB pour plus de détails sur les bibliothèques IBM MQ dont vous avez besoin.

     Si vous disposez de bibliothèques pour une version de IBM MQ dans LINKLIST (LINKLST), vous pouvez vous connecter à une autre version de IBM MQ en remplaçant les bibliothèques par STEPLIB.
3. La version de l'adaptateur de ressources IBM MQ est indépendante des versions de bibliothèque native (partagée) fournies par l'installation du gestionnaire de files d'attente.

Par exemple, WebSphere Application Server 8.5, avec un adaptateur de ressources IBM MQ 8.0 , peut toujours gérer une connexion de liaisons à un gestionnaire de files d'attente IBM MQ 9.0 à l'aide des bibliothèques natives IBM MQ 9.0 .

Pour plus d'informations, voir «Déclaration de prise en charge de l'adaptateur de ressources IBM MQ», à la page 447.

Les types de transport BINDINGS et CLIENT peuvent être utilisés pour se connecter à IBM MQ à partir de n'importe quelle version de WebSphere Application Server. Pour le type de transport BINDINGS, les restrictions suivantes s'appliquent:

- IBM MQ doit être installé sur la même machine que le serveur d'applications.
- WebSphere Application Server doit être configuré pour charger les bibliothèques natives fournies avec IBM MQ.
- z/OS Sous z/OS, si vous souhaitez connecter une fabrique de connexions WebSphere Application Server à un gestionnaire de files d'attente IBM MQ en mode liaisons, les bibliothèques IBM MQ appropriées doivent être spécifiées dans la concaténation STEPLIB WebSphere Application Server .

Le tableau suivant présente les versions de WebSphere Application Server dans lesquelles chaque version de l'adaptateur de ressources IBM MQ est prise en charge pour s'exécuter.

Tableau 70. Mappage des versions WebSphere Application Server aux versions de l'adaptateur de ressources IBM MQ .

Version de l'adaptateur de ressources IBM MQ	**Dans quelle version de WebSphere Application Server cette version de l'adaptateur de ressources peut-elle s'exécuter?**
IBM MQ 9.0 et versions ultérieures	L'adaptateur de ressources peut s'exécuter dans: • Toute version compatible Java EE 7 de WebSphere Liberty. • WebSphere Application Server traditional 9.0
IBM MQ 8.0	L'adaptateur de ressources peut s'exécuter dans n'importe quelle version compatible Java EE 7 de WebSphere Liberty L'exécution de l'adaptateur de ressources IBM MQ 8.0 n'est pas prise en charge dans WebSphere Application Server traditional. L'adaptateur de ressources déjà installé dans WebSphere Application Server traditional doit être utilisé pour la connexion aux gestionnaires de files d'attente IBM MQ 8.0 .

Concepts associés

«Déclaration de prise en charge de l'adaptateur de ressources IBM MQ», à la page 447
L'adaptateur de ressources IBM MQ que vous devez utiliser pour la communication entre une application et un gestionnaire de files d'attente varie selon que vous utilisez l'API Jakarta Messaging 3.0 ou l'API JMS 2.0 .

Information associée

Configuration système requise pour IBM MQ

Détermination du nombre de connexions TCP/IP créées entre WebSphere Application Server et IBM MQ

A l'aide de la fonction de partage de conversations, plusieurs conversations peuvent partager des instances de canal MQI, également appelée connexion TCP/IP.

Pourquoi et quand exécuter cette tâche

Les applications s'exécutant dans WebSphere Application Server 7 et WebSphere Application Server 8, qui utilisent le mode normal du fournisseur de messagerie IBM MQ , utiliseront automatiquement cette fonction. Cela signifie que plusieurs applications s'exécutant dans la même instance de serveurs d'applications et qui se connectent au même gestionnaire de files d'attente IBM MQ peuvent partager la même instance de canal.

Le nombre de conversations pouvant être partagées sur une seule instance de canal est déterminé par la propriété de canal IBM MQ **SHARECNV**. La valeur par défaut de cette propriété pour les canaux de connexion serveur est 10.

En examinant le nombre de conversations créées par WebSphere Application Server 7 et WebSphere Application Server 8, vous pouvez déterminer le nombre d'instances de canal créées.

Pour plus d'informations sur le mode du fournisseur de messagerie IBM MQ, voir PROVIDERVERSION mode normal.

Concepts associés

Utilisation du partage de conversations
Dans un environnement dans lequel le partage de conversations est autorisé, les conversations peuvent partager une instance de canal MQI.

«Partage d'une connexion TCP/IP dans IBM MQ classes for JMS», à la page 324
Plusieurs instances d'un canal MQI peuvent être créées pour partager une connexion TCP/IP unique.

Fabriques de connexions JMS

Les applications exécutées dans WebSphere Application Server, qui utilisent une fabrique de connexions de fournisseur de messagerie IBM MQ pour créer des connexions et des sessions, ont des conversations actives pour chaque connexion JMS créée à partir de la fabrique de connexions et pour chaque session JMS créée à partir d'une connexion JMS .

Une conversation pour chaque connexion JMS créée à partir de la fabrique de connexions.

Chaque fabrique de connexions JMS dispose d'un pool de connexions associé qui est divisé en deux sections, le pool libre et le pool actif. Les deux pools sont initialement vides.

Lorsqu'une application crée une connexion JMS à partir d'une fabrique de connexions, WebSphere Application Server vérifie s'il existe une connexion JMS dans le pool libre. Si c'est le cas, cette dernière est déplacée dans le pool actif et fournie à l'application. Si ce n'est pas le cas, une nouvelle connexion JMS est créée, placée dans le pool actif et fournie à l'application. Le nombre maximal de connexions pouvant être créées à partir d'une fabrique de connexions est spécifié par la propriété de pool de connexions de fabrique de connexions **Maximum connections**. La valeur par défaut de cette propriété est 10.

Lorsqu'une application a fini d'utiliser une connexion JMS et l'a fermée, la connexion est déplacée du pool actif vers le pool libre, où elle pourra être réutilisée. La propriété de pool de connexions **Unused timeout** définit la durée pendant laquelle une connexion JMS peut rester dans le pool libre avant d'être déconnectée. La valeur par défaut de cette propriété est de 1800 secondes (30 minutes).

Lorsqu'une connexion JMS est créée pour la première fois, une conversation entre WebSphere Application Server et IBM MQ démarre. La conversation reste active jusqu'à la fermeture de la connexion lorsque la valeur de la propriété **Unused timeout** pour le pool libre est dépassée.

Une conversation pour chaque session JMS créée à partir d'une connexion JMS.

Chaque connexion JMS créée à partir d'une fabrique de connexions du fournisseur de messagerie IBM MQ dispose d'un pool de sessions JMS associé. Les pools de sessions fonctionnent de la même manière que les pools de connexions. Le nombre maximal de sessions JMS pouvant être créées à partir d'une seule connexion JMS est déterminé par la propriété de pool de sessions de fabrique de connexions **Maximum connections**. La valeur par défaut de cette propriété est 10.

Une conversation démarre lorsqu'une session JMS est créée pour la première fois. La conversation reste active jusqu'à la fermeture de la session JMS car elle est restée dans le pool libre pendant plus longtemps que la valeur de la propriété **Unused timeout** pour le pool de sessions.

Calcul d'une valeur pour la propriété SHARECNV

Vous pouvez calculer le nombre maximal de conversations à partir d'une seule fabrique de connexions sur IBM MQ à l'aide de la formule suivante :

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Pour connaître le nombre d'instances de canal qui seront créées pour permettre ce nombre de conversations, faites le calcul suivant :

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Le reste de ce calcul peut être arrondi.

Pour une fabrique de connexions simple qui utilise la valeur par défaut pour les propriétés du pool de connexions **Maximum connections** et du pool de sessions **Maximum connections** , le nombre maximal de conversations pouvant exister entre WebSphere Application Server et IBM MQ pour cette fabrique de connexions est le suivant:

```
Maximum number of conversations =
   connection Pool Maximum Connections +
  (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Exemple :

```
        = 10 + (10 * 10)
        = 10 + 100
        = 110
```

Si cette fabrique de connexions se connecte à IBM MQ à l'aide d'un canal dont la propriété **SHARECNV** est définie sur 10, le nombre maximal d'instances de canal qui seront créées pour cette fabrique de connexions est le suivant :

```
Maximum number of channel instances = Maximum number of conversations / SHARECNV for the
channel being used
```

Exemple :

```
        = 110 / 10
        = 11 (rounded up to nearest connection)
```

Spécifications d'activation

Les applications de bean géré par message configurées pour utiliser une spécification d'activation comportent des conversations actives qui permettent à la spécification d'activation de surveiller une

destination JMS et à chaque session de serveur utilisée d'exécuter une instance de bean géré par message pour traiter des messages.

Les conversations suivantes sont actives pour les applications de bean géré par message qui sont configurées pour utiliser une spécification d'activation :

- Une conversation de la spécification d'activation pour la surveillance d'une destination JMS pour les messages appropriés. Cette conversation démarre dès le démarrage de la spécification d'activation et reste active jusqu'à l'arrêt de la spécification d'activation.
- Une conversation pour chaque session de serveur utilisée pour exécuter une instance de bean géré par message afin de traiter des messages.

La propriété avancée de spécification d'activation **Maximum server sessions** indique le nombre maximal de sessions de serveur pouvant être actives simultanément pour une spécification d'activation donnée. La valeur par défaut de cette propriété est 10. Des sessions de serveur sont créées si nécessaire et sont fermées si elles sont restées en veille pendant une durée spécifiée par la propriété avancée **Server session pool timeout** de la spécification d'activation. La valeur par défaut de cette propriété est de 300000 millisecondes (5 minutes).

Les conversations démarrent lorsqu'une session de serveur est créée, et sont arrêtées lorsque la spécification d'activation est arrêtée ou lorsqu'une session serveur arrive à expiration.

Cela signifie que le nombre maximal de conversations d'une seule spécification d'activation vers IBM MQ peut être calculé à l'aide de la formule suivante :

```
Maximum number of conversations = Maximum server sessions + 1
```

Pour connaître le nombre d'instances de canal qui seront créées pour permettre ce nombre de conversations, faites le calcul suivant :

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Le reste de ce calcul peut être arrondi.

Pour une spécification d'activation simple qui utilise la valeur par défaut de la propriété **Maximum server sessions** , le nombre maximal de conversations pouvant exister entre WebSphere Application Server et IBM MQ pour cette spécification d'activation est calculé comme suit:

```
Maximum number of conversations = Maximum server sessions + 1
```

Exemple :

```
    = 10 + 1
    = 11
```

Si cette spécification d'activation se connecte à IBM MQ à l'aide d'un canal dont la propriété **SHARECNV** est définie sur 10, le nombre d'instances de canal créées est alors calculé comme suit :

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Exemple :

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Ports d'écoute s'exécutant en mode ASF (Application Server Facilities)

Les ports d'écoute s'exécutant en mode ASF utilisés par des applications de bean géré par message créent des conversations pour chaque session de serveur. L'une d'elles surveille une destination pour les messages appropriés et une autre exécute une instance de bean géré par message pour traiter les messages. Le nombre de conversations pour chaque port d'écoute peut être calculé à partir d'un nombre maximal de sessions.

Par défaut, les ports d'écoute s'exécutent en mode ASF dans le cadre de la spécification JMS 1.1 qui définit le mécanisme devant être utilisé par les serveurs d'applications pour détecter des messages et les distribuer aux beans gérés par message pour traitement. Les applications de bean géré par message configurés pour utiliser des ports d'écoute dans ce mode d'opération par défaut créent des conversations :

Une conversation du port d'écoute pour la surveillance d'une destination pour les messages appropriés.

Les ports d'écoute sont configurés pour utiliser une fabrique de connexions JMS. Lorsqu'un port d'écoute démarre, une demande de connexion JMS à partir du pool libre de fabriques de connexions est effectuée. La connexion est renvoyée au pool libre lorsque le port d'écoute est arrêté. Pour plus d'informations sur l'utilisation du pool de connexions et l'impact sur le nombre de conversations vers IBM MQ, voir «Fabriques de connexions JMS», à la page 515.

Une conversation pour chaque session de serveur utilisée pour exécuter une instance de bean géré par message afin de traiter des messages.

La propriété du port d'écoute **Maximum sessions** indique le nombre maximal de sessions de serveur pouvant être actives simultanément pour un port d'écoute donné. La valeur par défaut de cette propriété est 10. Des sessions serveur sont créées si nécessaire et utilisent des sessions JMS provenant du pool de sessions associé à la connexion JMS utilisée par le port d'écoute.

Si une session serveur est restée en veille pendant une période spécifiée par la propriété personnalisée **SERVER.SESSION.POOL.UNUSED.TIMEOUT** du service d'écoute de messages, la session est fermée et la session JMS utilisée est renvoyée au pool libre de sessions. La session JMS reste dans le pool libre de sessions jusqu'à ce qu'elle soit utilisée ou est fermée car elle est restée en veille dans le pool libre plus longtemps que la valeur spécifiée de la propriété **Unused timeout** du pool de sessions.

Pour plus d'informations sur l'utilisation du pool de sessions et la gestion des conversations entre WebSphere Application Server et IBM MQ, voir «Fabriques de connexions JMS», à la page 515.

Pour plus d'informations sur la propriété personnalisée **SERVER.SESSION.POOL.UNUSED.TIMEOUT** du service d'écoute de messages, voir Monitoring server session pools for listener ports dans la documentation du produit WebSphere Application Server.

Calcul du nombre maximal de conversations à partir d'un seul port d'écoute sur IBM MQ

Vous pouvez calculer le nombre maximal de conversations à partir d'un seul port d'écoute sur IBM MQ à l'aide de la formule suivante :

```
Maximum number of conversations = Maximum sessions + 1
```

Pour connaître le nombre d'instances de canal qui seront créées pour permettre ce nombre de conversations, faites le calcul suivant :

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Le reste de ce calcul peut être arrondi.

Pour un port d'écoute simple qui utilise la valeur par défaut de la propriété **Maximum sessions** , le nombre maximal de conversations pouvant exister entre WebSphere Application Server et IBM MQ pour ce port d'écoute est calculé comme suit:

```
Maximum number of conversations = Maximum sessions + 1
```

Exemple :

```
    = 10 + 1
    = 11
```

Si ce port d'écoute se connecte à IBM MQ à l'aide d'un canal dont la propriété **SHARECNV** est définie sur 10, le nombre d'instances de canal créées est alors calculé comme suit :

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Exemple :

```
    = 11 / 10
    = 2 (rounded up to nearest connection)
```

Ports d'écoute s'exécutant en mode non ASF (Application Server Facilities)

Les ports d'écoute s'exécutant en mode non ASF peuvent être configurés pour surveiller la destination de file d'attente et la destination de rubrique en utilisant des sessions de serveur. Les sessions de serveur peuvent avoir plusieurs conversations dont le nombre maximal peut être calculé au cas par cas.

Les ports d'écoute peuvent être configurés pour s'exécuter en mode non ASF, ce qui modifie la manière dont ils surveillent les destinations JMS. Les applications de bean géré par message utilisant des ports d'écoute en mode non ASF créent une conversation pour chaque session de serveur utilisée pour exécuter une instance de bean géré par message pour traiter des messages. La propriété du port d'écoute **maximum sessions** indique le nombre maximal de sessions de serveur pouvant être actives simultanément pour un port d'écoute donné. Sa valeur par défaut est 10.

Lors de l'exécution en mode non ASF, un port d'écoute surveillant la destination d'une file d'attente crée automatiquement le nombre de sessions de serveur spécifié par la propriété **Maximum sessions** du port d'écoute. Toutes ces sessions de serveur utilisent les sessions JMS provenant du pool de sessions associé à la connexion JMS utilisée par le port d'écoute et elles surveillent en continu une destination JMS pour les messages appropriés.

Si le port d'écoute est configuré pour surveiller une destination de rubrique, la valeur de la propriété **Maximum sessions** est ignorée et une seule session de serveur est utilisée.

Les sessions de serveur utilisées par un port d'écoute s'exécutant en mode non ASF restent actives jusqu'à ce que le port d'écoute soit arrêté. Les sessions JMS ayant été utilisées sont alors renvoyées au pool libre de sessions pour la connexion JMS utilisée par le port d'écoute.

Pour plus d'informations sur l'utilisation du pool de sessions et la gestion des conversations entre WebSphere Application Server et IBM MQ, voir «Fabriques de connexions JMS», à la page 515.

Pour plus d'informations sur le mode de fonctionnement ASF et non ASF avec WebSphere Application Serveret sur la configuration des ports d'écoute pour utiliser le mode non ASF, voir Traitement des messages en mode ASF et non ASF.

Calcul du nombre maximal de conversations au cours de la surveillance d'une destination de file d'attente

Le nombre maximal de conversations à partir d'un seul port d'écoute, qui s'exécute en mode non ASF et surveille une destination de file d'attente pour IBM MQ, peut être calculé à l'aide de la formule suivante :

```
Maximum number of conversations = Maximum sessions
```

Pour connaître le nombre d'instances de canal qui seront créées pour permettre ce nombre de conversations, faites le calcul suivant :

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Le reste de ce calcul peut être arrondi.

Pour un port d'écoute simple qui s'exécute en mode non ASF en utilisant la valeur par défaut de la propriété **Maximum sessions** et en surveillant une destination de file d'attente, le nombre maximal de conversations pouvant exister entre WebSphere Application Server et IBM MQ pour ce port d'écoute est le suivant :

```
Maximum number of conversations = Maximum sessions
```

Exemple :

```
     = 10
```

Si ce port d'écoute se connecte à IBM MQ à l'aide d'un canal dont la propriété **SHARECNV** est définie sur 10, le nombre d'instances de canal créées est alors calculé comme suit :

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Exemple :

```
     = 10 / 10
     = 1
```

Calcul du nombre maximal de conversations au cours de la surveillance d'une destination de rubrique

Pour un port d'écoute s'exécutant en mode non ASF et configuré pour surveiller une destination de rubrique, le nombre de conversations à partir du port d'écoute sur IBM MQ est le suivant :

```
 Maximum number of conversations = 1
```

Pour connaître le nombre d'instances de canal qui seront créées pour permettre ce nombre de conversations, faites le calcul suivant :

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Le reste de ce calcul peut être arrondi.

Pour un port d'écoute simple qui s'exécute en mode non ASF en utilisant la valeur par défaut de la propriété **Maximum sessions** et en surveillant une destination de rubrique, le nombre maximal de conversations pouvant exister entre WebSphere Application Server et IBM MQ pour ce port d'écoute est le suivant :

```
Maximum number of conversations = Maximum sessions
```

Exemple :

```
     = 10
```

Si ce port d'écoute se connecte à IBM MQ à l'aide d'un canal dont la propriété **SHARECNV** est définie sur 10, le nombre d'instances de canal créées est alors calculé comme suit :

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Exemple :

```
= 10 / 10
= 1
```

Configuration des alias d'authentification pour sécuriser la connexion de WebSphere Application Server à IBM MQ

Les alias d'authentification sont mappés vers une combinaison de nom d'utilisateur et de mot de passe qui peut être utilisée pour sécuriser une connexion WebSphere Application Server à IBM MQ. Vous pouvez configurer une fabrique de connexions avec un alias d'authentification.

Utilisation des alias d'authentification avec des applications d'entreprise

Lorsqu'une application d'entreprise s'exécutant dans WebSphere Application Server tente de créer une connexion JMS à IBM MQ, elle recherche une définition de fabrique de connexions du fournisseur de messagerie IBM MQ depuis le référentiel Java Naming Directory Interface (JNDI) du serveur d'applications.

Lorsque la définition de fabrique de connexions du fournisseur de messagerie IBM MQ est localisée dans le référentiel JNDI du serveur d'applications, l'une des méthodes suivantes est appelée :

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Si la fabrique de connexions est configurée avec un alias d'authentification J2C défini, le nom d'utilisateur et le mot de passe dans l'alias d'authentification peuvent être transmis à IBM MQ lorsque la fabrique de connexions est utilisée pour créer une connexion.

Fabriques de connexions et alias d'authentification

Les fabriques de connexions du fournisseur de messagerie IBM MQ contiennent des informations sur la connexion aux gestionnaires de files d'attente IBM MQ. Les applications d'entreprise exécutées dans WebSphere Application Server peuvent utiliser les fabriques de connexion pour créer des connexions JMS à IBM MQ.

WebSphere Application Server stocke les définitions de fabriques de connexions dans un référentiel accessible à l'aide de JNDI. Lorsqu'une fabrique de connexions est créée, un nom JNDI lui est affecté de manière à ce qu'elle soit identifiée de manière unique dans la portée du serveur d'applications (au niveau de la cellule ou du serveur) au niveau de laquelle est définie.

Par exemple, une fabrique de connexions du fournisseur de messagerie IBM MQ définie à la portée de niveau cellule WebSphere Application Server contient des informations sur le mode de connexion au gestionnaire de files d'attente (`myQM`) à l'aide du transport BINDINGS. Le nom JNDI `jms/myCF` est attribué à cette fabrique de connexions afin que cette dernière soit identifiée de manière unique.

Les fabriques de connexions peuvent également être configurées pour utiliser un alias d'authentification. Les alias d'authentification sont mappés vers une combinaison de nom d'utilisateur et de mot de passe. Selon le mode d'utilisation de la fabrique de connexions, le nom d'utilisateur et le mot de passe dans l'alias d'authentification peuvent ou non être transmis à IBM MQ lorsque la connexion JMS est créée.

Important : Avant IBM MQ 8.0, le gestionnaire des droits d'accès aux objets IBM MQ par défaut (OAM) a effectué une vérification d'autorisation pour s'assurer uniquement que le nom d'utilisateur transmis à IBM MQ lors de l'établissement d'une connexion est autorisé à accéder au gestionnaire de files d'attente.

Aucune vérification n'est effectuée pour valider le mot de passe indiqué. Pour effectuer une vérification d'authentification et confirmer que l'identificateur utilisateur et le mot de passe correspondent, vous devez écrire un exit de sécurité de canal IBM MQ. Pour plus de détails sur la procédure à suivre, voir «Programmes d'exit de sécurité de canal», à la page 996.

Dans IBM MQ 8.0, le gestionnaire de files d'attente vérifie le mot de passe en plus du nom d'utilisateur.

Utilisation de la fabrique de connexions

Les rubriques suivantes contiennent des informations sur l'utilisation de la fabrique de connexions à l'aide des recherches directes et indirectes :

- «Utilisation d'une fabrique de connexions à l'aide d'une recherche directe», à la page 525
- «Utilisation d'une fabrique de connexions à l'aide d'une recherche indirecte», à la page 526

Utilisation du transport CLIENT

Les fabriques de connexions qui sont configurées pour utiliser le transport CLIENT doivent spécifier quel canal de connexion serveur IBM MQ (SVRCONN) ils vont utiliser pour se connecter au gestionnaire de files d'attente.

Si l'identifiant de l'utilisateur d'agent de canal IBM MQ affecté à la propriété MCAUSER reste à blanc pour le canal devant être employé par la fabrique de connexions, cette dernière peut être utilisée avec une recherche directe ou indirecte.

Si la propriété MCAUSER est définie sur un identificateur d'utilisateur, ce dernier est transmis à IBM MQ lorsque la fabrique de connexions est utilisée pour créer une connexion à IBM MQ, que l'application d'entreprise utilise la recherche directe ou la recherche indirecte.

Tableaux récapitulatifs

Les tableaux suivants indiquent les identificateurs d'utilisateur qui sont transmis à IBM MQ lorsque le transport LIAISONS et CLIENT respectivement sont utilisés :

Tableau 71. Mode LIAISONS

Configuration	**L'application appelle `ConnectionFactory.createConnection()`**	**L'application appelle `ConnectionFactory.createConnection(String username, String password)`**
Le descripteur de déploiement de l'application ne contient pas de référence de ressource pour la fabrique de connexions	L'identificateur utilisateur pour le traitement du serveur d'applications est transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe qui ont été indiqués dans la méthode `ConnectionFactory.createConnection(String username, String password)` sont transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions et la propriété **`res-auth`** est définie sur "Application"	L'identificateur utilisateur pour le traitement du serveur d'applications est transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe qui ont été indiqués dans la méthode `ConnectionFactory.createConnection(String username, String password)` sont transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions et la propriété **`res-auth`** est définie sur "Container"	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification pour la fabrique de connexions sont transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification pour la fabrique de connexions sont transmis à IBM MQ.

Tableau 71. Mode LIAISONS (suite)

Configuration	**L'application appelle `ConnectionFactory.createConnection()`**	**L'application appelle `ConnectionFactory.createConnection(String username, String password)`**
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **res-auth** est définie sur "Container" et l'application est configurée avec un alias d'authentification.	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification devant être utilisés par l'application sont transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification devant être utilisés par l'application sont transmis à IBM MQ.

Tableau 72. Mode CLIENT

Configuration	**L'application appelle `ConnectionFactory.createConnection()`**	**L'application appelle `ConnectionFactory.createConnection(String username, String password)`**
Le descripteur de déploiement de l'application ne contient pas de référence de ressource pour la fabrique de connexions et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER n'est pas définie	L'identificateur utilisateur pour le traitement du serveur d'applications est transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe qui ont été indiqués dans la méthode `ConnectionFactory.createConnection(String username, String password)` sont transmis à IBM MQ.
Le descripteur de déploiement de l'application ne contient pas de référence de ressource pour la fabrique de connexions et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER est définie sur un identificateur utilisateur	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **res-auth** est définie sur *Application* et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER n'est pas définie	L'identificateur utilisateur pour le traitement du serveur d'applications est transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe qui ont été indiqués dans la méthode `ConnectionFactory.createConnection(String username, String password)` sont transmis à IBM MQ.

Tableau 72. Mode CLIENT (suite)		
Configuration	**L'application appelle `ConnectionFactory.createConnection()`**	**L'application appelle `ConnectionFactory.createConnection(String username, String password)`**
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **`res-auth`** est définie sur *Application* et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER est définie sur un identificateur utilisateur	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **`res-auth`** est définie sur "*Conteneur* et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER n'est pas définie	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification pour la fabrique de connexions sont transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification pour la fabrique de connexions sont transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **`res-auth`** est définie sur "*Conteneur* et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER est définie sur un identificateur utilisateur	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **`res-auth`** est définie sur "*Conteneur* et l'application a été configurée avec un alias d'authentification et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont la propriété MCAUSER n'est pas définie	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification devant être utilisés par l'application sont transmis à IBM MQ.	L'identificateur utilisateur et le mot de passe indiqués dans l'alias d'authentification devant être utilisés par l'application sont transmis à IBM MQ.

Tableau 72. Mode CLIENT (suite)		
Configuration	**L'application appelle `ConnectionFactory.createConnection()`**	**L'application appelle `ConnectionFactory.createConnection(String username, String password)`**
Le descripteur de déploiement de l'application contient une référence de ressource pour la fabrique de connexions dont la propriété **`res-auth`** est définie sur *Conteneur* et l'application a été configurée avec un alias d'authentification et la fabrique de connexions est configurée pour utiliser un canal IBM MQ dont MCAUSER est défini sur un identificateur utilisateur	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.	L'identificateur utilisateur indiqué par la propriété MCAUSER sur le canal IBM MQ devant être utilisé par la fabrique de connexions est transmis à IBM MQ.

Utilisation d'une fabrique de connexions à l'aide d'une recherche directe

Lorsqu'une fabrique de connexions de fournisseur de messagerie IBM MQ a été définie, une application d'entreprise peut rechercher la définition de fabrique de connexions et l'utiliser pour créer une connexion JMS à un gestionnaire de files d'attente IBM MQ. Cela peut s'effectuer par le biais d'une recherche directe.

Pour utiliser une recherche directe, une application d'entreprise se connecte au référentiel JNDI du serveur d'applications en lançant l'appel de méthode suivant :

```
InitialContext ctx = new InitialContext();
```

Une fois connectée au référentiel JNDI, l'application d'entreprise identifie la définition de fabrique de connexions à l'aide du nom JNDI de la fabrique de connexions, comme suit :

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("jms/myCF");
```

Remarques :

- Votre développeur d'applications doit connaître le nom JNDI de la fabrique de connexions requise lorsque l'application d'entreprise est en cours de développement. Etant donné que le nom JNDI est codé en dur dans l'application, si le nom JNDI change, vous devez réécrire et redéployer l'application.
- Lorsqu'une définition de fabrique de connexions est utilisée de cette manière, le nom d'utilisateur et le mot de passe spécifiés dans l'alias d'authentification (devant être utilisés par la fabrique de connexions) ne sont pas transmis à IBM MQ. Cela a pour but d'éviter que les applications non autorisées identifient la fabrique de connexions et puissent se connecter à des systèmes IBM MQ sécurisés.

Le nom d'utilisateur et le mot de passe qui sont transmis à IBM MQ dépendent de la méthode qui est utilisée pour créer la connexion JMS à partir de la fabrique de connexions.

Si une application crée une connexion JMS à l'aide de la méthode :

```
ConnectionFactory.createConnection()
```

l'identité d'utilisateur par défaut est transmise à IBM MQ. Il s'agit du nom d'utilisateur et du mot de passe qui ont démarré le serveur d'applications sur lequelle s'exécute l'application d'entreprise.

Sinon, une application peut créer une connexion JMS en appelant la méthode suivante :

```
ConnectionFactory.createConnection(String username, String password)
```

Si une application a effectué une recherche directe d'une fabrique de connexions, puis appelé cette méthode, le nom d'utilisateur et le mot de passe qui ont été indiqués dans la méthode `createConnection()` sont transmis à IBM MQ.

Important : Avant IBM MQ 8.0, IBM MQ traitait une vérification d'autorisation uniquement dans le but de s'assurer que le nom d'utilisateur transmis était autorisé à accéder au gestionnaire de files d'attente.

Aucune vérification n'est effectuée sur le mot de passe. Pour effectuer une vérification d'authentification et confirmer que le nom d'utilisateur et le mot de passe sont valides, il convient d'écrire un exit de sécurité de canal IBM MQ. Pour plus de détails sur la procédure à suivre, voir «Programmes d'exit de sécurité de canal», à la page 996.

Dans IBM MQ 8.0, le gestionnaire de files d'attente vérifie le mot de passe en plus du nom d'utilisateur.

Utilisation d'une fabrique de connexions à l'aide d'une recherche indirecte

Si, lorsque vous écrivez une application d'entreprise, le nom JNDI de la fabrique de connexions est inconnu ou si l'application doit être installée sur des serveurs d'applications différents à l'aide d'une fabrique de connexions différente, avec un nom JNDI différent (en fonction du serveur d'applications sur lequel elle est installée), la fabrique de connexions peut être recherchée à l'aide d'une référence de ressource. Cette opération peut être réalisée via une recherche indirecte.

Exemple

Au lieu de rechercher la fabrique de connexions en utilisant `jms/myCF`, une application d'entreprise contient une référence de ressource ayant le nom JNDI local suivant : `jms/myResourceReferenceCF`.

Pour utiliser ce nom JNDI, l'application se connecte au référentiel JNDI du serveur d'applications comme si l'application effectuait une recherche directe :

```
InitialContext ctx = new InitialContext();
```

Au lieu d'identifier `jms/myCF` directement, l'application identifie maintenant le nom JNDI de la référence de ressource :

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

Vous devez ajouter le préfixe `java:comp/env` au nom JNDI local pour indiquer au serveur d'applications que l'application d'entrperise effectue une recherche directe.

Lorsque l'application est détruite, l'utilisateur mappe le nom JNDI de la référence de ressource `jms/myResourceReferenceCF` vers le nom JNDI de la fabrique de connexions que l'application a déjà créée : `jms/myCF`.

Lorsque l'application est exécutée, elle recherche une fabrique de connexions JMS à l'aide du nom JNDI local vers lequel est mappé le serveur d'applications : `jms/myCF`. Cette fabrique de connexions est ensuite utilisée par l'application pour créer une connexion à IBM MQ.

Alias d'authentification et recherches indirectes

Une référence de ressource permet également de définir des propriétés supplémentaires qui modifient le comportement de la fabrique de connexions fournie. L'une des propriétés d'une référence de ressource est **`res-auth`**. La valeur de cette propriété indique si l'application d'entreprise doit utiliser l'alias d'authentification de la fabrique de connexions vers laquelle la référence de ressource est mappée lors de la création d'une connexion à IBM MQ (si un alias d'authentification a été défini) ou si l'application spécifie son propre nom d'utilisateur et mot de passe.

La valeur par défaut de cette propriété est *Application*. Cela signifie que le nom d'utilisateur et le mot de passe qui sont transmis au gestionnaire de files d'attente lorsqu'une connexion JMS est créée est déterminé par l'application elle-même. L'alias d'authentification de la fabrique de connexions vers lequel la référence de ressource est mappée n'est pas utilisé.

Les applications peuvent créer des connexions JMS en suivant l'une des méthodes ci-dessous :

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Si une application utilise `ConnectionFactory.createConnection()` et si **res-auth** a la valeur *Application*, l'identité d'utilisateur par défaut est transmis à IBM MQ. Il s'agit du nom d'utilisateur et du mot de passe qui ont démarré le serveur d'applications sur lequelle s'exécute l'application d'entreprise.

Si une application utilise `ConnectionFactory.createConnection(String username, String password)` et que **res-auth** est défini sur *Application*, le nom d'utilisateur et le mot de passe transmis à la méthode sont envoyés à IBM MQ.

Pour utiliser l'alias d'authentification défini sur la fabrique de connexions vers laquelle est mappée la référence de ressource lors de la création d'une connexion, vous devez définir la propriété **res-auth** à la valeur *Container*. Lorsqu'une application crée une connexion JMS, les détails de l'alias d'authentification sont utilisés, même si l'appel `createConnection` indique un nom d'utilisateur et un mot de passe.

Remplacement de l'alias d'authentification lors de l'utilisation d'une recherche indirecte

Si une application utilise une référence de ressource dont la propriété **res-auth** a la valeur *Container*, vous pouvez remplacer l'alias d'authentification qui est utilisé lors de la création des connexions JMS.

Pour remplacer l'alias d'authentification, la référence de ressource doit inclure une propriété supplémentaire appelée **authDataAlias** qui est mappée vers un alias d'authentification déjà créé dans l'environnement de serveur d'applications dans lequel l'application sera déployée. Vous pouvez spécifier cette propriété sur toutes les références de ressources qui sont créées à l'aide des outils Rational fournis par IBM.

A l'aide de cette méthode, vous pouvez utliser un alias d'authentification lors de l'utilisation d'une fabrique de connexions JMS ayant été recherchée selon la méthode indirecte. Si l'alias d'authentification indiqué n'existe pas, un nouvel alias peut être spécifié une fois que l'application d'entreprise a été installée. Pour plus d'informations, voir *Références de ressource* dans la documentation du produit WebSphere Application Server.

Informations liées à WebSphere Application Server 8.5.5

Références de ressource

Informations liées à WebSphere Application Server 8.0

Références de ressource

Informations liées à WebSphere Application Server 7.0

Références de ressource

Equilibrage de la charge de travail pour les beans gérés par message lors de l'utilisation de clusters WebSphere Application Server

Lors de l'utilisation d'applications de bean géré par message déployées dans un cluster WebSphere Application Server 7.0 et WebSphere Application Server 8.0, et configurées pour s'exécuter dans le fournisseur de messagerie IBM MQ en mode normal, l'un des membres du cluster traite la majorité des messages. Vous pouvez équilibrer la charge de travail des membres du cluster afin de répartir le traitement des messages sur plusieurs membres du cluster.

IBM MQ inclut une fonction appelée **Asynchronous consume**, qui permet aux applications de consommer des messages de manière asynchrone à partir d'une file d'attente à l'aide d'API appelées **MQCB** et **MQCTL**.

Les applications de bean géré par message s'exécutant dans WebSphere Application Server 7.0 et WebSphere Application Server 8.0, et utilisant le fournisseur de messagerie IBM MQ en mode normal utiliseront automatiquement cette fonction. Lorsque les applications démarrent, elles configurent un consommateur asynchrone sur la destination JMS qui a été configurée à des fins de surveillance en appelant **MQCB**. L'API **MQCTL** est alors appelée pour indiquer que l'application est prête à recevoir des messages provenant de la destination JMS.

Lorsque les applications de bean géré par message ont été déployées dans un cluster WebSphere Application Server, chaque membre du cluster configure un consommateur asynchrone pour la destination JMS qui est surveillée pour les messages par le bean géré par message. Le gestionnaire de files d'attente IBM MQ qui héberge la destination JMS est alors chargé de notifier le membre du cluster de la présence d'un message approprié sur la destination JMS à traiter.

Lorsqu' WebSphere Application Server se connecte à un gestionnaire de files d'attente IBM MQ , les messages qui arrivent sur une destination JMS sont distribués de manière plus uniforme à tous les destinataires asynchrones qui ont été enregistrés sur cette destination JMS . En ce qui concerne les applications de bean géré par message déployées dans un cluster WebSphere Application Server 7.0 et WebSphere Application Server 8.0, cela signifie que les messages seront distribués plus équitablement entre les membres du cluster.

Tâches associées

Configuration de la propriété JMS **PROVIDERVERSION**

Utilisation du package IBM MQ Headers

Le package IBM MQ Headers fournit un ensemble d'interfaces et de classes auxiliaires que vous pouvez utiliser pour manipuler les en-têtes IBM MQ d'un message. En règle générale, vous utilisez le package IBM MQ Headers car vous souhaitez exécuter des services d'administration à l'aide du serveur de commandes (à l'aide des messages PCF (Programmable Command Format)).

Pourquoi et quand exécuter cette tâche

Le package IBM MQ Headers se trouve dans les packages `com.ibm.mq.headers` et `com.ibm.mq.headers.pcf` . Vous pouvez utiliser cette fonction pour les deux autres API fournies par IBM MQ dans les applications Java :

- IBM MQ classes for Java (également appelé IBM MQ Base Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, également appelé IBM MQ JMS).

Les applications IBM MQ Base Java manipulent généralement les objets MQMessage et les classes de prise en charge des en-têtes peuvent interagir directement avec ces objets, car elles comprennent de manière native les interfaces IBM MQ Base Java .

Dans IBM MQ JMS, le contenu d'un message est généralement une chaîne ou un objet de tableau d'octets, qui peut être manipulé avec des flux DataInput et DataOutput . Le package IBM MQ Headers peut être utilisé pour interagir avec ces flux de données et permet de manipuler les messages MQ envoyés et reçus par les applications IBM MQ JMS .

Par conséquent, bien que le package IBM MQ Headers contienne des références au package IBM MQ Base Java , il est également destiné à être utilisé dans les applications IBM MQ JMS et peut être utilisé dans les environnements Java Platform, Enterprise Edition (Java EE).

Une façon typique d'utiliser le package IBM MQ Headers consiste à manipuler les messages d'administration au format PCF (Programmable Command Format), par exemple pour l'une des raisons suivantes:

- Permet d'accéder aux détails d'une ressource IBM MQ .
- Permet de surveiller la longueur d'une file d'attente.
- Pour empêcher l'accès à une file d'attente.

En utilisant les messages PCF avec l'API IBM MQ JMS , ce type d'administration des ressources centrées sur les applications peut être effectué à partir des applications Java EE sans avoir à utiliser l'API IBM MQ Base Java .

Procédure

- Pour utiliser le package IBM MQ Headers afin de manipuler les en-têtes de message pour IBM MQ classes for Java, voir «utilisation avecIBM MQ classes for Java», à la page 529.
- Pour utiliser le package IBM MQ Headers afin de manipuler les en-têtes de message pour IBM MQ classes for JMS, voir «utilisation avecIBM MQ classes for JMS», à la page 529.

utilisation avecIBM MQ classes for Java

Les applications IBM MQ classes for Java manipulent généralement les objets MQMessage et les classes de prise en charge des en-têtes peuvent interagir directement avec ces objets, car elles comprennent en mode natif les interfaces IBM MQ classes for Java .

Pourquoi et quand exécuter cette tâche

IBM MQ fournit des exemples d'application qui montrent comment utiliser le package IBM MQ Headers avec l'API IBM MQ Base Java (IBM MQ classes for Java).

Les exemples montrent deux choses:

- Comment créer un message PCF pour effectuer une action d'administration et analyser le message de réponse.
- Comment envoyer ce message PCF à l'aide de IBM MQ classes for Java.

Selon la plateforme que vous utilisez, ces exemples sont installés sous le répertoire `pcf` dans le répertoire `samples` ou `tools` de votre installation IBM MQ (voir «Répertoires d'installation pour IBM MQ classes for Java», à la page 364).

Procédure

1. Créez un message PCF pour effectuer une action d'administration et analyser le message de réponse.
2. Envoyez ce message PCF à l'aide du IBM MQ classes for Java.

Concepts associés

«Traitement des en-têtes de message IBM MQ avec IBM MQ classes for Java», à la page 392
Des classes Java sont fournies pour représenter différents types d'en-tête de message. Deux classes auxiliaires sont également fournies.

«Traitement des messages PCF avec IBM MQ classes for Java», à la page 397
Les classes Java sont fournies pour créer et analyser des messages structurés PCF et pour faciliter l'envoi de demandes PCF et la collecte de réponses PCF.

utilisation avecIBM MQ classes for JMS

Pour utiliser les en-têtes IBM MQ avec IBM MQ classes for JMS, vous devez effectuer les mêmes étapes essentielles que pour IBM MQ classes for Java. Le message PCF peut être créé et la réponse analysée exactement de la même manière à l'aide du package IBM MQ Headers et du même exemple de code que pour IBM MQ classes for Java.

Pourquoi et quand exécuter cette tâche

Pour envoyer un message PCF à l'aide de l'API IBM MQ , la charge de message doit être écrite dans un message JMS Bytes Message et envoyée à l'aide des API JMS standard. La seule considération est que le message ne doit pas contenir d'en-tête JMS RFH2 ni aucun autre en-tête avec des valeurs spécifiques dans le MQMD.

Pour envoyer un message PCF, procédez comme suit. La façon dont le message PCF est créé et dont les informations sont extraites du message de réponse est la même que pour IBM MQ classes for Java (voir «utilisation avecIBM MQ classes for Java», à la page 529).

Procédure

1. Créez une destination de file d'attente JMS qui représente SYSTEM.ADMIN.COMMAND.QUEUE.

   Les applications IBM MQ JMS envoient les messages PCF à SYSTEM.ADMIN.COMMAND.QUEUEet besoin d'accéder à un objet de destination JMS qui représente cette file d'attente. Les propriétés suivantes doivent être définies pour la destination:

   ```
   WMQ_MQMD_WRITE_ENABLED = YES
   WMQ_MESSAGE_BODY = MQ
   ```

   Si vous utilisez WebSphere Application Server, vous devez définir ces propriétés en tant que propriétés personnalisées sur la destination.

   Pour créer la destination à l'aide d'un programme à partir d'une application, utilisez le code suivant:

   ```
   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. Convertissez un message PCF en message JMS Bytes contenant les valeurs MQMD correctes.

   Un message JMS Bytes doit être créé et le message PCF doit y être écrit. Une file d'attente de réponses doit être créée, mais elle ne doit pas comporter de paramètres spécifiques.

   L'exemple de fragment de code suivant montre comment créer un message JMS Bytes Message et y écrire un objet com.ibm.mq.headers,pcf.PCFMessage . L'objet PCFMessage (pcfCmd) a déjà été généré à l'aide du package IBM MQ Headers. (Notez que le package permettant de charger PCFMessage est 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. Envoyez le message et recevez la réponse à l'aide des API JMS standard.
4. Convertissez le message de réponse en message PCF à traiter.

   Pour extraire le message de réponse et le traiter en tant que message PCF, utilisez le code suivant:

   ```
    // 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);
```

Concepts associés

«Messages JMS», à la page 148
Les messages JMS sont composés d'un en-tête, de propriétés et d'un corps. JMS définit cinq types de corps de message.

IBM i Configuration de IBM MQ sous IBM i avec Java et JMS

Cette collection de rubriques présente comment configurer et tester IBM MQ avec Java et JMS sur IBM i à l'aide de commandes CL ou de l'environnement qshell.

Remarque :

- Depuis IBM MQ 8.0, `ldap.jar`, `jndi.jar` et `jta.jar` font partie du JDK.
- JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Utilisation des commandes CL

La variable CLASSPATH que vous définissez est destinée aux tests avec MQ base Java, JMS avec JNDI et JMS sans JNDI.

Si vous n'utilisez pas de fichier `.profile` dans votre répertoire `/home/Userprofile` , vous devez définir les variables d'environnement de niveau système ci-dessous. Vous pouvez vérifier s'ils sont définis à l'aide de la commande **WRKENVVAR** .

1. Pour afficher les variables d'environnement de l'ensemble du système, exécutez la commande suivante: **WRKENVVAR LEVEL (*SYS)**
2. Pour afficher les variables d'environnement spécifiques à votre travail, exécutez la commande suivante: **WRKENVVAR LEVEL (*JOB)**
3. Si la variable CLASSPATH n'est pas définie, exécutez la commande suivante:

JM 3.0
```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

JMS 2.0
```
ADDENVVAR ENVVAR(CLASSPATH)
       VALUE('.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar
       :/QIBM/ProdData/mqm/java/lib/connector.jar:/QIBM/ProdData/mqm/java/lib
       :/QIBM/ProdData/mqm/java/samples/base
       :/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar
       :/QIBM/ProdData/mqm/java/lib/jms.jar
       :/QIBM/ProdData/mqm/java/lib/providerutil.jar
       :/QIBM/ProdData/mqm/java/lib/fscontext.jar:') LEVEL(*SYS)
```

4. Si QIBM_MULTI_THREADED n'est pas défini, exécutez la commande suivante:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Si QIBM_USE_DESCRIPTOR_STDIO n'est pas défini, exécutez la commande suivante:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Si QSH_REDIRECTION_TEXTDATA n'est pas défini, exécutez la commande suivante:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Utilisation de l'environnement qshell

Si vous utilisez l'environnement QSHELL, vous pouvez configurer un `.profile` dans votre répertoire `/home/Userprofile` . Pour plus d'informations, reportez-vous à la documentation Qshell Interpreter (qsh).

Spécifiez ce qui suit dans le fichier `.profile`. Notez que l'instruction CLASSPATH doit se trouver sur une seule ligne ou être séparée sur des lignes différentes à l'aide du caractère \ , comme illustré.

JM 3.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jakarta.client.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

JMS 2.0

```
CLASSPATH=.:/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: \
/QIBM/ProdData/mqm/java/lib/connector.jar: \
/QIBM/ProdData/mqm/java/lib: \
/QIBM/ProdData/mqm/java/samples/base: \
/QIBM/ProdData/mqm/java/lib/com.ibm.mq.allclient.jar: \
/QIBM/ProdData/mqm/java/lib/jms.jar: \
/QIBM/ProdData/mqm/java/lib/providerutil.jar: \
/QIBM/ProdData/mqm/java/lib/fscontext.jar:
HOME=/home/XXXXX
LOGNAME=XXXXX
PATH=/usr/bin:
QIBM_MULTI_THREADED=Y QIBM_USE_DESCRIPTOR_STDIO=I
QSH_REDIRECTION_TEXTDATA=Y
TERMINAL_TYPE=5250
```

Vérifiez que la bibliothèque QMQMJAVA figure dans la liste des bibliothèques en exécutant la commande **DSPLIBL**.

Si la bibliothèque QMQMJAVA ne figure pas dans la liste, ajoutez-la à l'aide de la commande suivante: **ADDLIBLE LIB (QMQMJAVA)**

IBM i Test de IBM MQ sur IBM i avec Java

Comment tester IBM MQ avec Java à l'aide de l'exemple de programme MQIVP.

Test de IBM MQ base Java

Effectuez la procédure suivante :

1. Vérifiez que le gestionnaire de files d'attente est démarré et que son état est ACTIF, en exécutant la commande suivante:

```
WRKMQM MQMNAME(QMGRNAME)
```

2. Vérifiez que JAVA.CHANNEL a été créé à l'aide de la commande suivante:

```
WRKMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

   a. Si JAVA.CHANNEL n'existe pas, exécutez la commande suivante:

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
```

3. Vérifiez que le programme d'écoute du gestionnaire de files d'attente est en cours d'exécution pour le port 1414 ou quel que soit le port que vous utilisez, en exécutant la commande **WRKMQMLSR** .

   a. Si aucun programme d'écoute n'a été démarré pour le gestionnaire de files d'attente, exécutez la commande suivante:

```
STRMQMLSR PORT(xxxx) MQMNAME(QMGRNAME)
```

Exécution de l'exemple de programme de test MQIVP

1. Démarrez qshell à partir de la ligne de commande en exécutant la commande STRQSH
2. Vérifiez que la variable CLASSPATH correcte est définie à l'aide de la commande **export** , puis exécutez la commande **cd** comme suit:

```
cd /qibm/proddata/mqm/java/samples/wmqjava/samples
```

3. Exécutez le programme **java** en exécutant la commande suivante:

```
java MQIVP
```

Vous pouvez appuyer sur la touche ENTREE lorsque vous êtes invité à:

- Type de connexion
- Adresse IP
- Nom gest. de files

pour utiliser les valeurs par défaut. Cette opération vérifie les liaisons de produit, qui se trouvent dans la bibliothèque QMQMJAVA.

Vous recevez une sortie similaire à l'exemple suivant. Notez que la déclaration de copyright dépend de la version du produit que vous utilisez.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :>
Please enter the queue manager name :>
Attaching Java program to QIBM/ProdData/mqm/java/lib/connector.JAR.
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
```

```
Press Enter to continue ...>
$
```

Test de la connexion client IBM MQ Java

Vous devez spécifier les éléments suivants:

- Type de connexion
- Adresse IP
- Port
- Canal de connexion serveur
- Gestionnaire de files d'attente

Vous recevez une sortie similaire à l'exemple suivant. Notez que la déclaration de copyright dépend de la version du produit que vous utilisez.

```
> java MQIVP
MQSeries for Java Installation Verification Program
5724-H72 (C) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
============================================================

Please enter the IP address of the MQ server :> x.xx.xx.xx
Please enter the port to connect to : (1414)> 1470
Please enter the server connection channel name :> JAVA.CHANNEL
Please enter the queue manager name :> KAREN01
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...>
$
```

IBM i Test de IBM MQ sur IBM i avec JMS

Comment tester IBM MQ avec JMS avec et sans JNDI

Test de JMS sans JNDI à l'aide de l'exemple IVTRun

Effectuez la procédure suivante :

1. Vérifiez que le gestionnaire de files d'attente est démarré et que son état est ACTIF, en exécutant la commande suivante:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Démarrez l'interpréteur de commandes qshell à partir de la ligne de commande en exécutant la commande **STRQSH** .
3. Utilisez la commande **cd** pour changer de répertoire comme suit:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

4. Exécutez le fichier script:

   ```
   IVTRun -nojndi [-m qmgrname]
   ```

Vous recevez une sortie similaire à l'exemple suivant. Notez que les mentions de droits d'auteur dépendent des versions des produits que vous utilisez:

```
 IVTRun -nojndi -m ELCRTP19

Attaching Java program to
/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.JAR.
Attaching Java program to
/QIBM/ProdData/mqm/java/lib/jms.JAR.

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000c012
JMSTimestamp: 1020273404500
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:QP0ZSPWT STANLEY 170302
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMS_IBM_PutTime:13441354
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$>
$
```

Test du mode client IBM MQ JMS sans JNDI

Effectuez la procédure suivante :

1. Vérifiez que le gestionnaire de files d'attente est démarré et que son état est ACTIF, en exécutant la commande suivante:

   ```
   WRKMQM MQMNAME(QMGRNAME)
   ```

2. Vérifiez que le canal de connexion serveur est créé en exécutant la commande suivante:

   ```
   WRKMQMCHL CHLNAME( SYSTEM.DEF.SVRCONN ) CHLTYPE(*SVRCN)
   MQMNAME(QMGRNAME)
   ```

3. Vérifiez que le programme d'écoute est démarré pour le port approprié, en exécutant la commande **WRKMQMLSR**

4. Démarrez l'interpréteur de commandes qshell à partir de la ligne de commande en exécutant la commande **STRQSH** .
5. Vérifiez que la variable CLASSPATH est correcte en exécutant la commande **export** .
6. Utilisez la commande **cd** pour changer de répertoire comme suit:

```
cd /qibm/proddata/mqm/java/bin
```

7. Exécutez le fichier script:

```
IVTRun -nojndi -client -m QMgrName -host hostname [-port port] [-channel channel]
```

Vous recevez une sortie similaire à l'exemple suivant. Notez que les mentions de droits d'auteur dépendent des versions des produits que vous utilisez.

```
> IVTRun -nojndi -client -m ELCRTP19 -host ELCRTP19 -port 1414 -channel SYSTEM.DEF.SVRCONN

5724-H72, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2011, 2024.
All Rights Reserved.
WebSphere MQ classes for Java Message Service 5.300
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000d012
JMSTimestamp: 1020274009970
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_PutDate:20040326
JMSXAppID:MQSeries Client for Java
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:28
JMS_IBM_MsgType:8
JMSXUserID:QMQM
JMS_IBM_PutTime:14085237
JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Test de IBM MQ JMS avec JNDI

Vérifiez que le gestionnaire de files d'attente est démarré et que son état est ACTIF, en exécutant la commande suivante:

```
WRKMQM MQMNAME(QMGRNAME)
```

Utilisation de l'exemple de script de test IVTRun

Effectuez la procédure suivante :

1. Apportez les modifications appropriées au fichier `JMSAdmin.config` . Pour éditer ce fichier, utilisez la commande **EDTF** (Editer un fichier) à partir d'une ligne de commande IBM i

   ```
   EDTF '/qibm/proddata/mqm/java/bin/JMSAdmin.config'
   ```

   .

   a. Pour utiliser LDAP pour Weblogic, supprimez le commentaire de:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
      ```

   b. Pour utiliser LDAP pour WebSphere Application Server, supprimez le commentaire de:

      ```
      INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
      ```

   c. Pour tester le système de fichiers, supprimez le commentaire de:

      ```
      INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
      ```

   d. Vérifiez que vous avez sélectionné l'URL PROVIDER_URL correcte, en supprimant le commentaire de la ligne appropriée.

   e. Mettez en commentaire toutes les autres lignes à l'aide du symbole *#* .

   f. Une fois que vous avez terminé toutes vos modifications, appuyez sur **F2=Save** et **F3=Exit**.

2. Démarrez l'interpréteur de commandes qshell à partir de la ligne de commande en exécutant la commande **STRQSH** .
3. Vérifiez que la variable CLASSPATH est correcte en exécutant la commande **export** .
4. Utilisez la commande **cd** pour changer de répertoire comme suit:

   ```
   cd /qibm/proddata/mqm/java/bin
   ```

5. Démarrez le script **IVTSetup** pour créer les objets gérés (*MQQueueConnectionFactory* et *MQQueue*) en exécutant la commande **IVTSetup** .
6. Exécutez le script IVTRun en exécutant la commande suivante:

   ```
   IVTRun -url providerURL [-icf initCtxFact]
   ```

Vous recevez une sortie similaire à l'exemple suivant. Notez que les mentions de droits d'auteur dépendent des versions des produits que vous utilisez.

```
> IVTSetup
+ Creating script for object creation within JMSAdmin
+ Calling JMSAdmin in batch mode to create objects
Ignoring unknown flag: -i

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
Starting WebSphere MQ classes for Java Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java Message Service Administration

+ Administration done; tidying up files
+ Done!
$
> IVTRun -url file:////tmp/mqjms -icf com.sun.jndi.fscontext.RefFSContextFactory

5724-H72 (c) Copyright IBM Corp. 2011, 2024. All Rights Reserved.
```

```
MQSeries classes for Java Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:c1d4d840c5d3c3d9e3d7f1f9404040403ccf041f0000e012
JMSTimestamp: 1020274903770
JMSCorrelationID:null
JMSDestination: queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo: null
JMSRedelivered: false
JMS_IBM_Format:MQSTR
JMS_IBM_PutApplType:8
JMSXDeliveryCount:1
JMS_IBM_MsgType:8
JMSXUserID:STANLEY
JMSXAppID:QP0ZSPWT STANLEY 170308
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
$
```

Développement d'applications Java à l'aide d'un référentiel Maven

Lors du développement d'une application Java pour IBM MQ, en utilisant un référentiel Maven pour installer automatiquement les dépendances, vous n'avez pas besoin d'installer explicitement quoi que ce soit avant d'utiliser les interfaces IBM MQ .

Référentiel Maven Central

Maven est un outil permettant de générer des applications et fournit également un référentiel pour le stockage des artefacts auxquels votre application peut souhaiter accéder.

Le référentiel Maven (ou référentiel central) possède une structure qui permet aux fichiers tels que les fichiers JAR d'avoir des versions distinctes qui sont ensuite facilement reconnues à l'aide d'un mécanisme de dénomination bien connu. Les outils de génération peuvent ensuite utiliser ces noms pour extraire dynamiquement les dépendances de votre application. Dans la définition de votre application, qui, lors de l'utilisation de Maven en tant qu'outil de génération, est appelé fichier POM, vous nommez les dépendances et le processus de génération sait ce qu'il faut faire à partir de là.

Fichiers client IBM MQ

Des copies des interfaces client IBM MQ Java sont disponibles dans le référentiel central sous `com.ibm.mq GroupId`. Vous trouverez le fichier `com.ibm.mq.jakarta.client.jar` (Jakarta Messaging 3.0) et le fichier `com.ibm.mq.allclient.jar` (JMS 2.0). Ces fichiers sont généralement utilisés pour les programmes autonomes. Vous trouverez également le fichier `wmq.jakarta.jmsra.rar` (Jakarta Messaging 3.0) et le fichier `wmq.jmsra.rar` (JMS 2.0), qui doit être utilisé dans les serveurs d'applications Java EE . Le fichier `jakarta.client.jar` et le fichier

`allclient.jar` contiennent tous deux le fichier IBM MQ classes for JMS et le fichier IBM MQ classes for Java.

Important : L'utilisation du format Apache Maven Assembly Plugin *jar-with-dependencies* pour générer une application qui inclut le fichier JAR relocalisable IBM MQ n'est pas prise en charge.

Dans un fichier `pom.xml` traité par la commande maven, vous ajoutez des dépendances pour ces fichiers JAR comme illustré dans les exemples suivants:

- JM 3.0 Pour afficher la relation entre votre code d'application et `com.ibm.mq.jakarta.client.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.jakarta.client</artifactId>
        <version>9.3.0.0</version>
    </dependency>
```

- JMS 2.0 Pour afficher la relation entre votre code d'application et `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.2.2.0</version>
    </dependency>
```

- JM 3.0 Pour utiliser l'adaptateur de ressources Jakarta EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jakarta.jmsra</artifactId>
     <version>9.3.0.0</version>
 </dependency>
```

- JMS 2.0 Pour utiliser l'adaptateur de ressources JMS 2.0 Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
     <version>9.2.2.0</version>
 </dependency>
```

Pour un exemple de projet simple dans Eclipse permettant d'exécuter un projet JMS , voir l'article IBM Developer Developing Java applications for MQ just got ??? with Maven.

Développement d'applications C++

IBM MQ fournit des classes C++ équivalentes à des objets IBM MQ et des classes supplémentaires équivalentes aux types de données de tableau. Il fournit un certain nombre de fonctions qui ne sont pas disponibles via l'interface MQI.

IBM WebSphere MQ 7.0, les améliorations apportées aux interfaces de programmation IBM MQ ne sont pas appliquées aux classes C + +.

IBM MQ C++ fournit les fonctions suivantes:

- Initialisation automatique des structures de données IBM MQ .
- Connexion et ouverture de file d'attente du gestionnaire de files d'attente dans le temps.
- Fermeture implicite de la file d'attente et déconnexion du gestionnaire de files d'attente.
- Transmission et réception de l'en-tête de rebut.
- Transmission et réception de l'en-tête de pont IMS .
- Transmission et réception de l'en-tête de message de référence.
- Déclencher la réception des messages.

- Transmission et réception de l'en-tête CICS bridge .
- Transmission et réception de l'en-tête de travail.
- Définition de canal du client.

Les diagrammes de classes Booch suivants montrent que toutes les classes sont globalement parallèles aux entités IBM MQ dans l'interface MQI procédurale (par exemple, à l'aide de C) qui ont des descripteurs ou des structures de données. Toutes les classes héritent de la classe ImqError (voir ImqError C++ class), qui permet d'associer une condition d'erreur à chaque objet.

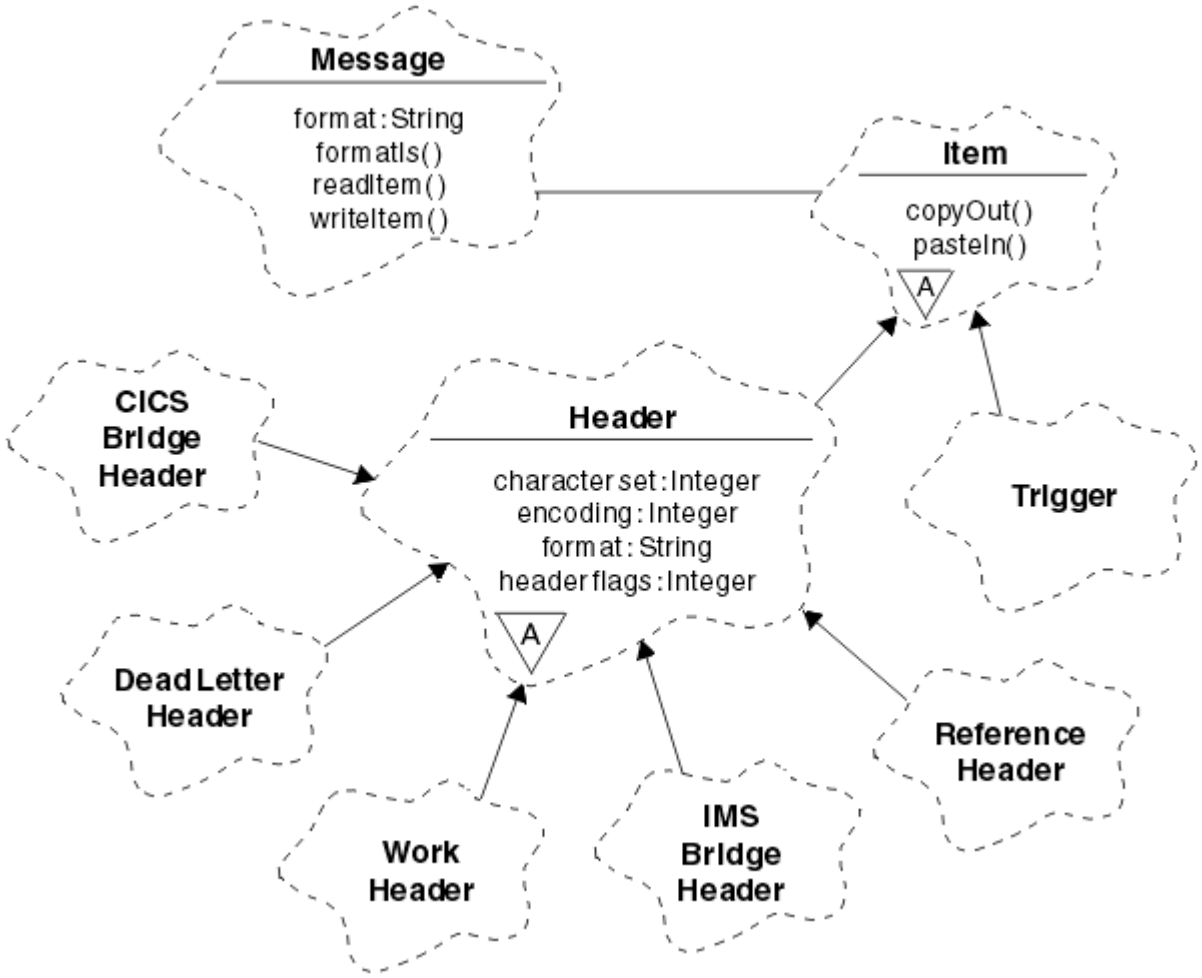


Figure 50. Classes C++ IBM MQ (gestion des éléments)

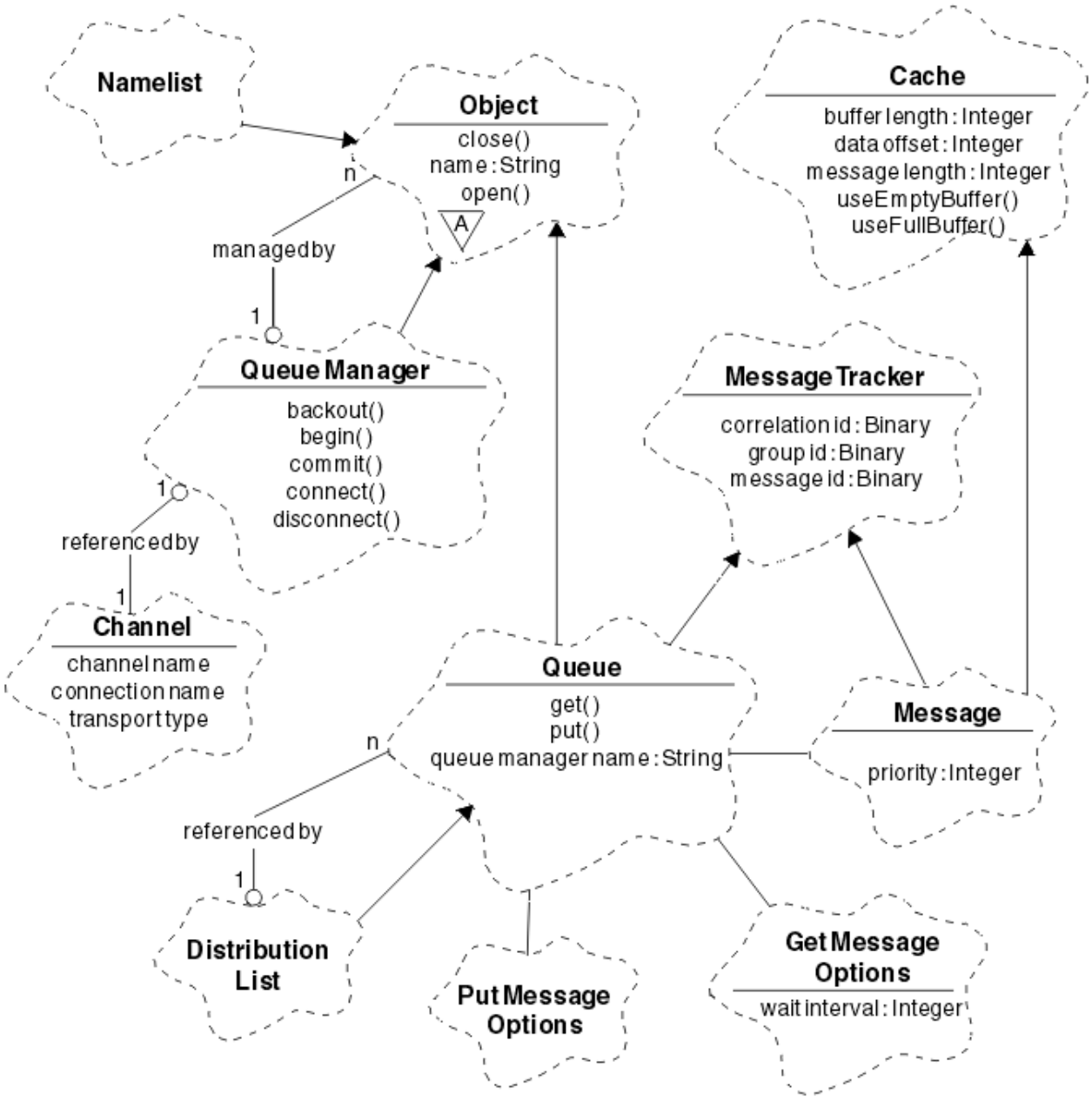


Figure 51. Classes C++ IBM MQ (gestion des files d'attente)

Pour interpréter correctement les diagrammes de classes Booch, tenez compte des conventions suivantes:

- Les méthodes et les attributs dignes de mention sont affichés sous le nom *class* .
- Un petit triangle dans un nuage indique une *classe abstraite*.
- *Héritage* est indiqué par une flèche vers la classe parent.
- Une ligne non décorée entre les nuages indique une *relation de coopération* entre les classes.
- Une ligne décorée d'un nombre indique une *relation référentielle* entre deux classes. Ce nombre indique le nombre d'objets pouvant participer à une relation particulière à un moment donné.

Les classes et types de données suivants sont utilisés dans les signatures de méthode C++ des classes de gestion de file d'attente (voir Figure 51, à la page 541) et les classes de traitement des articles (voir Figure 50, à la page 540):

- La classe ImqBinary (voir Classe C++ImqBinary), qui encapsule des tableaux d'octets tels que MQBYTE24.
- Le type de données ImqBoolean , qui est défini comme **typedef unsigned char ImqBoolean**.
- La classe ImqString (voir ImqString C++ class), qui encapsule des tableaux de caractères tels que MQCHAR64.

Les entités avec des structures de données sont subsumées dans les classes d'objets appropriées. Zones de structure de données individuelles (voir Références croisées C++ et MQI) sont accessibles à l'aide de méthodes.

Les entités avec des descripteurs sont placées dans la hiérarchie de classe ImqObject (voir ImqObject C++ class) et fournissent des interfaces encapsulées à l'interface MQI. Les objets de ces classes présentent un comportement intelligent qui peut réduire le nombre d'appels de méthode requis par rapport à l'interface MQI procédurale. Par exemple, vous pouvez établir et supprimer des connexions de gestionnaire de files d'attente selon les besoins, ou vous pouvez ouvrir une file d'attente avec les options appropriées, puis la fermer.

La classe ImqMessage (voir Classe C++ImqMessage) encapsule la structure de données MQMD et sert également de point de rétention pour les données utilisateur et les *éléments* (voir «Lecture des messages en C++», à la page 552) en fournissant des fonctions de mémoire tampon en cache. Vous pouvez fournir des tampons de longueur fixe pour les données utilisateur et utiliser la mémoire tampon plusieurs fois. La quantité de données présentes dans la mémoire tampon peut varier d'une utilisation à l'autre. Alternativement, le système peut fournir et gérer une mémoire tampon de longueur souple. La taille de la mémoire tampon (la quantité disponible pour la réception des messages) et la quantité réellement utilisée (soit le nombre d'octets pour la transmission, soit le nombre d'octets réellement reçus) deviennent des considérations importantes.

Concepts associés

Présentation technique

«Exemples de programmes C++», à la page 542
Quatre exemples de programmes sont fournis pour illustrer l'obtention et l'insertion de messages.

«Remarques sur le langage C++», à la page 546
Cette collection de rubriques détaille les aspects de l'utilisation du langage C++ et les conventions que vous devez prendre en compte lors de l'écriture de programmes d'application qui utilisent l'interface MQI (Message Queue Interface).

«Préparation des données de message en C++», à la page 551
Les données de message sont préparées dans une mémoire tampon, qui peut être fournie par le système ou l'application. Il y a des avantages à l'une ou l'autre des méthodes. Des exemples d'utilisation d'une mémoire tampon sont donnés.

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

Référence associée

«Génération de programmes C++ IBM MQ», à la page 557
L'URL des compilateurs pris en charge est répertoriée, ainsi que les commandes à utiliser pour compiler, lier et exécuter des programmes C++ et des exemples sur les plateformes IBM MQ .

Références croisées C++ et MQI

IBM MQ classes C++

Exemples de programmes C++

Quatre exemples de programmes sont fournis pour illustrer l'obtention et l'insertion de messages.

Les exemples de programme sont les suivants:

- HELLO WORLD (imqwrld.cpp)

- SPUT (imqsput.cpp)
- SGET (imqsget.cpp)
- DPUT (imqdput.cpp)

Les exemples de programme se trouvent dans les répertoires indiqués dans le Tableau 73, à la page 543.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Tableau 73. Emplacement des exemples de programme

Environnement	Répertoire contenant la source	Répertoire contenant les éléments générés programmes
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
V 9.4.0 AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ca (voir la remarque «1», à la page 543)
IBM i	`/QIBM/ProdData/mqm/samp/`	(voir la remarque «2», à la page 543)
Linux	*MQ_INSTALLATION_PATH*/samp	Aucun
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\exemples	*MQ_INSTALLATION_PATH*\tools\cplus\exemples \bin\vn (voir la remarque «3», à la page 543)
z/OS	thlqual.SCSQCPPS	

Remarques :

1. V 9.4.0 AIX Les programmes générés à l'aide du compilateur XLC 17 se trouvent dans le dossier "`ca`", tandis que les programmes générés à l'aide du compilateur XLC 16 se trouvent dans le dossier "`ia`".
2. IBM i Les programmes générés à l'aide du compilateur ILE C++ pour IBM i se trouvent dans la bibliothèque QMQM. Les fichiers source se trouvent dans `/QIBM/ProdData/mqm/samp`.
3. Windows Les programmes générés à l'aide de Microsoft Visual Studio Visual Studio se trouvent dans *MQ_INSTALLATION_PATH*`\tools\cplus\samples\bin\vn`. Pour plus d'informations sur ces compilateurs, voir «Génération de programmes C++ sur Windows», à la page 563.

Exemple de programme HELLO WORLD (imqwrld.cpp)

Cet exemple de programme C++ montre comment placer et obtenir un datagramme standard (structure C) à l'aide de la classe ImqMessage .

Ce programme montre comment placer et obtenir un datagramme standard (structure C) à l'aide de la classe ImqMessage . Cet exemple utilise peu d'appels de méthode, tirant parti des appels de méthode implicites tels que **open**, **close**et **disconnect**.

Sur toutes les plateformes sauf z/OS

Si vous utilisez une connexion serveur à IBM MQ, suivez l'une des procédures suivantes:

- Pour utiliser la file d'attente par défaut existante, SYSTEM.DEFAULT.LOCAL.QUEUE, exécutez le programme **imqwrlds** sans transmettre de paramètres
- Pour utiliser une file d'attente affectée dynamiquement temporaire, exécutez **imqwrlds** en transmettant le nom de la file d'attente modèle par défaut, SYSTEM.DEFAULT.MODEL.QUEUE.

Si vous utilisez une connexion client à IBM MQ, suivez l'une des procédures suivantes:

- Configurez la variable d'environnement MQSERVER (voir *MQSERVER* pour plus d'informations) et exécutez **imqwrldc**, ou
- Exécutez **imqwrldc** en transmettant les paramètres **queue-name**, **queue-manager-name**et **channel-definition**, où un **channel-definition** standard peut être SYSTEM.DEF.SVRCONN/TCP/*nom_hôte* (1414)

Sous z/OS

Construisez et exécutez un travail par lots à l'aide de l'exemple de JCL **imqwrldr**.

Pour plus d'informations, voir z/OS Batch, RRS Batch et CICS .

Exemple de code

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // IBM MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
```

```
    // with the addition of an input option during the "get".
  }

  // Prepare a message containing the text "Hello world".
  pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
  pmsg -> setFormat( MQFMT_STRING );

  // Place the message on the queue, using default put message
  // Options.
  // The queue will be automatically opened with an output option.
  if ( pqueue -> put( * pmsg ) ) {
    ImqString strQueue( pqueue -> name( ) );

    // Discover the name of the queue manager.
    ImqString strQueueManagerName( manager.name( ) );
    printf( "The queue manager name is %s.\n",
            (char *)strQueueManagerName );

    // Show the name of the queue.
    printf( "Message sent to %s.\n", (char *)strQueue );

    // Retrieve the data message just sent ("Hello world" expected)
    // from the queue, using default get message options. The queue
    // is automatically closed and reopened with an input option
    // if it is not already open with an input option. We get the
    // message just sent, rather than any other message on the
    // queue, because the "put" will have set the ID of the message
    // so, as we are using the same message object, the message ID
    // acts as in the message object, a filter which says that we
    // are interested in a message only if it has this
    // particular ID.

    if ( pqueue -> get( * pmsg ) ) {
      int iDataLength = pmsg -> dataLength( );

      // Show the text of the received message.
      printf( "Message of length %d received, ", iDataLength );

      if ( pmsg -> formatIs( MQFMT_STRING ) ) {
        char * pszText = pmsg -> bufferPointer( );

        // If the last character of data is a null, then we can
        // assume that the data can be interpreted as a text
        // string.
        if ( ! pszText[ iDataLength - 1 ] ) {
          printf( "text is \"%s\".\n", pszText );
        } else {
          printf( "no text.\n" );
        }

      } else {
        printf( "non-text message.\n" );
      }
    } else {
      printf( "ImqQueue::get failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

  } else {
    printf( "ImqQueue::open/put failed with reason code %ld\n",
            pqueue -> reasonCode( ) );
    iReturnCode = (int)pqueue -> reasonCode( );
  }

  // Deletion of the queue will ensure that it is closed.
  // If the queue is dynamic then it will also be destroyed.
  delete pqueue ;
  delete pmsg ;

} else {
  printf( "ImqQueueManager::connect failed with reason code %ld\n"
          manager.reasonCode( ) );
  iReturnCode = (int)manager.reasonCode( );
}

// Destruction of the queue manager ensures that it is
// disconnected.  If the queue object were still available
// and open (which it is not), the queue would be closed
// prior to disconnection.
```

```
  return iReturnCode ;
}
```

Exemples de programmes SPUT (imqsput.cpp) et SGET (imqsget.cpp)

Ces programmes C++ placent des messages dans une file d'attente nommée et en extraient des messages.

Ces exemples montrent l'utilisation des classes suivantes:

- ImqError (voir ImqError C++ class)
- ImqMessage (voir ImqMessage C++ class)
- ImqObject (voir ImqObject C++ class)
- ImqQueue (voir la classe C++ImqQueue)
- ImqQueueManager (voir ImqQueueManager C++ class)

Suivez les instructions appropriées pour exécuter les programmes.

Sur toutes les plateformes sauf z/OS

1. Exécutez **imqsputs** *nom-file-attente*.
2. Entrez des lignes de texte sur la console. Ces lignes sont placées en tant que messages dans la file d'attente spécifiée.
3. Entrez une ligne nulle pour terminer l'entrée.
4. Exécutez **imqsgets** *queue-name* pour extraire toutes les lignes et les afficher sur la console.

z/OS Pour plus d'informations, voir «Building C++ programs on z/OS Batch, RRS Batch and CICS», à la page 565.

Sous z/OS

z/OS

1. Construisez et exécutez un travail par lots à l'aide de l'exemple de JCL **imqsputr**. Les messages sont lus à partir du fichier SYSIN.
2. Construisez et exécutez un travail par lots à l'aide de l'exemple de JCL **imqsgetr**. Les messages sont extraits de la file d'attente et envoyés au fichier SYSPRINT.

Exemple de programme DPUT (imqdput.cpp)

Cet exemple de programme C++ insère des messages dans une liste de distribution composée de deux files d'attente.

DPUT montre l'utilisation de la classe ImqDistributionList (voir ImqDistributionList C++ class). Cet exemple n'est pas pris en charge sous z/OS.

1. Exécutez **imqdputs** *queue-name-1 queue-name-2* pour placer les messages dans les deux files d'attente nommées.
2. Exécutez **imqsgets** *queue-name-1* et **imqsgets** *queue-name-2* pour extraire les messages de ces files d'attente.

Remarques sur le langage C++

Cette collection de rubriques détaille les aspects de l'utilisation du langage C++ et les conventions que vous devez prendre en compte lors de l'écriture de programmes d'application qui utilisent l'interface MQI (Message Queue Interface).

Fichiers d'en-tête C++

Les fichiers d'en-tête sont fournis dans le cadre de la définition de l'interface MQI pour vous aider à écrire des programmes d'application IBM MQ en langage C + +.

Ces fichiers d'en-tête sont récapitulés dans le tableau suivant.

Tableau 74. Fichiers d'en-tête C/C++

Nom de fichier	Contenu
IMQI.HPP	C++ MQI Classes (inclut CMQC.H et IMQTYPE.H)
IMQTYPE.H	Définit le type de données **ImqBoolean**
CMQC.H	Structures de données MQI et constantes de manifeste

Pour améliorer la portabilité des applications, codez le nom du fichier d'en-tête en minuscules sur la directive de préprocesseur **#include** :

```
#include <imqi.hpp> // C++ classes
```

Méthodes et attributs C++

Les noms de méthode sont en casse mixte. Diverses considérations s'appliquent aux paramètres et aux valeurs de retour. Les attributs sont accessibles à l'aide des méthodes set et get appropriées.

Les paramètres des méthodes *const* sont réservés aux entrées. Paramètres avec des signatures incluant un pointeur (*) ou une référence (&) sont transmises par référence. Les valeurs de retour qui n'incluent pas de pointeur ou de référence sont transmises par valeur ; dans le cas d'objets renvoyés, il s'agit de nouvelles entités qui deviennent la responsabilité de l'appelant.

Certaines signatures de méthode incluent des éléments qui prennent une valeur par défaut s'ils ne sont pas spécifiés. Ces éléments sont toujours à la fin des signatures et sont indiqués par un signe égal (=) ; la valeur après le signe égal indique la valeur par défaut qui s'applique si l'élément est omis.

Tous les noms de méthode de ces classes sont en casse mixte, commençant par des minuscules. Chaque mot, à l'exception du premier dans un nom de méthode, commence par une lettre majuscule. Les abréviations ne sont utilisées que si leur signification est largement comprise. Les abréviations utilisées incluent *id* (pour l'identité) et *sync* (pour la synchronisation).

Les attributs d'objet sont accessibles à l'aide des méthodes set et get. Une méthode set commence par le mot *set* ; une méthode get n'a pas de préfixe. Si un attribut est en *lecture seule*, il n'existe pas de méthode set.

Les attributs sont initialisés avec des états valides lors de la construction de l'objet et l'état d'un objet est toujours cohérent.

Types de données en C++

Tous les types de données sont définis par l'instruction C **typedef** .

Le type **ImqBoolean** est défini en tant que **caractère non signé** dans IMQTYPE.H et peut avoir les valeurs TRUE et FALSE. Vous pouvez utiliser des objets de classe **ImqBinary** à la place des tableaux **MQBYTE** et des objets de classe **ImqString** à la place de **char ***. De nombreuses méthodes renvoient des objets au lieu de pointeurs **char** ou **MQBYTE** pour faciliter la gestion du stockage. Toutes les valeurs de retour deviennent la responsabilité de l'appelant et, dans le cas d'un objet renvoyé, le stockage peut être supprimé à l'aide de la fonction de suppression.

Manipulation des chaînes binaires en C++

Les chaînes de données binaires sont déclarées en tant qu'objets de la classe **ImqBinary** . Les objets de cette classe peuvent être copiés, comparés et définis à l'aide des opérateurs C familiers. Un exemple de code est fourni.

L'exemple de code suivant montre des opérations sur une chaîne binaire:

```
#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.
...
```

Manipulation des chaînes de caractères en C++

Les données de type caractère sont souvent renvoyées dans les objets de classe **ImqString** qui peuvent être transtypés en **char *** à l'aide d'un opérateur de conversion. La classe ImqString contient des méthodes permettant d'aider au traitement des chaînes de caractères.

Lorsque des données de type caractère sont acceptées ou renvoyées à l'aide de méthodes MQI C++, les données de type caractère se terminent toujours par une valeur nulle et peuvent être de n'importe quelle longueur. Toutefois, certaines limites imposées par IBM MQ peuvent entraîner la troncature des informations. Pour faciliter la gestion du stockage, les données de type caractères sont souvent renvoyées dans les objets de classe **ImqString** . Ces objets peuvent être transtypés en **char *** à l'aide de l'opérateur de conversion fourni et utilisés à des fins de *lecture seule* dans de nombreuses situations où un **char *** est requis.

Remarque : Le résultat de la conversion **char *** d'un objet de classe **ImqString** peut être null.

Bien que les fonctions C puissent être utilisées sur le **char ***, il existe des méthodes spéciales de la classe **ImqString** qui sont préférables ; **operator length** () est l'équivalent de **strlen** et **storage** () indique la mémoire allouée aux données de type caractères.

Etat initial des objets en C++

Tous les objets ont un état initial cohérent reflété par leurs attributs. Les valeurs initiales sont définies dans les descriptions de classe.

Utilisation de C à partir de C++

Lorsque vous utilisez des fonctions C à partir d'un programme C + +, incluez les en-têtes appropriés.

L'exemple suivant montre `string.h` inclus dans un programme C++:

```
extern "C" {
#include <string.h>
}
```

Conventions de notation C++

Cet exemple montre comment appeler des méthodes et déclarer des paramètres.

Cet exemple de code utilise les méthodes et les paramètres **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Déclarez et utilisez les paramètres comme suit:

```
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( );

...
}
```

Opérations implicites en C++

Plusieurs opérations peuvent se produire implicitement, *juste à temps*, pour satisfaire les conditions préalables à l'exécution réussie d'une méthode. Ces opérations implicites sont la connexion, l'ouverture, la réouverture, la fermeture et la déconnexion. Vous pouvez contrôler le comportement de connexion et d'ouverture implicite à l'aide d'attributs de classe.

Connexion

Un objet de gestionnaire ImqQueueest connecté automatiquement pour toute méthode qui génère un appel à l'interface MQI (voir Références croisées C++ et MQI).

Ouverture

Un objet ImqObject est ouvert automatiquement pour toute méthode qui génère un appel MQGET, MQINQ, MQPUT ou MQSET. Utilisez la méthode **openFor** pour spécifier une ou plusieurs valeurs d' **option d'ouverture** pertinentes.

Ouvrir à nouveau

Un objet ImqObject est rouvert automatiquement pour toute méthode qui génère un appel MQGET, MQINQ, MQPUT ou MQSET, où l'objet est déjà ouvert, mais les **options d'ouverture** existantes ne permettent pas l'aboutissement de l'appel MQI. L'objet est temporairement fermé à l'aide de la valeur temporaire **close options** MQCO_NONE. Utilisez la méthode **openFor** pour ajouter une**option d'ouverture**.

La réouverture peut entraîner des problèmes dans des circonstances spécifiques:

- Une file d'attente dynamique temporaire est détruite lorsqu'elle est fermée et ne peut jamais être rouverte.
- Une file d'attente ouverte pour une entrée exclusive (explicitement ou par défaut) peut être accessible par d'autres dans la fenêtre de l'opportunité lors de la fermeture et de la réouverture.
- Une position de curseur de navigation est perdue lorsqu'une file d'attente est fermée. Cette situation n'empêche pas la fermeture et la réouverture, mais empêche l'utilisation ultérieure du curseur jusqu'à ce que MQGMO_BROWSE_FIRST soit à nouveau utilisé.
- Le contexte du dernier message extrait est perdu lorsqu'une file d'attente est fermée.

Si l'une de ces circonstances se produit ou peut être prévue, évitez les réouvertures en définissant explicitement des **options d'ouverture** appropriées avant qu'un objet ne soit ouvert (explicitement ou implicitement).

La définition explicite des **options d'ouverture** pour les situations de gestion de file d'attente complexes permet d'obtenir de meilleures performances et d'éviter les problèmes liés à l'utilisation de la réouverture.

Fermer

Un objet ImqObject est fermé automatiquement à tout moment où l'état de l'objet n'est plus viable, par exemple si une référence de connexion ImqObject est coupée ou si un objet ImqObject est détruit.

Déconnecter

Un gestionnaire ImqQueueest automatiquement déconnecté à tout moment où la connexion n'est plus viable, par exemple si une référence de connexion ImqObject est coupée ou si un objet gestionnaire ImqQueueest détruit.

Chaînes binaires et de caractères en C++

La classe ImqString encapsule le format de données *char ** traditionnel. La classe ImqBinary encapsule le tableau d'octets binaire. Certaines méthodes qui définissent des données de type caractères peuvent tronquer les données.

Méthodes qui définissent le caractère (**char ***) Les données prennent toujours une copie des données, mais certaines méthodes peuvent tronquer la copie, car certaines limites sont imposées par IBM MQ.

La classe ImqString (voir Classe C++ImqString) encapsule le **char *** traditionnel et prend en charge:

- Comparaison
- Concaténation
- Copie en cours
- Conversion d'entier en texte et conversion de texte en entier
- Extraction de jeton (mot)
- Conversion en majuscules

La classe ImqBinary (voir Classe C++ImqBinary) encapsule des tableaux d'octets binaires de taille arbitraire. En particulier, il est utilisé pour contenir les attributs suivants:

- **jeton de comptabilité** (MQBYTE32)
- **balise de connexion** (MQBYTE128)
- **ID corrélation** (MQBYTE24)
- **jeton de fonction** (MQBYTE8)
- **ID groupe** (MQBYTE24)
- **ID instance** (MQBYTE24)
- **ID message** (MQBYTE24)
- **jeton de message** (MQBYTE16)
- **ID instance de transaction** (MQBYTE16)

Où ces attributs appartiennent à des objets des classes suivantes:

- ImqCICSBridgeHeader (voir ImqCICSBridgeHeader C++ class)
- ImqGetMessageOptions (voir ImqGetMessageOptions C++ class)
- ImqIMSBridgeHeader (voir ImqIMSBridgeHeader C++ class)
- ImqMessageTracker (voir ImqMessageTracker C++ class)
- ImqQueueManager (voir ImqQueueManager C++ class)
- ImqReferenceEn-tête (voir ImqReferenceHeader C++ class)
- ImqWorkHeader (voir ImqWorkHeader C++ class)

La classe ImqBinary prend également en charge la comparaison et la copie.

Fonctions non prises en charge dans C++

Les classes et méthodes C++ IBM MQ sont indépendantes de la plateforme IBM MQ . Ils peuvent donc offrir des fonctions qui ne sont pas prises en charge sur certaines plateformes.

Si vous essayez d'utiliser une fonction sur une plateforme sur laquelle elle n'est pas prise en charge, la fonction est détectée par IBM MQ mais pas par les liaisons de langage C + +. IBM MQ signale l'erreur à votre programme, comme toute autre erreur MQI.

Messagerie en C++

Cette collection de rubriques explique comment préparer, lire et écrire des messages en C + +.

Préparation des données de message en C++

Les données de message sont préparées dans une mémoire tampon, qui peut être fournie par le système ou l'application. Il y a des avantages à l'une ou l'autre des méthodes. Des exemples d'utilisation d'une mémoire tampon sont donnés.

Lorsque vous envoyez un message, les données de message sont d'abord préparées dans une mémoire tampon gérée par un objet ImqCache (voir Classe C++ImqCache). Une mémoire tampon est associée (par héritage) à chaque objet ImqMessage (voir ImqMessage classe C++): elle peut être fournie par l'application (à l'aide de la méthode **useEmptyBuffer** ou **useFullBuffer**) ou automatiquement par le système. L'avantage de l'application fournissant la mémoire tampon de messages est qu'aucune copie de données n'est nécessaire dans de nombreux cas car l'application peut utiliser directement des zones de données préparées. L'inconvénient est que le tampon fourni est de longueur fixe.

La mémoire tampon peut être réutilisée et le nombre d'octets transmis peut varier à chaque fois, à l'aide de la méthode **setMessageLength** avant la transmission.

Lorsqu'elles sont fournies automatiquement par le système, le nombre d'octets disponibles est géré par le système et les données peuvent être copiées dans la mémoire tampon du message à l'aide, par exemple, de la méthode ImqCache **write** ou de la méthode ImqMessage **writeItem** . La mémoire tampon des messages augmente en fonction des besoins. Au fur et à mesure de la croissance de la mémoire tampon, il n'y a pas de perte de données précédemment écrites. Un message volumineux ou à plusieurs parties peut être écrit en éléments séquentiels.

Les exemples suivants illustrent des envois de messages simplifiés.

1. Utiliser les données préparées dans une mémoire tampon fournie par l'utilisateur

```
char szBuffer[ ] = "Hello world" ;

msg.useFullBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
```

2. Utiliser les données préparées dans une mémoire tampon fournie par l'utilisateur, où la taille de la mémoire tampon dépasse celle des données

```
char szBuffer[ 24 ] = "Hello world" ;

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.setMessageLength( 12 );
```

3. Copie de données dans une mémoire tampon fournie par l'utilisateur

```
char szBuffer[ 12 ];

msg.useEmptyBuffer( szBuffer, sizeof( szBuffer ) );
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

4. Copie de données dans une mémoire tampon fournie par le système

```
msg.setFormat( MQFMT_STRING );
msg.write( 12, "Hello world" );
```

5. Copier des données dans une mémoire tampon fournie par le système à l'aide d'objets (les objets définissent le format de message ainsi que le contenu)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Lecture des messages en C++

Une mémoire tampon peut être fournie par l'application ou le système. Les données sont accessibles directement à partir de la mémoire tampon ou sont lues séquentiellement. Il existe une classe équivalente à chaque type de message. Un exemple de code est fourni.

Lors de la réception de données, l'application ou le système peut fournir une mémoire tampon de message appropriée. La même mémoire tampon peut être utilisée pour plusieurs transmissions et plusieurs réceptions pour un objet ImqMessage particulier. Si la mémoire tampon de messages est fournie automatiquement, elle augmente pour tenir compte de la longueur des données reçues. Toutefois, il se peut qu'une mémoire tampon de messages fournie par l'application ne soit pas assez grande pour contenir les données reçues. Ensuite, une troncature ou un échec peut se produire, en fonction des options utilisées pour la réception des messages.

Les données entrantes sont accessibles directement à partir de la mémoire tampon de messages, auquel cas la longueur des données indique la quantité totale de données entrantes. Les données entrantes peuvent également être lues séquentiellement à partir de la mémoire tampon de messages. Dans ce cas, le pointeur de données adresse le prochain octet de données entrantes, et le pointeur de données et la longueur de données sont mis à jour à chaque lecture de données.

Les *éléments* sont des éléments d'un message, tous dans la zone utilisateur de la mémoire tampon de message, qui doivent être traités séquentiellement et séparément. Outre les données utilisateur standard, un élément peut être un en-tête de rebut ou un message de déclenchement. Les éléments sont toujours associés à des formats de message ; les formats de message ne sont **pas** toujours associés à des éléments.

Il existe une classe d'objet pour chaque élément qui correspond à un format de message IBM MQ reconnaissable. Il y en a un pour un en-tête de rebut et un pour un message de déclenchement. Il n'existe pas de classe d'objet pour les données utilisateur. C'est-à-dire qu'une fois les formats reconnaissables épuisés, le traitement du reste est laissé au programme d'application. Les classes des données utilisateur peuvent être écrites en spécialisant la classe ImqItem .

L'exemple suivant montre une réception de message qui prend en compte un certain nombre d'éléments potentiels pouvant précéder les données utilisateur, dans une situation imaginaire. Les données utilisateur non liées à un élément sont définies comme tout ce qui se produit après les éléments qui peuvent être identifiés. Une mémoire tampon automatique (valeur par défaut) est utilisée pour stocker une quantité arbitraire de données de message.

```
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.                      */
      ...

    }

  }
}
```

Dans cet exemple, FMT_USERCLASS est une constante représentant le nom de format à 8 caractères associé à un objet de la classe `UserClass`et est définie par l'application.

`UserClass` est dérivé de la classe ImqItem (voir ImqItem classe C++) et implémente les méthodes virtuelles **copyOut** et **pasteIn** de cette classe.

Les deux exemples suivants illustrent le code de la classe ImqDeadLetterHeader (voir ImqDeadLetterHeader C++ class). Le premier exemple illustre un message encapsulé personnalisé- *écriture de code* .

```
// 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 ;
}
```

Le deuxième exemple illustre un message encapsulé personnalisé- *lecture* du code.

```
// 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 ;
}
```

Avec une mémoire tampon automatique, la mémoire tampon est *volatile*. C'est-à-dire que les données de la mémoire tampon peuvent être conservées à un emplacement physique différent après chaque appel de méthode **get** . Par conséquent, chaque fois que des données de mémoire tampon sont référencées, utilisez les méthodes **bufferPointer** ou **dataPointer** pour accéder aux données de message.

Vous pouvez souhaiter qu'un programme réserve une zone fixe pour la réception des données de message. Dans ce cas, appelez la méthode **useEmptyBuffer** avant d'utiliser la méthode **get** .

L'utilisation d'une zone fixe non automatique limite les messages à une taille maximale. Il est donc important de prendre en compte l'option MQGMO_ACCEPT_TRUNCATED_MSG de l'objet ImqGetMessageOptions . Si cette option n'est pas spécifiée (valeur par défaut), le code anomalie MQRC_TRUNCATED_MSG_FAILED peut être attendu. Si cette option est spécifiée, le code anomalie MQRC_TRUNCATED_MSG_ACCEPTED peut être attendu en fonction de la conception de l'application.

L'exemple suivant montre comment une zone de stockage fixe peut être utilisée pour recevoir des messages:

```
char * pszBuffer = new char[ 100 ];

msg.useEmptyBuffer( pszBuffer, 100 );
gmo.setOptions( MQGMO_ACCEPT_TRUNCATED_MSG );
queue.get( msg, gmo );

delete [ ] pszBuffer ;
```

Dans ce fragment de code, la mémoire tampon peut toujours être adressée directement, avec *pszBuffer*, au lieu d'utiliser la méthode **bufferPointer** . Toutefois, il est préférable d'utiliser la méthode **dataPointer** pour l'accès général. L'application (et non l'objet de classe ImqCache) doit supprimer une mémoire tampon définie par l'utilisateur (non automatique).

Attention: la spécification d'un pointeur null et d'une longueur de zéro avec **useEmptyBuffer** ne désigne pas une mémoire tampon de longueur fixe de longueur zéro comme prévu. Cette combinaison est interprétée comme une demande visant à ignorer toute mémoire tampon précédemment définie par l'utilisateur et à revenir à l'utilisation d'une mémoire tampon automatique.

Ecriture d'un message dans la file d'attente de rebut en C++

Exemple de code de programme pour l'écriture d'un message dans la file d'attente de rebut.

Un cas typique d'un message à plusieurs parties est un message contenant un en-tête de lettre morte. Les données d'un message qui ne peuvent pas être traitées sont ajoutées à l'en-tête de rebut.

```
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 );
```

Ecriture d'un message dans le pont IMS en C++

Exemple de code de programme pour l'écriture d'un message sur le pont IMS .

Les messages envoyés au pont IBM MQ - IMS peuvent utiliser un en-tête spécial. L'en-tête de pont IMS est préfixé aux données de message standard.

```
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 );
```

Ecriture d'un message dans CICS bridge en C++

Exemple de code de programme pour l'écriture d'un message dans CICS bridge.

Les messages envoyés à IBM MQ for z/OS à l'aide de CICS bridge nécessitent un en-tête spécial. L'en-tête CICS bridge est préfixé aux données de message standard.

```
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 );
```

Ecriture d'un message avec un en-tête de travail en C++

Exemple de code de programme pour l'écriture d'un message destiné à une file d'attente gérée par z/OS Workload Manager.

Les messages envoyés à IBM MQ for z/OS, qui sont destinés à une file d'attente gérée par z/OS Workload Manager, nécessitent un en-tête spécial. L'en-tête de travail est préfixé aux données de message standard.

```
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 );
```

Génération de programmes C++ IBM MQ

L'URL des compilateurs pris en charge est répertoriée, ainsi que les commandes à utiliser pour compiler, lier et exécuter des programmes C++ et des exemples sur les plateformes IBM MQ .

Pour la liste des compilateurs pour chaque plateforme et version prises en charge de IBM MQ, voir Configuration système requise pour IBM MQ.

La commande dont vous avez besoin pour compiler et lier votre programme C++ IBM MQ dépend de votre installation et de vos exigences. Les exemples suivants illustrent des commandes de compilation et de liaison classiques pour certains compilateurs utilisant l'installation par défaut de IBM MQ sur un certain nombre de plateformes.

AIX Génération de programmes C++ sur AIX

Générez des programmes IBM MQ C++ sous AIX à l'aide du compilateur XL C Enterprise Edition .

V 9.4.0 Pour plus d'informations sur le mappage des différentes options de compilation entre les compilateurs XLC 16 et XLC 17, voir Mappage des options.

Deprecated V 9.4.0 La prise en charge du compilateur XL C/C++ for AIX 16 sous AIX est obsolète à partir de IBM MQ 9.4.0.

Environnement

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Application 32 bits sans unités d'exécution

```
xlC -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Application à unités d'exécution 32 bits

```
xlC_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Application 64 bits sans unités d'exécution

```
xlC -q64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Application à unités d'exécution 64 bits

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

V 9.4.0 **Application non à unités d'exécution 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 **Application à unités d'exécution 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ca_r -limqb23ca_r -lmqic_r
```

V 9.4.0 **Application 64 bits sans unités d'exécution (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca -limqb23ca -lmqic
```

V 9.4.0 **Application à unités d'exécution 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ca_r -limqb23ca_r -lmqic_r
```

serveur

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Application 32 bits sans unités d'exécution

```
xlC -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Application à unités d'exécution 32 bits

```
xlC_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Application 64 bits sans unités d'exécution

```
xlC -q64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Application à unités d'exécution 64 bits

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

V 9.4.0 **Application non à unités d'exécution 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 **Application à unités d'exécution 32 bits (XLC 17)**

```
ibm-clang++_r -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ca_r -limqb23ca_r -lmqm_r
```

V 9.4.0 **Application 64 bits sans unités d'exécution (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca -limqb23ca -lmqm
```

V 9.4.0 **Application à unités d'exécution 64 bits (XLC 17)**

```
ibm-clang++_r -m64 -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ca_r -limqb23ca_r -lmqm_r
```

IBM i Génération de programmes C++ sur IBM i

Générez des programmes IBM MQ C++ sous IBM i à l'aide du compilateur ILE C + +.

IBM ILE C++ for IBM i est un compilateur natif pour les programmes C + +. Les instructions suivantes décrivent comment utiliser ce compilateur pour créer des applications IBM MQ C++ à l'aide de *Hello World !* Exemple de programme IBM MQ .

1. Installez le compilateur ILE C++ for IBM i comme indiqué dans le manuel *Read Me first !* qui accompagne le produit.
2. Vérifiez que la bibliothèque QCXXN figure dans votre liste de bibliothèques.
3. Créez l'exemple de programme HELLO WORLD:
   a. Créez un module:

   ```
   CRTCPPMOD MODULE(MYLIB/IMQWRLD) +
   SRCSTMF('/QIBM/ProdData/mqm/samp/imqwrld.cpp') +
   INCDIR('/QIBM/ProdData/mqm/inc') DFTCHAR(*SIGNED) +
   TERASPACE(*YES)
   ```

   La source des exemples de programme C++ se trouve dans `/QIBM/ProdData/mqm/samp` et les fichiers d'inclusion dans `/QIBM/ProdData/mqm/inc`.

   Vous pouvez également trouver la source dans la bibliothèque `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

b. Associez cet élément à des programmes de service fournis par IBM MQpour générer un objet programme:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

Pour générer une application à unités d'exécution, utilisez les programmes de service réentrant:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

c. Exécutez l'exemple de programme HELLO WORLD à l'aide de SYSTEM.DEFAULT.LOCAL.QUEUE:

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux Génération de programmes C++ sur Linux

Générez des programmes IBM MQ C++ sous Linux à l'aide du compilateur GNU g + +.

System p

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Client: System p

Application 32 bits sans unités d'exécution

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Serveur : System p

Application 32 bits sans unités d'exécution

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

IBM Z

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Client: IBM Z

Application 32 bits sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Serveur : IBM Z

Application 32 bits sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

x86-64 (32 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Client: x86-64 (32 bits)

Application 32 bits sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L
MQ_INSTALLATION_PATH/lib -Wl,
-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Serveur: x86-64 (32 bits)

Application 32 bits sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -L
MQ_INSTALLATION_PATH/lib64
-Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Windows Génération de programmes C++ sur Windows

Générez des programmes IBM MQ C++ sous Windows à l'aide du compilateur Microsoft Visual Studio C ++.

Avertissement : Les bibliothèques fournies par IBM MQ sont des bibliothèques dynamiques et non des bibliothèques statiques. IBM MQ fournit un élément appelé "`import libraries`" que vous pouvez utiliser uniquement lors de la compilation. Pour l'exécution, vous devez utiliser les bibliothèques dynamiques.

Depuis la IBM MQ 8.0.0 Fix Pack 4, IBM MQ fournit des clients redistribuables contenant les bibliothèques requises pour l'exécution des applications IBM MQ . Ces bibliothèques peuvent être conditionnées et redistribuées avec des applications client. Pour plus d'informations, voir Clients redistribuables sur Windows.

Les fichiers de bibliothèque (.lib) et les fichiers dll à utiliser avec les applications 32 bits sont installés dans *MQ_INSTALLATION_PATH*`/Tools/Lib`. Les fichiers à utiliser avec les applications 64 bits sont installés dans *MQ_INSTALLATION_PATH*`/Tools/Lib64`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Environnement

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

serveur

```
cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Installation de l'environnement d'exécution C universel

Si vous utilisez Windows 8.1 ou Windows Server 2012 R2, vous devez installer la mise à jour d'exécution C universelle (Universal CRT) à partir de Microsoft. Cet environnement d'exécution est inclus dans Windows 10 et Windows Server 2016.

La mise à jour d'Universal CRT est Microsoft mise à jour KB3118401. Vous pouvez vérifier si vous disposez de cette mise à jour en recherchant un fichier appelé `ucrtbase.dll` dans votre répertoire

C:\Windows\System32 . Si ce n'est pas le cas, vous pouvez télécharger la mise à jour à partir de la page Microsoft suivante: https://www.catalog.update.microsoft.com/Search.aspx?q=kb3118401.

La tentative d'exécution d'un programme IBM MQ ou d'un programme que vous compilez vous-même à l'aide de Microsoft Visual Studio 2017, sans l'environnement d'exécution installé, génère des erreurs telles que les suivantes:

```
The program can't start because api-ms-win-crt-runtime-|1-1-0.dll
is missing from your computer. Try reinstalling the program to
fix this problem.
```

Fourniture de contextes d'exécution pour les programmes Microsoft Visual Studio 2012

Si vous avez compilé un programme IBM MQ à l'aide de Microsoft Visual Studio 2012, sachez que le programme d'installation de IBM MQ n'installe pas les environnements d'exécution Microsoft Visual Studio 2012 C/C + +. Si votre version précédente de IBM MQ a été installée sur le même ordinateur, les environnements d'exécution Microsoft Visual Studio 2012 sont disponibles à partir de cette installation.

Toutefois, si vous utilisez un programme qui a été généré à l'aide de Microsoft Visual Studio 2012 et qu'aucune version précédente de IBM MQ n'a été installée, vous devez effectuer l'une des opérations suivantes:

- Téléchargez et installez **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** à partir de Microsoft.
- Recompilez votre programme avec Microsoft Visual Studio 2017ou un autre niveau Microsoft Visual Studio pour lequel les environnements d'exécution sont installés.

Bibliothèques client C++ générées avec le compilateur Microsoft Visual Studio 2015

IBM MQ fournit des bibliothèques client C++ qui sont générées avec le compilateur C++ Microsoft Visual Studio 2015 et le compilateur C++ Microsoft Visual Studio 2017 .

Les versions 32 bits et 64 bits des bibliothèques C++ IBM MQ sont fournies. Les bibliothèques 32 bits sont installées dans le dossier bin\vs2015 et les bibliothèques 64 bits sont installées dans les dossiers bin64\vs2015 .

Par défaut, IBM MQ est configuré pour utiliser les bibliothèques Microsoft Visual Studio 2017 . Pour utiliser les bibliothèques Microsoft Visual Studio 2015 , vous devez définir la variable d'environnement MQ_PREFIX_VS_LIBRARIES sur MQ_PREFIX_VS_LIBRARIES=vs2015 avant d'installer IBM MQou avant d'utiliser la commande **setmqenv** ou **setmqinst** .

Utilisation de bibliothèques C++ IBM MQ nommées différemment

IBM MQ fournit des bibliothèques client C++ supplémentaires qui sont nommées différemment. Ces bibliothèques sont générées avec les compilateurs Microsoft Visual Studio 2015 et Microsoft Visual Studio 2017 C + +. Ces bibliothèques sont fournies en plus des bibliothèques C++ existantes qui sont également générées avec le compilateur C++ Microsoft Visual Studio 2017 . Etant donné que ces bibliothèques IBM MQ C++ supplémentaires ont des noms différents, vous pouvez exécuter des applications IBM MQ C++ qui sont générées à l'aide de IBM MQ C++ et compilées avec Microsoft Visual Studio 2017 et des versions antérieures du produit sur le même ordinateur.

Les bibliothèques Microsoft Visual Studio 2017 supplémentaires portent les noms suivants:

- imqb23vnvs2017.dll
- imqc23vnvs2017.dll
- imqs23vnvs2017.dll
- imqx23vnvs2017.dll

Les bibliothèques Microsoft Visual Studio 2015 supplémentaires portent les noms suivants:

- `imqb23vnvs2015.dll`
- `imqc23vnvs2015.dll`
- `imqs23vnvs2015.dll`
- `imqx23vnvs2015.dll`

Les versions 32 bits et 64 bits de ces bibliothèques sont fournies. Les bibliothèques 32 bits sont installées dans le dossier `bin` et les bibliothèques 64 bits sont installées dans le dossier `bin64` . Les bibliothèques d'importation correspondantes sont installées dans les répertoires `Tools\lib` et `Tools\lib64` .

Si votre application utilise des fichiers `imq*vs2015.lib` , vous devez la compiler à l'aide du compilateur Microsoft Visual Studio 2015 . Pour exécuter des applications IBM MQ C++ qui sont compilées avec Microsoft Visual Studio 2015ou des applications qui sont compilées avec une version antérieure du produit sur le même ordinateur, la variable d'environnement PATH doit être préfixée comme illustré dans les exemples suivants:

- Pour les applications 32 bits:

```
SET PATH=installation folder\bin\vs2015;%PATH%
```

- Pour les applications 64 bits :

```
SET PATH=installation folder\bin64\vs2015;%PATH%
```

Concepts associés

Windows : modifications apportées depuis IBM MQ 8.0

z/OS Building C++ programs on z/OS Batch, RRS Batch and CICS

Build IBM MQ C++ programs on z/OS for the Batch, RRS batch or CICS environments and run the sample programs.

You can write C++ programs for three of the environments that IBM MQ for z/OS supports:

- Batch
- RRS batch
- CICS

Compile, prelink and link

Create an z/OS application by compiling, pre-linking, and link-editing your C++ source code.

IBM MQ C++ for z/OS is implemented as z/OS DLLs for the IBM C++ for z/OS language. Using DLLs, you concatenate the supplied definition sidedecks with the compiler output at pre-link time. This allows the linker to check your calls to the IBM MQ C++ member functions.

Note: There are three sets of sidedecks for each of the three environments.

To build an IBM MQ for z/OS C++ application, create and run JCL. Use the following procedure:

1. If your application runs under CICS, use the CICS-supplied procedure to translate CICS commands in your program.

   In addition, for CICS applications you need to:

   a. Add the SCSQLOAD library to the DFHRPL concatenation.

   b. Define the CSQCAT1 CEDA group using the member IMQ4B100 in the SCSQPROC library.

   c. Install CSQCAT1.

2. Compile the program to produce object code. The JCL for your compilation must include statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Be sure to specify the `/cxx` compiler option.

   Note: The name **thlqual** is the high level qualifier of the IBM MQ installation library on z/OS.
3. Pre-link the object code created in step "2" on page 566, including the following definition sidedecks, which are supplied in **thlqual**.SCSQDEFS:
   a. imqs23dm and imqb23dm for batch
   b. imqs23dr and imqb23dr for RRS batch
   c. imqs23dc and imqb23dc for CICS

   These are the corresponding DLLs.
   a. imqs23im and imqb23im for batch
   b. imqs23ir and imqb23ir for RRS batch
   c. imqs23ic and imqb23ic for CICS
4. Link-edit the object code created in step "3" on page 566, to produce a load module, and store it in your application load library.

To run batch or RRS batch programs, include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a CICS program, first get your system administrator to define it to CICS as an IBM MQ program and transaction. You can then run it in the usual way.

Run the sample programs

The programs are described in "Exemples de programmes C++" on page 542.

The sample applications are supplied in source form only. The files are:

Table 75. z/OS sample program files

Sample	Source program (in library thlqual.SCSQCPPS)	JCL (in library thlqual.SCSQPROC)
HELLO WORLD	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

To run the samples, compile and link-edit them as with any C++ program (see "Building C++ programs on z/OS Batch, RRS Batch and CICS" on page 565). Use the supplied JCL to construct and run a batch job. You must initially customize the JCL, by following the commentary included with it.

z/OS Building C++ programs on z/OS UNIX System Services

Build IBM MQ C++ programs on z/OS UNIX System Services (z/OS UNIX).

To build an application under the z/OS UNIX shell, you must give the compiler access to the IBM MQ include files (located in `thlqual.SCSQC370` and `hlqual.SCSQHPPS`), and link against two of the DLL sidedecks (located in `thlqual.SCSQDEFS`). At runtime, the application needs access to the IBM MQ data sets `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`, and one of the language specific data sets, such as `thlqual.SCSQANLE` [6].

Compiling

1. Copy the sample into the file system using the TSO **oput** command, or use FTP. The rest of this example assumes that you have copied the sample into a directory called `/u/fred/sample`, and named it `imqwrld.cpp`.
2. Log into the z/OS UNIX shell, and change to the directory where you placed the sample.
3. Set up the C++ compiler so that it can accept the DLL sidedeck and .cpp files as input:

```
/u/fred/sample:> export _CXX_EXTRA_ARGS=1
/u/fred/sample:> export _CXX_CXXSUFFIX="cpp"
```

4. Compile and link the sample program. The following command links the program with the batch sidedecks; the RRS batch sidedecks can be used instead. The \ character is used to split the command over more than one line. Do not enter this character; enter the command as a single line:

```
/u/fred/sample:> c++ -o imqwrld -I "//'thlqual.SCSQC370'" \
-I "//'thlqual.SCSQHPPS'" imqwrld.cpp \
"//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"
```

For more information on the TSO **oput** command, refer to the z/OS UNIX Command Reference.

You can also use the make utility to simplify building C++ programs. Here is a sample makefile to build the HELLO WORLD C++ sample program. It separates the compiling and linking stages. Set up the environment as in step “3” on page 567 before running make.

```
flags = -I "//'thlqual.SCSQC370'" -I "//'thlqual.SCSQHPPS'"
decks = "//'thlqual.SCSQDEFS(IMQS23DM)'" "//'thlqual.SCSQDEFS(IMQB23DM)'"

imqwrld: imqwrld.o
   c++ -o imqwrld imqwrld.o $(decks)

imqwrld.o: imqwrld.cpp
   c++ -c -o imqwrld $(flags) imqwrld.cpp
```

Refer to z/OS UNIX System Services Programming Tools for more information on using make.

Running

1. Log into the z/OS UNIX shell, and change to the directory where you built the sample.
2. Set up the STEPLIB environment variable to include the IBM MQ data sets:

```
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQLOAD
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQAUTH
/u/fred/sample:> export STEPLIB=$STEPLIB:thlqual.SCSQANLE
```

3. Run the sample:

```
/u/fred/sample:> ./imqwrld
```

Développement d'applications .NET

Les applications IBM MQ classes for .NET permettent aux applications .NET de se connecter à IBM MQ en tant que IBM MQ MQI client ou de se connecter directement à un serveur IBM MQ .

[6] You can link with any of the sidedecks listed in "Pre-link the object code to run your z/OS UNIX in any of the three environments, “Building C++ programs on z/OS Batch, RRS Batch and CICS” on page 565

Avant de commencer

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client IBM MQ .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Pourquoi et quand exécuter cette tâche

IBM MQ classes for .NET est un ensemble de classes qui permettent aux applications .NET d'interagir avec IBM MQ. Ils représentent les différents composants d' IBM MQ utilisés par votre application, tels que les gestionnaires de files d'attente, les files d'attente, les canaux et les messages. Pour plus d'informations sur ces classes, voir Classes et interfaces IBM MQ .NET.

V 9.4.0 IBM MQ 9.4.0 fournit une bibliothèque client IBM MQ .NET générée avec .NET 6 comme infrastructure cible. Pour plus d'informations, voir «Installation de IBM MQ classes for .NET», à la page 569.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for .NET. Pour plus d'informations, voir «Installation de IBM MQ classes for .NET», à la page 569.

Si vous disposez d'applications qui utilisent Microsoft .NET Framework et que vous souhaitez tirer parti des fonctions de IBM MQ, vous devez utiliser IBM MQ classes for .NET Framework. Pour plus d'informations, voir «Installation de IBM MQ classes for .NET Framework», à la page 575.

Pour plus d'informations sur les différences entre IBM MQ classes for .NET Framework et IBM MQ classes for .NET, voir «Installation de IBM MQ classes for .NET», à la page 569.

Les applications gérées par IBM MQ .NET peuvent équilibrer automatiquement les connexions entre les gestionnaires de files d'attente en cluster. Les bibliothèques IBM MQ classes for .NET et IBM MQ classes for .NET Framework sont prises en charge. Pour plus d'informations, voir A propos des clusters uniformes et Equilibrage automatique des applications.

L'interface IBM MQ .NET orientée objet est différente de l'interface MQI en ce qu'elle utilise des méthodes d'objets plutôt que les instructions MQI. L'interface de programme d'application IBM MQ procédurale est construite autour d'instructions telles que celles de la liste suivante:

```
MQCONN, MQDISC, MQOPEN, MQCLOSE,
MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Ces instructions prennent toutes, en tant que paramètre, un descripteur de l'objet IBM MQ sur lequel elles doivent fonctionner. Etant donné que .NET est orienté objet, l'interface de programmation .NET effectue cette rotation. Votre programme se compose d'un ensemble d'objets IBM MQ sur lesquels vous agissez en appelant des méthodes sur ces objets. Vous pouvez écrire des programmes dans n'importe quel langage pris en charge par .NET.

Lorsque vous utilisez l'interface de procédure, vous vous déconnectez d'un gestionnaire de files d'attente à l'aide de l'appel MQDISC (*Hconn*, CompCode, Reason), où *Hconn* est un descripteur du gestionnaire de files d'attente. Dans l'interface .NET , le gestionnaire de files d'attente est représenté par un objet de classe MQQueueManager. Vous vous déconnectez du gestionnaire de files d'attente en appelant la méthode Disconnect () sur cette classe.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
```

```
// disconnect from the queue manager
queueManager.Disconnect();
```

Concepts associés

Présentation technique

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

Tâches associées

Contacter le support IBM

Identification et résolution des problèmes liés à IBM MQ.NET

«Développement d'applications Microsoft Windows Communication Foundation avec IBM MQ», à la page 1299
Le canal personnalisé Microsoft Windows Communication Foundation (WCF) pour IBM MQ envoie et reçoit des messages entre les clients et les services WCF.

«Développement d'applications XMS .NET», à la page 629
IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fournit une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que Java Message Service (JMS) API. IBM MQ Message Service Client (XMS) for .NET comprend une implémentation complète de XMS, qui peut être utilisée par tout langage compatible avec .NET.

Windows Linux Installation de IBM MQ classes for .NET

IBM MQ classes for .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux

Prérequis et installation

V 9.4.0 IBM MQ 9.4.0 fournit une bibliothèque client IBM MQ .NET générée avec .NET 6 comme infrastructure cible. Depuis la IBM MQ 9.4.0, Microsoft .NET 6.0 est la version minimale requise pour l'exécution d'applications à l'aide de bibliothèques IBM MQ générées à l'aide de .NET 6 comme infrastructure cible. La bibliothèque client IBM MQ .NET générée à l'aide de .NET 6 comme infrastructure cible est disponible sous *MQ_INSTALLATION_PATH*`/bin` sous Windows et sous *MQ_INSTALLATION_PATH*`/lib64` sous Linux.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for .NET. Si vous utilisez une application .NET 6 , vous pouvez exécuter cette application sans qu'aucune recompilation ne soit nécessaire en effectuant une petite modification dans le fichier `runtimeconfig` pour définir `targetframeworkversion` sur `"net8.0"`.

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client IBM MQ .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

La version la plus récente de IBM MQ classes for .NET est installée par défaut dans le cadre de l'installation standard de IBM MQ dans la fonction *Java and .NET Messaging and Web Services* .

Windows Pour plus d'informations sur les prérequis et l'installation sous Windows:

- Voir Conditions requises pour IBM MQ classes for .NET pour les logiciels prérequis pour l'exécution de IBM MQ classes for .NET.
- Pour obtenir des instructions d'installation, voir Installation d'un serveur IBM MQ sur Windows ou Installation d'un client IBM MQ sur des systèmes Windows .

Linux Pour plus d'informations sur les prérequis et l'installation sous Linux:

- Pour connaître les logiciels prérequis pour l'exécution de IBM MQ classes for .NET, voir Conditions requises pour IBM MQ classes for .NET .
- Pour obtenir des instructions d'installation rpm, voir Installation d'un client IBM MQ sur des systèmes Linux.
- Pour Linux Ubuntu, à l'aide de packages Debian , voir Installation d'un client IBM MQ sur des systèmes Linux.

La bibliothèque IBM MQ classes for .NET Standard , `amqmdnetstd.dll`, peut être téléchargée à partir du référentiel NuGet . Pour plus d'informations, voir «Téléchargement de IBM MQ classes for .NET à partir du référentiel NuGet», à la page 574.

Bibliothèque `amqmdnetstd.dll`

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, la bibliothèque `amqmdnetstd.dll` générée à l'aide de .NET 6 comme infrastructure cible est disponible aux emplacements suivants:

- Windows Sous Windows: *MQ_INSTALLATION_PATH*`\bin`. Les exemples d'application sont installés dans *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux Sous Linux: *MQ_INSTALLATION_PATH*`\lib64`. Les exemples .NET se trouvent dans *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

Avertissement : V 9.4.0 V 9.4.0 Removed A partir de IBM MQ 9.4.0, les bibliothèques client IBM MQ .NET générées à l'aide de .NET Standard 2.0 comme infrastructure cible sont supprimées. Ces bibliothèques sont obsolètes à l'adresse IBM MQ 9.3.1.

Stabilized LTS La bibliothèque `amqmdnet.dll` pour .NET Framework est toujours fournie, mais cette bibliothèque est stabilisée, c'est-à-dire qu'aucune nouvelle fonction n'y sera introduite. Pour les fonctions les plus récentes, vous devez migrer vers la bibliothèque `amqmdnetstd.dll` . Toutefois, vous pouvez continuer à utiliser la bibliothèque `amqmdnet.dll` sur les éditions IBM MQ 9.1 ou ultérieures Long Term Support ou Continuous Delivery .

V 9.4.0 V 9.4.0 Voici deux scénarios que vous pouvez rencontrer après la suppression des bibliothèques `netstandard2.0` :

- Si vous utilisez une application IBM MQ classes for .NET Framework générée à l'aide des bibliothèques `netstandard2.0` telles que `amqmdnetstd.dll`, vous devez régénérer votre application avec les bibliothèques Microsoft.NET Framework 4.7.2 telles que `amqmdnet.dll`, afin que votre application s'exécute correctement. Si vous ne régénérez pas votre application, vous risquez d'obtenir un System.IO.Unexceptionable non exceptionnel:

```
Exception interceptée: System.IO.FileLoadException: Impossible de charger le fichier ou
l'assemblage'amqmdnetstd, Version=9.3.5.0, Culture=neutre, PublicKeyToken=23d6cb914eeaac0e'ou
l'une de ses dépendances. La définition de manifeste de l'assemblage localisé ne correspond pas
à la référence d'assemblage. (Exception de HRESULT: 0x80131040)
Nom de fichier:'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e'
   à SimplePut.SimplePut.PutMessages()
   sur SimplePut.SimplePut.Main (String [ ] args) dans C:\SampleCode\Program.cs:line 132
```

- Si vous utilisez une application .NET 6 générée à l'aide de bibliothèques `netstandard2.0` , il vous suffit de remplacer ces bibliothèques par les mêmes bibliothèques .NET 6 dans le dossier `bin` du répertoire d'exécution de l'application. Aucune régénération n'est requise.

  Remarque : Le niveau de la bibliothèque .NET 6 de remplacement doit toujours être identique ou supérieur à celui de la bibliothèque `netstandard2.0` remplacée.

commande dspmqver

Vous pouvez utiliser la commande **dspmqver** pour afficher les informations de version et de génération pour le composant .NET Core .

Comparaison des fonctions entre IBM MQ classes for .NET Framework et IBM MQ classes for .NET

Le tableau suivant répertorie les fonctions de IBM MQ classes for .NET Framework par rapport aux fonctions de IBM MQ classes for .NET

Tableau 76. Différences entre IBM MQ classes for .NET Framework et IBM MQ classes for .NET .

Fonction	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
Noms de classe (API)	Toutes les classes restent identiques sur chaque réseau.	Toutes les classes restent identiques sur chaque réseau.
Système d'exploitation	Windows	Windows Conteneurs dockérisés Linux macOS
Fichier `app.config` (fichier de configuration permettant d'activer la trace dans le client redistribuable)	Le fichier `app.config` est utilisé pour activer la trace pour le package redistribuable et le client IBM MQ .NET autonome. Pour plus d'informations sur les variables que vous utilisez pour la trace, notamment **MQTRACEPATH** et **MQTRACELEVEL**, voir Traçage d'un client IBM MQ classes for .NET Framework à l'aide d'un fichier de configuration d'application .	`app.config` n'est pas pris en charge. Utilisez des variables d'environnement.

Tableau 76. Différences entre IBM MQ classes for .NET Framework et IBM MQ classes for .NET . (suite)

Fonction	IBM MQ classes for .NET Framework	IBM MQ classes for .NET
Fonction de trace	Pour une installation client complète de IBM MQ, vous pouvez utiliser la commande **strmqtrc** afin d'activer la trace pour IBM MQ classes for .NET Framework. Pour les clients redistribuables, le fichier `app.config` est également utilisé pour activer la trace. Pour plus d'informations, voir Traçage des applications IBM MQ .NET. V 9.4.0 Depuis IBM MQ 9.4.0, vous pouvez activer et désactiver la trace à l'aide du fichier `mqclient.ini` et en définissant les propriétés appropriées de la section Trace. Vous pouvez également activer et désactiver la fonction de trace de manière dynamique avec le fichier `mqclient.ini` . Pour plus d'informations, voir Suivi des applications IBM MQ .NET avec mqclient.ini.	La variable d'environnement **MQDOTNET_TRACE_ON** permet d'activer la trace pour les clients redistribuables. Les valeurs inférieures ou égales à 0 n'activent pas la trace. La valeur 1 active le traçage de niveau par défaut. Une valeur supérieure à 1 active le traçage détaillé. La définition de cette variable d'environnement sur une autre valeur telle que la chaîne n'active pas la trace. Voir Traçage des applications IBM MQ .NET à l'aide de variables d'environnement. La variable d'environnement **MQDOTNET_TRACE_ON** vérifie si le répertoire de trace IBM MQ est disponible ou non. Si le répertoire de trace est disponible, le fichier de trace est généré dans le répertoire de trace. Toutefois, si IBM MQ n'est pas installé, le fichier de trace est copié dans le répertoire de travail en cours. D'autres variables d'environnement, telles que **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**, etc., utilisées pour IBM MQ classes for .NET Framework, peuvent être utilisées et fonctionner de la même manière. V 9.4.0 Depuis IBM MQ 9.4.0, vous pouvez activer et désactiver la trace à l'aide du fichier `mqclient.ini` et en définissant les propriétés appropriées de la section Trace. Vous pouvez également activer et désactiver la fonction de trace de manière dynamique avec le fichier `mqclient.ini` . Pour plus d'informations, voir Suivi des applications IBM MQ .NET avec mqclient.ini.
Modes de transport	Géré, non géré, liaisons	Gérée

Tableau 76. Différences entre IBM MQ classes for .NET Framework et IBM MQ classes for .NET . (suite)		
Fonction	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET**
TLS	Le magasin de clés Windows est utilisé pour stocker les certificats.	Windows Sous Windows, le magasin de clés doit être utilisé pour stocker les certificats. Les valeurs admises sont *USER et *SYSTEM. En fonction de l'entrée, le client IBM MQ .NET examine le magasin de clés Windows de l'utilisateur en cours ou de l'ensemble du système. Linux Sous Linux, il est recommandé d'utiliser la classe X509Store pour installer les certificats et .NET Core installe les certificats à l'emplacement suivant: "`.dotnet/corefx/cryptography/x509stores`".
Table de définition de canal du client (CCDT)	Pris en charge	Prise en charge. Les paramètres du chemin de la table de définition de canal du client (CCDT) sont les mêmes que pour les classes .NET Framework.
Reconnexion automatique du client	Pris en charge	Pris en charge
Transactions réparties	Pris en charge	Non pris en charge
Installation de bibliothèques de liaison dynamique (dll) dans le cache d'assemblage global (GAC)	Les bibliothèques de liaison dynamique sont installées dans le cache d'assemblage global (GAC) dans le cadre de l'installation d'IBM MQ.	Les bibliothèques de liaison dynamique ne sont pas installées dans le cache d'assemblage global (GAC) dans le cadre de l'installation d'IBM MQ.

Remarque : Windows Identificateurs de sécurité (SID) Windows :

L'authentification au niveau du domaine n'est pas prise en charge pour IBM MQ classes for .NET (bibliothèques.NET Standard et .NET 6). L'ID utilisateur connecté est utilisé pour l'authentification.

Développement d'applications IBM MQ .NET Core sous macOS

macOS

Les applications IBM MQ .NET Core peuvent être développées sur macOS.

Les bibliothèques IBM MQ .NET ne sont pas fournies avec le kit d'outils macOS . Vous devez donc les copier à partir d'un client Windows ou Linux IBM MQ sur macOS. Vous pouvez ensuite utiliser ces bibliothèques pour développer des applications IBM MQ .NET Core sur macOS.

Une fois développées, ces applications peuvent être exécutées et prises en charge dans des environnements Windows ou Linux.

Concepts associés

«Installation de IBM MQ classes for .NET Framework», à la page 575
IBM MQ classes for .NET Framework, y compris les exemples, sont installés avec IBM MQ. Il existe un prérequis pour Microsoft.NET Framework sous Windows.

«Installation de IBM MQ classes for XMS .NET», à la page 634
IBM MQ classes for XMS .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux.

Windows Linux Téléchargement de IBM MQ classes for .NET à partir du référentiel NuGet

Le fichier IBM MQ classes for .NET peut être téléchargé à partir du référentiel NuGet afin de pouvoir être facilement utilisé par les développeurs .NET .

Pourquoi et quand exécuter cette tâche

NuGet est le gestionnaire de package pour les plateformes de développement Microsoft , y compris .NET. Les outils client NuGet permettent de produire et de consommer des packages. Un package NuGet est un fichier compressé unique avec l'extension `.nupkg` qui contient du code compilé (DLL), d'autres fichiers associés à ce code et un manifeste descriptif qui inclut des informations telles que le numéro de version du package.

Vous pouvez télécharger le package `IBMMQDotnetClient` NuGet , qui contient la bibliothèque `amqmdnetstd.dll` , à partir de la galerie NuGet , qui est le référentiel de package central utilisé par tous les auteurs et destinataires de package.

Remarque : V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, le package NuGet contient des bibliothèques générées à l'aide de .NET 6 comme infrastructure cible.

Removed La bibliothèque client IBM MQ .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for .NET. Si vous utilisez une application .NET 6 , vous pouvez exécuter cette application sans qu'aucune recompilation ne soit nécessaire en effectuant une petite modification dans le fichier `runtimeconfig` pour définir `targetframeworkversion` sur `"net8.0"`.

Il existe trois façons de télécharger le package `IBMMQDotnetClient` :

- En utilisant Microsoft Visual Studio. NuGet est distribué en tant qu'extension Microsoft Visual Studio . Depuis Microsoft Visual Studio 2012, NuGet est préinstallé par défaut.
- A partir de la ligne de commande, à l'aide du gestionnaire de package NuGet ou de l'interface de ligne de commande .NET .
- En utilisant un navigateur Web.

Comme pour le package redistribuable, vous activez la trace à l'aide de la variable d'environnement **`MQDOTNET_TRACE_ON`**.

Procédure

- Pour télécharger le package `IBMMQDotnetClient` à l'aide de l'interface utilisateur de Package Manager dans Microsoft Visual Studio, procédez comme suit:
  a) Cliquez avec le bouton droit de la souris sur le projet .NET , puis cliquez sur **Manage Nuget Packages**.
  b) Cliquez sur l'onglet **Parcourir** et recherchez `"IBMMQDotnetClient"`.
  c) Sélectionnez le package et cliquez sur **Installer**.
  Lors de l'installation, le gestionnaire de packages fournit des informations de progression sous la forme d'instructions de console.
- Pour télécharger le package `IBMMQDotnetClient` à partir de la ligne de commande, choisissez l'une des options suivantes:

- A l'aide de NuGet Package Manager, entrez la commande suivante:

  ```
  Install-Package IBMMQDotnetClient -Version 9.1.4.0
  ```

  Lors de l'installation, le gestionnaire de packages fournit des informations de progression sous la forme d'instructions de console. Vous pouvez rediriger la sortie vers un fichier journal.
- A l'aide de l'interface de ligne de commande .NET , entrez la commande suivante:

  ```
  dotnet add package IBMMQDotnetClient --version 9.1.4
  ```

- A l'aide d'un navigateur Web, téléchargez le package `IBMMQDotnetClient` à partir de https://www.nuget.org/packages/IBMMQDotnetClient.

Concepts associés

Informations sur la licence de IBM MQ Client for .NET

Tâches associées

«Téléchargement d'IBM MQ classes for XMS .NET depuis le référentiel NuGet», à la page 638
Les IBM MQ classes for XMS .NET sont disponibles pour téléchargement à partir du référentiel NuGet , afin qu'ils puissent être facilement consommés par les développeurs .NET .

Windows Installation de IBM MQ classes for .NET Framework

IBM MQ classes for .NET Framework, y compris les exemples, sont installés avec IBM MQ. Il existe un prérequis pour Microsoft.NET Framework sous Windows.

La version la plus récente de IBM MQ classes for .NET Framework est installée par défaut dans le cadre de l'installation standard de IBM MQ dans la fonction *Java and .NET Messaging and Web Services* . Pour obtenir des instructions d'installation, voir Installation d'un serveur IBM MQ sur Windows ou Installation d'un client IBM MQ sur des systèmes Windows.

Depuis IBM MQ 9.3.0, pour exécuter IBM MQ classes for .NET Framework , vous devez installer Microsoft.NET Framework V4.7.2 ou ultérieure.

Les applications existantes qui sont compilées avec Microsoft.NET Framework V3.5 peuvent être exécutées sans recompilation en ajoutant la balise suivante dans le fichier `app.config` de l'application:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2"/>
 </startup>
</configuration>
```

Remarque : Si Microsoft .NET Framework V4.7.2 ou ultérieure n'est pas installé avant IBM MQ, l'installation du produit IBM MQ se poursuit sans erreur, mais le IBM MQ classes for .NET n'est pas disponible. Si.NET Framework est installé après l'installation de IBM MQ, les assemblages IBM MQ.NET doivent être enregistrés en exécutant le script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , où *WMQInstallDir* correspond au répertoire dans lequel IBM MQ est installé. Ce script installe les assemblages requis dans le cache d'assemblage global (GAC). Un ensemble de fichiers `amqi*.log` qui enregistrent les actions effectuées sont créés dans le répertoire %TEMP% . Il n'est pas nécessaire de réexécuter le script `amqiRegisterdotNet.cmd` si .NET est mis à niveau vers V4.7.2 ou une version ultérieure à partir d'une version antérieure, par exemple, à partir de .NET V3.5.

Dans un environnement à installations multiples, si vous avez précédemment installé le IBM MQ classes for .NET en tant que module de prise en charge, vous ne pouvez pas installer IBM MQ sauf si vous avez d'abord désinstallé le module de prise en charge. La fonction IBM MQ classes for .NET installée avec IBM MQ contient les mêmes fonctions que le module de prise en charge.

Des exemples d'application, y compris des fichiers source, sont également fournis ; voir «Exemples d'application pour .NET», à la page 576.

Pour plus d'informations sur l'utilisation du canal personnalisé IBM MQ pour Microsoft WCF avec .NET, voir «Développement d'applications Microsoft Windows Communication Foundation avec IBM MQ», à la page 1299

Concepts associés

«Installation de IBM MQ classes for .NET», à la page 569
IBM MQ classes for .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux

Tâches associées

Traçage des applications IBM MQ .NET

Options de connexion d' IBM MQ classes for .NET à un gestionnaire de files d'attente

Il existe trois modes de connexion de IBM MQ classes for .NET à un gestionnaire de files d'attente. Déterminez le type de connexion qui convient le mieux à vos besoins.

Connexion de liaisons client

Pour utiliser IBM MQ classes for .NET en tant que IBM MQ MQI client, vous pouvez l'installer, avec IBM MQ MQI client, sur la machine du serveur IBM MQ ou sur une machine distincte. Une connexion de liaisons client peut utiliser des transactions XA ou non XA

Connexion de liaisons de serveur

Lorsqu'il est utilisé en mode liaisons de serveur, IBM MQ classes for .NET utilise l'API du gestionnaire de files d'attente plutôt que de communiquer via un réseau. Cela offre de meilleures performances pour les applications IBM MQ que l'utilisation de connexions réseau.

Pour utiliser la connexion de liaisons, vous devez installer IBM MQ classes for .NET sur le serveur IBM MQ .

Connexion client gérée

Une connexion établie dans ce mode se connecte en tant que client IBM MQ à un serveur IBM MQ s'exécutant sur la machine locale ou distante.

Les IBM MQ classes for .NET qui se connectent dans ce mode restent dans le code géré .NET et n'effectuent aucun appel aux services natifs. Pour plus d'informations sur le code géré, voir la documentation Microsoft .

L'utilisation du client géré est soumise à un certain nombre de limitations. Pour plus d'informations à ce sujet, voir «Connexions client gérées», à la page 593.

Exemples d'application pour .NET

Pour exécuter vos propres applications .NET , utilisez les instructions pour les programmes de vérification, en remplaçant le nom de votre application par celui des exemples d'application.

Les exemples d'application suivants sont fournis:

- Une application de message d'insertion
- Une application d'obtention de message
- Une application'hello world'
- Une application de publication / abonnement
- Une application utilisant des propriétés de message

Tous ces exemples d'application sont fournis en langage C#, et certains sont également fournis en C++ et en Visual Basic. Vous pouvez écrire des applications dans n'importe quelle langue prise en charge par .NET.

Programme "Put message" SPUT (nmqsput.cs, mmqsput.cpp, vmqsput.vb)

Ce programme montre comment placer un message dans une file d'attente nommée. Le programme comporte trois paramètres:

- Nom d'une file d'attente (obligatoire), par exemple, SYSTEM.DEFAULT.LOCAL.QUEUE
- Nom d'un gestionnaire de files d'attente (facultatif)
- Définition d'un canal (facultatif), par exemple, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si aucun nom de gestionnaire de files d'attente n'est indiqué, le gestionnaire de files d'attente par défaut est le gestionnaire de files d'attente local par défaut. Si un canal est défini, il a le même format que la variable d'environnement MQSERVER.

Programme "Obtenir un message" SGET (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Ce programme montre comment extraire un message d'une file d'attente nommée. Le programme comporte trois paramètres:

- Nom d'une file d'attente (obligatoire), par exemple, SYSTEM.DEFAULT.LOCAL.QUEUE
- Nom d'un gestionnaire de files d'attente (facultatif)
- Définition d'un canal (facultatif), par exemple, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si aucun nom de gestionnaire de files d'attente n'est indiqué, le gestionnaire de files d'attente par défaut est le gestionnaire de files d'attente local par défaut. Si un canal est défini, il a le même format que la variable d'environnement MQSERVER.

Programme "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
Ce programme montre comment insérer et obtenir un message. Le programme comporte trois paramètres:

- Nom d'une file d'attente (facultatif), par exemple, SYSTEM.DEFAULT.LOCAL.QUEUE ou SYSTEM.DEFAULT.MODEL.QUEUE
- Nom d'un gestionnaire de files d'attente (facultatif)
- Une définition de canal (facultative), par exemple, SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Si aucun nom de file d'attente n'est indiqué, le nom par défaut est SYSTEM.DEFAULT.LOCAL.QUEUE. Si aucun nom de gestionnaire de files d'attente n'est indiqué, le gestionnaire de files d'attente par défaut est le gestionnaire de files d'attente local par défaut.

Programme "Publish/subscribe" (MQPubSubSample.cs)
Ce programme montre comment utiliser la publication / l'abonnement IBM MQ . Il est fourni en C# uniquement. Le programme comporte deux paramètres:

- Nom d'un gestionnaire de files d'attente (facultatif)
- Une définition de canal (facultatif)

Programme "Propriétés de message" (MQMessagePropertiesSample.cs)
Ce programme montre comment utiliser les propriétés de message. Il est fourni en C# uniquement. Le programme comporte deux paramètres:

- Nom d'un gestionnaire de files d'attente (facultatif)
- Une définition de canal (facultatif)

Vous pouvez vérifier votre installation en compilant et en exécutant ces applications.

Emplacements d'installation

Les exemples d'application sont installés aux emplacements suivants, en fonction de la langue dans laquelle ils sont écrits. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

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

C++ géré

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

Génération des modèles d'application

Pour générer les exemples d'application, un fichier de traitement par lots est fourni pour chaque langage.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

Le fichier bldcssamp.bat contient une ligne pour chaque exemple, qui est tout ce qui est nécessaire pour générer cet exemple de programme:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

C++ géré

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

Le fichier bldmcpsamp.bat contient une ligne pour chaque exemple, qui est tout ce qui est nécessaire pour générer cet exemple de programme:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Si vous souhaitez compiler ces applications sous Microsoft Visual Studio 2003 /.NET SDKv1.1, remplacez la commande de compilation:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

avec

```
cl /clr MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

Visual Basic

MQ_INSTALLATION_PATH\Tools\dotnet\samples\vb\bldvbsamp.bat

Le fichier bldvbsamp.bat contient une ligne pour chaque exemple, qui est tout ce qui est nécessaire pour générer cet exemple de programme:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Exemples d'utilisation de IBM MQ avec Microsoft .NET Core

IBM MQ prend en charge les applications .NET Core for IBM MQ .NET dans les environnements Windows . IBM MQ classes for .NET Standard, y compris les exemples, sont installés par défaut dans le cadre de l'installation standard de IBM MQ .

Les exemples d'application pour IBM MQ .NET sont installés dans `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. Un script est également fourni, qui peut être utilisé pour compiler les exemples.

Vous pouvez générer les exemples à l'aide des fichiers `build.bat` fournis. Il existe un `build.bat` pour chaque exemple à l'emplacement suivant sur Windows:

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux IBM MQ prend également en charge Core pour les applications dans les environnements Linux .

Pour plus d'informations sur l'utilisation de IBM MQ avec Microsoft .NET Core, voir «Installation de IBM MQ classes for .NET», à la page 569.

Configuration de votre gestionnaire de files d'attente pour l'acceptation des connexions client TCP/IP

Configurez un gestionnaire de files d'attente pour qu'il accepte les demandes de connexion entrantes provenant des clients.

Pourquoi et quand exécuter cette tâche

Cette tâche explique les étapes de base de la configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client TCP/IP. Pour un système de production, vous devez également prendre en compte les implications sur la sécurité lors de la configuration des gestionnaires de files d'attente.

Procédure

1. Définissez un canal de connexion serveur:
   a. Démarrez le gestionnaire de files d'attente.
   b. Définissez un exemple de canal appelé NET.CHANNEL:

```
DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for IBM MQ classes for .NET')
```

   Important : Cet exemple est destiné à être utilisé dans un environnement de bac à sable uniquement, car il ne prend pas en compte les implications en matière de sécurité. Pour un système de production, envisagez d'utiliser TLS ou un exit de sécurité. Pour plus d'informations, voir Sécurisation de IBM MQ.
2. Démarrez un programme d'écoute:

```
runmqlsr -t tcp [-m qmnqme ] [-p portnum ]
```

   Remarque : Les crochets indiquent des paramètres facultatifs ; *qmname* n'est pas requis pour le gestionnaire de files d'attente par défaut et le numéro de port *portnum* n'est pas requis si vous utilisez la valeur par défaut (1414).

Transactions réparties dans .NET

Les transactions réparties ou globales permettent aux applications client d'inclure plusieurs sources de données différentes sur deux ou plusieurs systèmes en réseau dans une même transaction.

Dans les transactions réparties, un gestionnaire de transactions coordonne et gère la transaction entre deux ou plusieurs gestionnaires de ressources.

Les transactions peuvent être des processus de validation en une phase ou en deux phases. La validation en une seule phase est un processus dans lequel un seul gestionnaire de ressources participe à la transaction et le processus de validation en deux phases est un processus dans lequel plusieurs gestionnaires de ressources participent à la transaction. Dans le processus de validation en deux phases, le gestionnaire de transactions envoie un appel de préparation pour vérifier si tous les gestionnaires de ressources sont prêts à être validés. Lorsqu'il reçoit l'accusé de réception de tous les gestionnaires de ressources, l'appel de validation est émis. Sinon, une annulation de l'ensemble de la transaction se produit. Pour plus d'informations, voir Gestion et support des transactions . Les gestionnaires de ressources devraient informer les gestionnaires de transactions de leur participation à la transaction. Lorsque le gestionnaire de ressources informe le gestionnaire de transactions de sa participation, le gestionnaire de ressources obtient des rappels du gestionnaire de transactions lorsque la transaction est sur le point d'être validée ou invalidée.

Les classes IBM MQ .NET prennent déjà en charge les transactions réparties dans les connexions en mode liaisons de serveur et non gérées. Dans ces modes, les classes IBM MQ .NET délègue tous ses appels au client de transaction étendue C, qui gère le traitement des transactions pour le compte de .NET.

Les classes IBM MQ.NET prennent désormais en charge les transactions réparties en mode géré où les classes IBM MQ .NET utilisent l'espace de nom System.Transactions pour la prise en charge des transactions réparties. L'infrastructure System.Transactions rend la programmation transactionnelle simple et efficace en prenant en charge les transactions initiées dans tous les gestionnaires de ressources, y compris IBM MQ. L'application IBM MQ .NET peut insérer et extraire des messages à l'aide d'une programmation de transaction implicite .NET ou d'un modèle de programmation de transaction explicite. Dans les transactions implicites, les limites de transaction sont créées par le programme d'application qui décide quand valider, annuler (pour les transactions explicites) ou terminer la transaction. Dans les transactions explicites, vous devez spécifier explicitement si vous souhaitez valider, annuler et terminer la transaction.

IBM MQ.NET utilise le coordinateur de transactions réparties Microsoft (MS DTC) comme gestionnaire de transactions, qui coordonne et gère la transaction entre plusieurs gestionnaires de ressources. IBM MQ est utilisé comme gestionnaire de ressources. Notez que vous ne pouvez pas utiliser TLS avec des transactions XA. Vous devez utiliser CCDT. Pour plus d'informations, voir Utilisation du client transactionnel étendu avec des canaux TLS.

IBM MQ.NET suit le modèle DTP (Distributed Transaction Processing) X/Open. Le modèle de traitement réparti des transactions X/Open est un modèle de traitement réparti des transactions proposé par Open Group, un consortium de fournisseurs. Ce modèle est une norme parmi la plupart des fournisseurs commerciaux dans le traitement des transactions et les domaines de base de données. La plupart des produits de gestion des transactions commerciales prennent en charge le modèle X/DTP.

Modes de transaction

- «Transactions réparties en mode géré .NET», à la page 581
- Transactions distribuées pour le mode non géré

Coordination des transactions dans divers scénarios

- Une connexion peut participer à plusieurs transactions, mais une seule transaction est active à un moment donné.
- Au cours d'une transaction, MQQueueManager.L'appel de déconnexion est pris en compte. Dans ce cas, l'annulation de la transaction est demandée.

- Au cours d'une transaction, l'appel MQQueue.Close ou MQTopic.Close est effectué. Dans ce cas, il est demandé d'annuler la transaction.
- Les limites de transaction sont créées par le programme d'application qui décide quand valider, annuler (pour les transactions explicites) ou terminer (pour les transactions implicites) la transaction.
- Si l'application client se rompt lors d'une transaction avec une erreur inattendue avant d'émettre un appel Put ou Get sur une file d'attente ou un appel de rubrique, la transaction est annulée et une exception MQException est émise.
- Si le code anomalie MQCC_FAILED est renvoyé lors d'un appel Put ou Get sur un appel de file d'attente ou de rubrique, une exception MQException est émise avec le code anomalie et la transaction est annulée. Si un appel de préparation a déjà été émis par le gestionnaire de transactions, IBM MQ .NET renvoie la demande de préparation en annulant de force la transaction. Ensuite, le code défaut du gestionnaire de transactions entraîne une annulation du travail en cours avec tous les gestionnaires de ressources dans les transactions ambiantes en cours.
- Au cours d'une transaction impliquant plusieurs gestionnaires de ressources si une raison d'environnement entraîne un blocage indéfini de l'appel Put ou Get, le gestionnaire de transactions attend jusqu'à une heure définie. Une fois le délai écoulé, il entraîne l'annulation de toutes les tâches en cours avec tous les gestionnaires de ressources dans les transactions ambiantes en cours. Si cette attente indéfinie se produit pendant la phase de préparation, le gestionnaire de transactions peut dépasser le délai d'attente ou émettre un appel en attente de validation sur la ressource, auquel cas la transaction est annulée.
- Les applications qui utilisent des transactions doivent placer ou extraire des messages sous SYNC_POINT. Si un appel d'insertion ou d'extraction de message est émis dans un contexte transactionnel qui n'est pas sous SYNC_POINT, l'appel échoue avec le code anomalie MQRC_UNIT_OF_WORK_NOT_STARTED.

Différences de comportement entre la prise en charge des transactions client gérées et non gérées à l'aide de l'espace de nom Microsoft.NET System.Transactions

Les transactions imbriquées ont un objet TransactionScope dans un autre objet TransactionScope

- Le client IBM MQ .NET entièrement géré ne prend pas en charge l'élément imbriqué TransactionScope
- Le client IBM MQ .NET non géré ne prend pas en charge les TransactionScope imbriquées

Transactions dépendantes de System.Transactions

- Le client IBM MQ .NET entièrement géré prend en charge la fonction de transactions dépendantes fournie par System.Transactions.
- Le client non géré IBM MQ .NET ne prend pas en charge la fonction de transactions dépendantes fournie par System.Transactions.

Exemples de produit

Les exemples de produit `SimpleXAPutet` `SimpleXAGet` sont disponibles sous `WebSphere MQ\tools\dotnet\samples\cs\base`. Les exemples sont des applications C# qui présentent l'utilisation de MQPUT et de MQGET sous Transactions distribuées à l'aide de l'espace de nom SystemTransactions . Pour plus d'informations sur ces exemples, voir «Création de messages d'insertion et d'obtention simples dans une zone TransactionScope», à la page 585.

Transactions réparties en mode géré .NET

Les classes IBM MQ .NET utilisent l'espace de nom System.Transactions pour la prise en charge des transactions réparties en mode géré. En mode géré, MS DTC coordonne et gère les transactions réparties sur tous les serveurs inclus dans une transaction.

Les classes IBM MQ .NET fournissent un modèle de programmation explicite basé sur la classe System.Transactions.Transaction et un modèle de programmation implicite utilisant la classe

System.Transactions.TransactionScope, dans laquelle les transactions sont automatiquement gérées par l'infrastructure.

Transaction implicite

La partie de code suivante décrit comment une application IBM MQ .NET insère un message à l'aide de la programmation de transaction implicite .NET .

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Explication du flux de codes de la transaction implicite

Le code crée *TransactionScope* et place le message sous la portée. Il appelle ensuite *Terminé* pour informer le coordinateur de transaction de l'achèvement de la transaction. Le coordinateur de transaction émet désormais *prepare* et *commit* pour terminer la transaction. Si un problème est détecté, une *annulation* est appelée.

Transaction explicite

Le code suivant décrit comment une application IBM MQ .NET insère des messages à l'aide du modèle de programmation de transaction explicite .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();
}
```

Explication du flux de codes de la transaction explicite

L'élément de code crée une transaction à l'aide de la classe *CommitableTransaction* . Il place un message dans cette portée, puis appelle explicitement *commit* pour terminer la transaction. S'il y a des problèmes, *rollback* est appelé.

Transactions réparties en mode non géré .NET

Les classes IBM MQ.NET prennent en charge les connexions non gérées (client) utilisant le client de transaction étendue et COM + /MTS comme coordinateur de transaction, en utilisant un modèle de programmation de transaction implicite ou explicite. En mode non géré, les classes IBM MQ .NET délèguent tous leurs appels au client de transaction étendue C qui gère le traitement des transactions pour le compte de .NET.

Le traitement des transactions est contrôlé par un gestionnaire de transactions externe, qui coordonne l'unité de travail globale sous le contrôle de l'API du gestionnaire de transactions. Les instructions MQBEGIN, MQCMIT et MQBACK ne sont pas disponibles.IBM MQ Les classes .NET exposent cette prise en charge par le biais de son mode de transport non géré (client C). Voir Configuration de gestionnaires de transactions compatibles XA

MTS est développé en tant que système de traitement des transactions (TP) pour fournir les mêmes fonctions sur Windows NT que celles disponibles dans CICS, Tuxedo et sur d'autres plateformes. Lorsque

MTS est installé, un service distinct est ajouté à Windows NT appelé Microsoft Distributed Transaction Coordinator (MSDTC). MSDTC coordonne les transactions qui s'étendent sur des magasins de données ou des ressources distincts. Pour fonctionner, chaque magasin de données doit implémenter son propre gestionnaire de ressources propriétaire.

IBM MQ devient compatible avec MSDTC en implémentant une interface (interface de gestionnaire de ressources propriétaire) dans laquelle il parvient à mapper les appels DTC XA à des appels IBM MQ(X/Open). IBM MQ joue le rôle de gestionnaire de ressources.

Lorsqu'un composant tel que COM + demande l'accès à un IBM MQ, le COM vérifie généralement avec l'objet de contexte MTS approprié si une transaction est requise. Si une transaction est requise, le COM en informe le code défaut et démarre automatiquement une transaction IBM MQ intégrale pour cette opération. Ensuite, le COM travaille avec les données via le logiciel MQMTS, en plaçant et en obtenant des messages selon les besoins. L'instance d'objet obtenue à partir du COM appelle la méthode SetComplete ou SetAbort une fois que toutes les actions sur les données sont terminées. Lorsque l'application émet la commande SetComplete, l'appel signale au code défaut que l'application a terminé la transaction et le code défaut peut être exécuté avec le processus de validation en deux phases. Le DTC émet ensuite des appels à MQMTS qui, à son tour, émet des appels à IBM MQ pour valider ou annuler la transaction.

Ecriture d'une application IBM MQ .NET à l'aide d'un client non géré

Pour une exécution dans le contexte de COM +, une classe .NET doit hériter de System.EnterpriseServices.ServicedComponent. Les règles et recommandations de création d'assemblages qui utilisent des composants traités sont les suivantes:

Remarque : Les étapes suivantes s'appliquent uniquement si vous utilisez le mode System.EnterpriseServices .

- La classe et la méthode démarrées dans COM + doivent toutes deux être publiques (aucune classe interne et aucune méthode protégée ou statique).
- Attributs de classe et de méthode: l'attribut TransactionOption détermine le niveau de transaction de la classe, c'est-à-dire si les transactions sont désactivées, prises en charge ou requises. L'attribut AutoComplete de la méthode ExecuteUOW() demande à COM + de valider la transaction si aucune exception non gérée n'est émise.
- Nom fort d'un assemblage: l'assemblage doit avoir un nom fort et être enregistré dans le cache d'assemblage global (GAC). L'assemblage est enregistré dans COM + de manière explicite ou par enregistrement paresseux après avoir été enregistré dans le GAC.
- Enregistrement d'un assemblage dans COM +: préparez l'assemblage pour qu'il soit exposé aux clients COM. Créez ensuite une bibliothèque de types à l'aide de l'outil Assembly Registration, regasm.exe.

  ```
  regasm UnmanagedToManagedXa.dll
  ```

- Enregistrez l'assemblage dans GAC `gacutil /i UnmanagedToManagedXa.dll`.
- Enregistrez l'assemblage dans COM + à l'aide de l'outil d'installation des services .NET , regsvcs.exe. Voir la bibliothèque de types créée par regasm.exe:

  ```
  Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
  ```

- L'assemblage est déployé dans le cache d'assemblage global (GAC), puis il est enregistré dans COM + par un enregistrement différé. L'infrastructure .NET s'occupe de l'enregistrement après la première exécution du code.

L'exemple de flux de code utilisant le modèle System.EnterpriseServices et System.Transactions avec COM + sont décrits dans les sections suivantes:

Exemple de flux de code utilisant le modèle 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();
}
```

Exemple de flux de code utilisant System.Transactions pour les interactions avec 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();
}
```

Création de messages d'insertion et d'obtention simples dans une zone TransactionScope

Les exemples d'applications C# du produit sont disponibles dans IBM MQ. Ces applications simples illustrent l'insertion et l'obtention de messages dans une TransactionScope. A la fin de la tâche, vous pourrez insérer et extraire des messages d'une file d'attente ou d'une rubrique.

Avant de commencer

Le service MSDTC doit être en cours d'exécution et activé pour les transactions XA.

Pourquoi et quand exécuter cette tâche

L'exemple est une application simple, `SimpleXAPut` et `SimpleXAGet`. Les programmes `SimpleXAPut` et `SimpleXAGet` sont des applications C# disponibles dans IBM MQ. `SimpleXAPut` illustre l'utilisation de MQPUT, sous Transactions réparties à l'aide de l'espace de nom SystemTransactions . `SimpleXAGet` illustre l'utilisation de MQGET, sous Transactions distribuées à l'aide de l'espace de nom SystemTransactions .

`SimpleXAPut` se trouve dans `MQ\tools\dotnet\samples\cs\base`

Procédure

Les applications peuvent être exécutées avec les paramètres de ligne de commande de `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]
```

où les paramètres sont :

-destinationURI

Il peut s'agir d'une file d'attente ou d'une rubrique. Pour une file d'attente, indiquez `queue://queueName` et pour une rubrique, indiquez `topic://topicName`.

-host

Il peut s'agir d'un nom d'hôte tel que localhost ou d'une adresse IP.

-port

Port sur lequel le gestionnaire de files d'attente s'exécute.

-channel

Canal de connexion utilisé. La valeur par défaut est `SYSTEM.DEF.SVRCONN`

-transaction

Résultat de la transaction, par exemple validation ou annulation.

-mode

Mode de transport, par exemple géré ou non géré.

-numberOfMsgs

Nombre de messages. La valeur par défaut est 1.

Exemple

```
SimpleXAPut -d topic://T01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

```
SimpleXAGet -d queue://Q01 -h localhost -p 2345 -tx rollback -tm unmanaged
```

Récupération de transactions dans IBM MQ .NET

Cette section décrit le processus de récupération des transactions dans IBM MQ .NET XA à l'aide du mode géré.

Pourquoi et quand exécuter cette tâche

Dans le traitement des transactions réparties, les transactions peuvent aboutir, mais il peut exister des scénarios dans lesquels une transaction peut échouer pour de nombreuses raisons. Ces raisons peuvent inclure une panne système, une panne matérielle, une erreur réseau, des données incorrectes ou non valides, des erreurs d'application ou des catastrophes naturelles ou causées par l'homme. Il n'est pas possible d'éviter les échecs de transaction. Le système de transactions réparties doit être capable de gérer ces incidents. Il doit être capable de détecter et de corriger les erreurs lorsqu'elles se produisent. Ce processus est connu sous le nom de récupération de transaction.

Un aspect important du traitement réparti des transactions consiste à récupérer les transactions incomplètes ou en attente de validation. Il est essentiel d'exécuter la récupération car la partie Unité de travail d'une transaction particulière est verrouillée jusqu'à ce qu'elle soit récupérée. Microsoft.NET à partir de sa bibliothèque de classes System.Transactions offre la possibilité de récupérer des transactions incomplètes / en attente de validation. Cette prise en charge de la reprise s'attend à ce que le Resource Manager gère les journaux de transactions et exécute la reprise lorsque cela est nécessaire.

Dans le modèle de récupération des transactions Microsoft .NET , le gestionnaire de transactions (System.Transactionsou Microsoft Distributed Transaction (MS DTC), ou les deux), initie, coordonne et contrôle la récupération des transactions. Les gestionnaires de ressources basés sur le protocole OLE Tx (protocole Microsoft XA) fournissent les options permettant de configurer le code défaut pour piloter, coordonner et contrôler la reprise pour ces derniers. Pour ce faire, les gestionnaires de ressources doivent enregistrer XA_Switch avec MS DTC à l'aide de l'interface native.

XA_Switch fournit les points d'entrée des fonctions XA telles que xa_start, xa_end et xa_recover dans le Resource Manager au Distributed Transaction Coordinator.

Reprise à l'aide du coordinateur Microsoft Distributed Transaction (DTC):

Le coordinateur Microsoft Distributed Transaction fournit deux types de processus de reprise.

Récupération à froid

La récupération à froid est effectuée si le processus du gestionnaire de transactions échoue alors qu'une connexion à un gestionnaire de ressources XA est ouverte. Lorsque le gestionnaire de transactions redémarre, il lit les journaux du gestionnaire de transactions et rétablit la connexion au gestionnaire de ressources XA, puis lance la reprise.

Récupération à chaud

La reprise à chaud est effectuée si le gestionnaire de transactions reste actif alors que la connexion entre le gestionnaire de transactions et le gestionnaire de ressources XA échoue en raison de l'échec du gestionnaire de ressources XA ou du réseau. Après l'échec, le gestionnaire de transactions tente régulièrement de se reconnecter au gestionnaire de ressources XA. Lorsque la connexion est rétablie, le gestionnaire de transactions lance la reprise XA.

L'espace de nom System.Transactions fournit une implémentation gérée des transactions distribuées basées sur MS DTC en tant que gestionnaire de transactions. Il fournit des fonctions similaires à celles de l'interface native de MS DTC, mais dans un environnement entièrement géré. La seule différence concerne la récupération des transactions. System.Transactions attend des gestionnaires

de ressources qu'ils gèrent eux-mêmes la récupération, puis qu'ils se coordonnent avec les gestionnaires de transactions (MS DTC). Le Resource Manager doit demander la récupération d'une transaction incomplète particulière, puis le gestionnaire de transactions l'accepte et se coordonne en fonction du résultat réel de cette transaction particulière.

Processus de reprise de transaction pour IBM MQ .NET

Cette section décrit comment les transactions réparties peuvent être récupérées avec les classes IBM MQ .NET .

Présentation

Pour récupérer une transaction incomplète, les informations de récupération sont requises. Les informations de reprise de transaction doivent être consignées dans le stockage par les gestionnaires de ressources. IBM MQ Les classes .NET suivent un chemin similaire. Les informations de reprise des transactions sont consignées dans une file d'attente système appelée SYSTEM.DOTNET.XARECOVERY.QUEUE.

La récupération des transactions dans IBM MQ .NET est un processus en deux étapes:

1. Consignation des informations de reprise de transaction dans SYSTEM.DOTNET.XARECOVERY.QUEUE.
2. Récupération de transactions à l'aide de l'application XA Monitor WmqDotnetXAMonitor.

SYSTEM.DOTNET.XARECOVERY.QUEUE

SYSTEME SYSTEM.DOTNET.XARECOVERY.QUEUE est une file d'attente système qui contient les informations de récupération des transactions pour les transactions incomplètes. Cette file d'attente est créée lors de la création d'un gestionnaire de files d'attente.

Pour chaque transaction, lors de la phase de préparation, un message persistant contenant les informations de reprise est ajouté à SYSTEM.DOTNET.XARECOVERY.QUEUE. Le message est supprimé si l'appel de validation aboutit.

Remarque : Vous ne devez pas supprimer SYSTEM.DOTNET.XARECOVERY.QUEUE .

Application WMQDotnetXAMonitor

L'application IBM MQ .NET XA Monitor, WmqDotnetXAMonitor, est une application gérée par .NET qui surveille un gestionnaire de files d'attente et traite les messages dans SYSTEM.DOTNET.XARECOVERY.QUEUE et récupère les transactions incomplètes

Si l'agent MCA ne parvient pas à placer le message dans la file d'attente de destination, il génère un rapport d'exception contenant le message d'origine et le place dans une file d'attente de transmission à envoyer à la file d'attente de réponse indiquée dans le message d'origine. (Si la file d'attente de réponse se trouve dans le même gestionnaire de files d'attente que l'agent MCA, le message est placé directement dans cette file d'attente et non dans une file d'attente de transmission.)

Sont réputées être des transactions incomplètes et sont recouvrées:

- Si la transaction est préparée mais que COMMIT ne s'est pas terminé dans le délai imparti.
- Si la transaction est préparée mais que le gestionnaire de files d'attente IBM MQ a été arrêté.
- Si la transaction est préparée, mais que le gestionnaire de transactions est arrêté.

  L'application du moniteur XA doit être exécutée à partir du même système que celui sur lequel s'exécute l'application client IBM MQ .NET . Si des applications s'exécutent sur plusieurs systèmes et se connectent au même gestionnaire de files d'attente, l'application XAMonitor WmqDotnetdoit être exécutée à partir de tous les systèmes. Bien que chaque machine client dispose d'une instance de l'application de moniteur XA en cours d'exécution pour récupérer l'application, chaque instance de moniteur XA doit être en mesure d'identifier le message correspondant à la transaction coordonnée par le code défaut MS local du moniteur XA en cours afin de pouvoir le réinscrire et le terminer.

Concepts associés

«Cas d'utilisation de la reprise de transaction pour IBM MQ .NET», à la page 588
Il existe plusieurs cas d'utilisation à partir desquels les transactions peuvent avoir besoin d'être récupérées.

Tâches associées

«Utilisation de l'application WMQDotnetXAMonitor», à la page 589
Le client IBM MQ .NET fournit une application de surveillance XA, WmqDotnetXAMonitor, que vous pouvez utiliser pour récupérer toutes les transactions distribuées incomplètes. L'application XAMonitor WmqDotnetétablit une connexion au gestionnaire de files d'attente dans lequel les transactions sont en attente de validation, puis résout la transaction en fonction des paramètres que vous avez définis.

Cas d'utilisation de la reprise de transaction pour IBM MQ .NET

Il existe plusieurs cas d'utilisation à partir desquels les transactions peuvent avoir besoin d'être récupérées.

- **IBM MQ Application utilisant le code défaut unique et une instance de gestionnaire de files d'attente unique:** Dans ce cas d'utilisation, lorsque vous vous connectez au gestionnaire de files d'attente et que vous exécutez l'unité de travail (UoW) sous la transaction, et si la transaction échoue et devient incomplète, l'application du moniteur XA récupère la transaction et la termine.

  Dans ce cas d'utilisation, une seule instance de l'application du moniteur XA est en cours d'exécution, car un seul gestionnaire de files d'attente est associé aux transactions.
- **Plusieurs applications IBM MQ utilisant un code défaut unique et une instance de gestionnaire de files d'attente unique:** Dans ce cas d'utilisation, il existe plusieurs applications IBM MQ sous un code défaut unique et toutes se connectent au même gestionnaire de files d'attente et exécutent UoW sous des transactions.

  Si les transactions échouent et deviennent incomplètes, l'application du moniteur XA les récupère et termine les transactions appartenant à toutes les applications.

  Dans ce cas d'utilisation, une seule instance de l'application du moniteur XA s'exécute, car un gestionnaire de files d'attente est utilisé dans les transactions.
- **Plusieurs applications IBM MQ , plusieurs codes d'accès client, différentes instances de gestionnaire de files d'attente:** Dans ce cas d'utilisation, il existe plusieurs applications IBM MQ sous des codes d'accès client différents (c'est-à-dire que chaque application s'exécute sur une machine différente) et se connectent à des gestionnaires de files d'attente différents.

  Si un incident se produit et que la transaction est incomplète, l'application de surveillance vérifie l'emplacement de TransactionManagerdans le message afin de déterminer l'adresse DTC. Si la valeur TransactionManagercorrespond à l'adresse DTC sous laquelle le moniteur s'exécute, elle effectue la reprise. Sinon, elle poursuit la recherche jusqu'à ce que le message correspondant à son code DTC soit trouvé.

  Dans ce cas d'utilisation, une seule instance de l'application du moniteur XA est exécutée par client (utilisateur ou ordinateur) car chaque client possède son propre gestionnaire de files d'attente utilisé dans les transactions.
- **Plusieurs applications IBM MQ , plusieurs codes d'accès client, plusieurs mêmes instances de gestionnaire de files d'attente:** Dans ce cas d'utilisation, il existe plusieurs applications IBM MQ sous des codes d'accès client différents (c'est-à-dire que chaque application s'exécute sur une machine différente) et toutes se connectent au même gestionnaire de files d'attente.

  Si un incident se produit et que la transaction devient incomplète, l'application de surveillance vérifie la localisation TransactionManagerdans le message pour vérifier si l'adresse et la valeur du code défaut correspondent au code défaut sous lequel le moniteur est exécuté. Si les deux valeurs correspondent, la reprise se poursuit jusqu'à ce que le message correspondant à son code défaut soit trouvé.

  Dans ce cas d'utilisation, une seule instance de l'application du moniteur XA sera exécutée par client (utilisateur ou ordinateur), car chaque client possède sa propre association de gestionnaire de files d'attente utilisée dans les transactions.

- **Plusieurs applications IBM MQ , code défaut unique, différentes instances de gestionnaire de files d'attente:** Dans ce cas d'utilisation, il existe plusieurs applications IBM MQ sous un code défaut unique (c'est-à-dire, sur un ordinateur, plusieurs applications IBM MQ sont en cours d'exécution) et se connectent à différents gestionnaires de files d'attente.

  Si la transaction échoue et devient incomplète, l'application de surveillance récupère la transaction.

  Dans ce cas d'utilisation, il y aura autant d'instances d'application de surveillance en cours d'exécution que de gestionnaires de files d'attente connectés, car chaque application possède son propre gestionnaire de files d'attente utilisé dans les transactions et chacune d'elles doit être récupérée.

  Remarque : Si l'application du moniteur XA n'est pas en cours d'exécution en arrière-plan, vous pouvez la démarrer.

Concepts associés

«Processus de reprise de transaction pour IBM MQ .NET», à la page 587
Cette section décrit comment les transactions réparties peuvent être récupérées avec les classes IBM MQ .NET .

Tâches associées

«Utilisation de l'application WMQDotnetXAMonitor», à la page 589
Le client IBM MQ .NET fournit une application de surveillance XA, WmqDotnetXAMonitor, que vous pouvez utiliser pour récupérer toutes les transactions distribuées incomplètes. L'application XAMonitor WmqDotnetétablit une connexion au gestionnaire de files d'attente dans lequel les transactions sont en attente de validation, puis résout la transaction en fonction des paramètres que vous avez définis.

Utilisation de l'application WMQDotnetXAMonitor

Le client IBM MQ .NET fournit une application de surveillance XA, WmqDotnetXAMonitor, que vous pouvez utiliser pour récupérer toutes les transactions distribuées incomplètes. L'application XAMonitor WmqDotnetétablit une connexion au gestionnaire de files d'attente dans lequel les transactions sont en attente de validation, puis résout la transaction en fonction des paramètres que vous avez définis.

Pourquoi et quand exécuter cette tâche

L'application WMQDotnetXAMonitor doit être exécutée manuellement. Il peut être démarré à tout moment. Vous pouvez le démarrer lorsque vous voyez les messages sur SYSTEM.DOTNET.XARECOVERY.QUEUE ou vous pouvez l'exécuter en arrière-plan avant d'effectuer un travail transactionnel avec les applications écrites à l'aide des classes IBM MQ .NET .

Vous pouvez définir les valeurs de paramètre pour WMQDotnetXAMonitor via la ligne de commande ou à l'aide d'un fichier de configuration d'application. Les valeurs fournies via le fichier de configuration d'application sont prioritaires par rapport aux valeurs définies via la ligne de commande.

Avant IBM MQ 9.3.0, la connexion établie par WMQDotnetXAMonitor est une connexion non sécurisée.

Depuis la IBM MQ 9.3.0, vous avez la possibilité d'établir une connexion sécurisée au gestionnaire de files d'attente en définissant des paramètres supplémentaires pour WMQDotnetXAMonitor.

Procédure

- Pour fournir une entrée à WmqDotNETXAMonitor à l'aide d'un fichier de configuration d'application, voir «Paramètres du fichier de configuration de l'application WmqDotNETXAMonitor», à la page 591.
- Pour démarrer l'application WMQDotnetXAMonitor à partir de la ligne de commande, utilisez la commande suivante avec les paramètres dont vous avez besoin:

  Avant IBM MQ 9.3.0:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

Depuis IBM MQ 9.3.0:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i -k SSL Key
Repository -s Cipher Spec
```

Les paramètres que vous pouvez spécifier sont les suivants:

– **-m *QueueManagerNom***
  Nom du gestionnaire de files d'attente.

  Facultatif

  -n *ConnectionName*
  Nom de connexion au format hôte (port). *ConnectionName* peut contenir plusieurs noms de connexion. Plusieurs noms de connexion doivent être indiqués dans une liste séparée par des virgules, par exemple `localhost (1414), localhost (1415), localhost (1416)`. L'application WMQDotnetXAMonitor exécute la récupération pour chacun des noms de connexion spécifiés dans la liste séparée par des virgules.

  -c *NomCanal*
  Nom du canal.

  -i
  Fin de la branche heuristique.

  Facultatif

  -k *Référentiel de clés SSL*
  Nom du référentiel de clés SSL. Les valeurs prises en charge sont les suivantes :

  - *SYSTEM (valeur par défaut)
  - *UTILISATEUR

  Facultatif

  -s *Spécification de chiffrement*
  Le CipherSpec que vous définissez doit être l'un des CipherSpecs de la version prise en charge. Il peut de préférence être identique à celui spécifié dans la stratégie de groupe Windows . Pour plus d'informations, voir «Prise en charge de CipherSpec pour le client .NET géré», à la page 613.

  Obligatoire pour établir une connexion sécurisée au gestionnaire de files d'attente.

  -dn *Nom d'homologue SSL*
  Nom d'homologue SSL utilisé pour vérifier le nom distinctif (DN) du certificat à partir du gestionnaire de files d'attente homologue.

  Facultatif

  -cl *Libellé du certificat*
  Nom de libellé qui identifie le certificat.

  Facultatif

  -sn *OutboundSNI*
  Indique si l'indication de nom de serveur (SNI) doit être définie sur le nom de canal IBM MQ cible sur le système distant lors du lancement d'une connexion TLS ou sur le nom d'hôte. Les valeurs prises en charge pour cette option sont les suivantes:

  - CHANNEL (il s'agit de la valeur par défaut)
  - HOSTNAME
  - *

  Si aucune valeur n'est définie, la valeur par défaut, `CHANNEL`, est utilisée.

  Facultatif

-cr ***Vérification de révocation de certificat***
Indique si la vérification de la révocation de certification doit être effectuée. Les valeurs prises en charge pour cette option sont les suivantes:

- Oui
- false (il s'agit de la valeur par défaut)

Facultatif

-kr ***KeyReset***
Nombre total d'octets non chiffrés envoyés et reçus sur le canal avant que la clé secrète utilisée pour le chiffrement ne soit renégociée.

La valeur par défaut 0 indique que les clés secrètes ne sont jamais renégociées

Facultatif

L'application WMQDotnetXAMonitor effectue les actions suivantes:

1. Vérifie la longueur de la file d'attente de SYSTEM.DOTNET.XARECOVERY.QUEUE à un intervalle de 100 secondes.
2. Si la longueur de la file d'attente est supérieure à zéro, parcourt la file d'attente à la recherche de messages et vérifie si les messages répondent aux critères de transaction incomplets.
3. Si un message répond aux critères de transaction incomplète, il est extrait et extrait les informations de récupération de la transaction.
4. Détermine si les informations de reprise sont liées au Microsoft Distributed Transaction Coordinator (MS DTC) local. Si tel est le cas, WMQDotnetXAMonitor procède à la récupération de la transaction, sinon il revient à parcourir le message suivant.
5. Effectue des appels au gestionnaire de files d'attente pour récupérer la transaction incomplète.

Paramètres du fichier de configuration de l'application WmqDotNETXAMonitor
Vous pouvez fournir une entrée à l'application IBM MQ .NET XA Monitor, WmqDotNETXAMonitor, à l'aide d'un fichier de configuration d'application. Un exemple de fichier de configuration d'application est fourni avec IBM MQ .NET. Vous pouvez modifier cet exemple de fichier en fonction de vos besoins.

Les valeurs d'entrée fournies via le fichier de configuration d'application ont la priorité la plus élevée. Si vous fournissez des valeurs d'entrée à la ligne de commande comme décrit dans «Utilisation de l'application WMQDotnetXAMonitor», à la page 589 et dans le fichier de configuration d'application, les valeurs du fichier de configuration d'application sont prioritaires.

Exemple de fichier de configuration d'application pour les fichiers antérieurs à IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
</dnetxa>
</dnetxa>
</configuration>
```

Exemple de fichier de configuration d'application à partir de IBM MQ 9.3.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<configSections>
```

```
<sectionGroup name="IBM.WMQ">
<section name="dnetxa" type="System.Configuration.NameValueFileSectionHandler" />
</sectionGroup>
</configSections>
<IBM.WMQ>
<dnetxa>
<add key="ConnectionName" value=""/>
<add key="ChannelName" value="" />
<add key="QueueManagerName" value="" />
<add key="UserId" value="" />
<add key="SecurityExit" value="" />
<add key="SecurityExitUserData" value = "">
<add key="SSLKeyRepository" value="" />
<add key="SSLCipherSpec" value="" />
<add key="SSLPeerName" value="" />
<add key="SSLKeyResetCount" value="" />
<add key="SSLCertRevocationCheck" value="" />
<add key="CertificateLabel" value="" />
<add key="OutboundSNI" value="" />
</dnetxa>
</dnetxa>
</configuration>
```

WmqDotNetXAMonitor
L'application Monitor crée un fichier journal dans le répertoire de l'application pour consigner la progression du moniteur et l'état de reprise des transactions. La consignation commence par le nom de la connexion et les détails du canal pour afficher le gestionnaire de files d'attente en cours pour lequel la récupération est en cours.

Une fois la récupération démarrée, l'ID MessageId du message de récupération de transaction, TransactionId de la transaction incomplète et la sortie réelle de la transaction, conformément à la coordination du gestionnaire de transactions, sont consignés.

Exemple de fichier journal:

```
Time|ProcessId|ThreadId|WMQ .NET XA Recovery Monitor, Running now for
ConnectionName:xxxx, Time|ProcessId|ThreadId|Channel=xxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Roll back
Time|ProcessId|ThreadId| Recovery Completed for TransactionId= xxxxx
```

Ecriture et déploiement de programmes IBM MQ .NET

Pour utiliser IBM MQ classes for .NET pour accéder aux files d'attente IBM MQ , vous écrivez des programmes dans n'importe quelle langue prise en charge par .NET contenant des appels qui placent des messages dans des files d'attente IBM MQ et en extraient.

Avant de commencer

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client IBM MQ .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Pourquoi et quand exécuter cette tâche

La documentation IBM MQ contient des informations uniquement sur les langages C#, C++ et Visual Basic.

Les rubriques de cette section fournissent des informations pour vous aider à écrire des applications afin d'interagir avec les systèmes IBM MQ . Pour plus de détails sur les classes individuelles, voir Classes et interfaces IBM MQ .NET.

Différences de connexion

La façon dont vous programmez pour IBM MQ.NET comporte des dépendances sur les modes de connexion que vous souhaitez utiliser.

Lorsque IBM MQ classes for .NET est utilisé en tant que client géré, il existe un certain nombre de différences par rapport à un IBM MQ MQI clientstandard, car certaines fonctions ne sont pas disponibles pour un client géré.

IBM MQ.NET détermine le type de connexion à utiliser à partir des paramètres que vous spécifiez pour le nom de connexion, le nom de canal, la valeur de personnalisation NMQ_MQ_LIB et la propriété MQC.TRANSPORT_PROPERTY.

Connexions client gérées

Lorsque les IBM MQ classes for .NET sont utilisés en tant que client géré, il existe un certain nombre de différences par rapport à un IBM MQ MQI clientstandard.

Les fonctions suivantes ne sont pas disponibles pour un client géré:

- Compression de canal
- Chaînage d'exit de canal

Si vous tentez d'utiliser ces fonctions avec un client géré, une exception MQException est renvoyée. Si l'erreur est détectée à l'extrémité client d'une connexion, elle utilise le code anomalie MQRC_ENVIRONMENT_ERROR. S'il est détecté à l'extrémité du serveur, le code anomalie renvoyé par le serveur sera utilisé.

Les exits de canal écrits pour un client non géré ne fonctionnent pas. Vous devez écrire de nouveaux exits spécifiquement pour le client géré. Vérifiez qu'aucun exit de canal non valide n'est spécifié dans la table de définition de canal du client (CCDT).

Le nom d'un exit de canal géré peut comporter jusqu'à 999 caractères. Toutefois, si vous utilisez la table de définition de canal du client pour spécifier le nom de l'exit de canal, il est limité à 128 caractères.

La communication est prise en charge uniquement via TCP/IP.

Lorsque vous arrêtez un gestionnaire de files d'attente à l'aide de la commande **endmqm** , la fermeture d'un canal de connexion serveur à un client géré par .NET peut prendre plus de temps que celle d'un canal de connexion serveur à d'autres clients.

Si vous avez défini *NMQ_MQ_LIB* sur `managed` pour utiliser les diagnostics d'incident IBM MQ gérés, aucun des paramètres -i, -p, -s, -b ou -c de la commande **strmqtrc** n'est pris en charge.

Une application .NET gérée utilisant des transactions XA ne fonctionnera pas avec un gestionnaire de files d'attente z/OS . Un client .NET géré qui tente de se connecter à un gestionnaire de files d'attente z/OS échoue avec une erreur, MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354), lors d'un appel MQOPEN. Toutefois, une application .NET gérée utilisant des transactions XA fonctionne avec le gestionnaire de files d'attente réparties.

Définition du type de connexion à utiliser

Le type de connexion est déterminé par le paramètre du nom de connexion, du nom de canal, de la valeur de personnalisation NMQ_MQ_LIB et de la propriété MQC.TRANSPORT_PROPERTY.

Vous pouvez spécifier le nom de connexion comme suit:

- Explicitement sur un constructeur MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- En définissant les propriétés MQC.HOST_NAME_PROPERTY et, éventuellement, MQC.PORT_PROPERTY dans une entrée de table de hachage sur un constructeur MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- En tant que valeurs MQEnvironment explicites

```
MQEnvironment.Hostname
```

`MQEnvironment.Port` (Facultatif)

- En définissant les propriétés MQC.HOST_NAME_PROPERTY et, éventuellement, MQC.PORT_PROPERTY dans la table de hachage MQEnvironment.properties .

Vous pouvez spécifier le nom de canal comme suit:

- Explicitement sur un constructeur MQQueueManager :

```
public MQQueueManager(String queueManagerName, MQLONG Options, string Channel,
string ConnName)
```

```
public MQQueueManager(String queueManagerName, string Channel, string ConnName)
```

- En définissant la propriété MQC.CHANNEL_PROPERTY dans une entrée de table de hachage sur un constructeur MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- En tant que valeur MQEnvironment explicite

```
MQEnvironment.Channel
```

- En définissant la propriété MQC.CHANNEL_PROPERTY dans la table de hachage MQEnvironment.properties .

Vous pouvez spécifier la propriété de transport comme suit:

- En définissant la propriété MQC.TRANSPORT_PROPERTY dans une entrée de table de hachage sur un constructeur MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- En définissant la propriété MQC.TRANSPORT_PROPERTY dans la table de hachage MQEnvironment.properties .

Sélectionnez le type de connexion dont vous avez besoin en utilisant l'une des valeurs suivantes:

MQC.TRANSPORT_MQSERIES_BINDINGS -connexion en tant que serveur
MQC.TRANSPORT_MQSERIES_CLIENT -se connecte en tant que client non XA
MQC.TRANSPORT_MQSERIES_XACLIENT -se connecte en tant que client XA
MQC.TRANSPORT_MQSERIES_MANAGED -connexion en tant que client géré non XA

Vous pouvez définir la valeur de personnalisation NMQ_MQ_LIB pour choisir explicitement le type de connexion, comme indiqué dans le tableau suivant.

Valeur NMQ_MQ_LIB	Type de connexion
mqic.dll	Se connecter en tant que client non XA
mqicxa.dll	Se connecter en tant que client XA
mqm.dll	Connexion en tant que serveur ou en tant que client non XA
géré	Se connecter en tant que client géré non XA

Remarque : Les valeurs de mqic32.dll et mqic32xa.dll sont acceptées comme synonymes de mqic.dll et mqicxa.dll pour la compatibilité avec les versions antérieures. Toutefois, mqm.dll et mqm.pdb ne font partie que du package client à partir de IBM WebSphere MQ 7.1 .

Si vous choisissez un type de connexion qui n'est pas disponible dans votre environnement, par exemple si vous spécifiez mqic32xa.dll et que vous ne disposez pas de la prise en charge XA, IBM MQ.NET émet une exception.

La définition de NMQ_MQ_LIB sur "managed" permet au client d'utiliser des tests de diagnostic d'incident IBM MQ gérés, la conversion de données .NET et d'autres fonctions IBM MQ de bas niveau gérées.

Toutes les autres valeurs de NMQ_MQ_LIB entraînent le processus .NET à utiliser des tests de diagnostic d'incident IBM MQ non gérés et la conversion de données, ainsi que d'autres fonctions IBM MQ de niveau inférieur non gérées (en supposant qu'un IBM MQ MQI client ou un serveur est installé sur le système).

IBM MQ.NET choisit le type de connexion comme suit:

1. Si MQC.TRANSPORT_PROPERTY est spécifié, il se connecte en fonction de la valeur de MQC.TRANSPORT_PROPERTY.

   Notez, cependant, que le paramètre MQC.TRANSPORT_PROPERTY vers MQC.TRANSPORT_MQSERIES_MANAGED ne garantit pas que le processus client s'exécute de manière gérée. Même avec ce paramètre, le client n'est pas géré dans les cas suivants:

   - Si une autre unité d'exécution du processus s'est connectée à MQC.TRANSPORT_PROPERTY défini sur une valeur autre que MQC.TRANSPORT_MQSERIES_MANAGED.
   - Si NMQ_MQ_LIB n'est pas défini sur "managed", les tests de diagnostic d'incident, la conversion de données et d'autres fonctions de bas niveau ne sont pas entièrement gérés (en supposant qu'un IBM MQ MQI client ou un serveur est installé sur le système).
2. Si un nom de connexion a été spécifié sans nom de canal ou qu'un nom de canal a été spécifié sans nom de connexion, une erreur est générée.
3. Si un nom de connexion et un nom de canal ont été spécifiés:
   - Si NMQ_MQ_LIB est défini sur mqic32xa.dll, il se connecte en tant que client XA.
   - Si NMQ_MQ_LIB est défini sur géré, il se connecte en tant que client géré.
   - Sinon, il se connecte en tant que client non XA.
4. Si NMQ_MQ_LIB est spécifié, il se connecte en fonction de la valeur de NMQ_MQ_LIB.
5. Si un serveur IBM MQ est installé, il se connecte en tant que serveur.
6. Si un IBM MQ MQI client est installé, il se connecte en tant que client non XA.
7. Sinon, il se connecte en tant que client géré.

Windows Utilisation du canevas de projet IBM MQ .NET

Le client IBM MQ .NET vous permet d'utiliser un modèle de projet pour vous aider à développer vos applications .NET Core .

Avant de commencer

Vous devez disposer de Microsoft Visual Studio 2017ou version ultérieure et de .NET Core 2.1 sur votre système.

Vous devez copier le modèle .NET à partir du

&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip

vers le répertoire

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

répertoire, où:

- *& MQ_INSTALL_ROOT* est le répertoire racine de votre installation
- *& USER_HOME_DIRECTORY* est votre répertoire de base.

Vous devez arrêter et redémarrer Microsoft Visual Studio pour récupérer le modèle.

Pourquoi et quand exécuter cette tâche

Le canevas de projet .NET inclut du code commun que vous pouvez utiliser pour vous aider à développer vos applications. Avec le code intégré, vous pouvez vous connecter au gestionnaire de files d'attente IBM MQ et effectuer une opération d'insertion ou d'extraction en modifiant simplement les propriétés dans le code intégré.

Procédure

1. Ouvrez Microsoft Visual Studio.
2. Cliquez sur **Fichier**, puis sur **Nouveau** , puis sur **Projet**.
3. Dans la *fenêtre Créer un projet*, sélectionnez `IBM MQ .NET Client App (.NET Core)` et cliquez sur **Suivant**.
4. Dans la fenêtre *Configurer votre nouveau projet* , modifiez le *nom de projet* de votre projet si vous le souhaitez, puis cliquez sur **Créer** pour créer le projet .NET .

   `MQDotnetApp.cs` est le fichier créé avec le fichier de projet. Ce fichier contient le code qui se connecte au gestionnaire de files d'attente et effectue une opération d'insertion et d'extraction.

   Les propriétés de connexion sont définies sur les valeurs par défaut:

   - MQC.CONNECTION_NAME_PROPERTY est défini sur *localhost (1414)*
   - MQC.CHANNEL_PROPERTY est défini sur *DOTNET.SVRCONN*

   La file d'attente est définie sur *Q1*et vous pouvez modifier ces propriétés en conséquence.
5. Compilez et exécutez l'application.

Concepts associés

Composants et fonctions d'IBM MQ

Environnement d'application .NET - Windows uniquement

Fichiers de configuration pour IBM MQ classes for .NET

Une application client .NET peut utiliser un fichier de configuration IBM MQ MQI client et, si vous utilisez le type de connexion gérée, un fichier de configuration d'application .NET . Les paramètres du fichier de configuration de l'application sont prioritaires.

Fichier de configuration du client

Une application client IBM MQ classes for .NET peut utiliser un fichier de configuration client de la même manière que n'importe quel autre IBM MQ MQI client. Ce fichier est généralement appelé `mqclient.ini`, mais vous pouvez spécifier un autre nom de fichier. Pour plus d'informations sur le fichier de configuration client, voir le fichier de configuration IBM MQ MQI client , `mqclient.ini`.

Seuls les attributs suivants d'un fichier de configuration IBM MQ MQI client sont pertinents pour IBM MQ classes for .NET. Si vous spécifiez d'autres attributs, cela n'a aucun effet.

Tableau 77. Attributs du fichier de configuration client relatifs à IBM MQ classes for .NET

section	**Attribut**
Canaux	CCSID
Canaux	Répertoire ChannelDefinition
Canaux	Fichier ChannelDefinition
Canaux	ReconDelay
Canaux	DefRecon
Canaux	MQReconnectTimeout
Canaux	Paramètres ServerConnection
Canaux	Put1DefaultAlwaysSync
Canaux	PasswordProtection
ClientExit-Chemin	ExitsDefaultPath
ClientExit-Chemin	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
Sécurité	Fichier MQIInitialKey
SSL	SSLKeyRepository
SSL	Mot de passe SSLKeyRepository
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

Vous pouvez remplacer ces attributs à l'aide de la variable d'environnement appropriée.

Fichier de configuration d'application

Si vous utilisez le type de connexion gérée, vous pouvez également remplacer le fichier de configuration du client IBM MQ et les variables d'environnement équivalentes à l'aide du fichier de configuration de l'application .NET .

Les paramètres du fichier de configuration de l'application .NET ne sont traités que lors de l'exécution avec le type de connexion gérée et sont ignorés pour les autres types de connexion.

Le fichier de configuration de l'application .NET et son format sont définis par Microsoft pour une utilisation générale dans l'infrastructure .NET , mais les noms de section, les clés et les valeurs mentionnés dans cette documentation sont spécifiques à IBM MQ.

Le format du fichier de configuration d'application .NET est un certain nombre de *sections*. Chaque section contient une ou plusieurs *clés*, et chaque clé est associée à une *valeur*. L'exemple suivant illustre les sections, les clés et les valeurs utilisées dans un fichier de configuration d'application .NET pour contrôler la propriété TCP/IP KeepAlive :

```
<configuration>
  <configSections>
    <section name="TCP" type="System.Configuration.NameValueSectionHandler"/>
  </configSections>
  <TCP>
    <add key="KeepAlive" value="true"></add>
  </TCP>
<configuration>
```

Les mots clés utilisés dans les noms de section et les clés du fichier de configuration de l'application .NET correspondent exactement aux mots clés des strophes et des attributs définis dans le fichier de configuration du client.

La section `<configSections>` doit être le premier élément enfant de l'élément `<configuration>` .

Pour plus d'informations, voir la documentation d'Microsoft.

Exemple de fragment de code C# à utiliser avec .NET

Fragment de code C# démontrant qu'une application se connecte à un gestionnaire de files d'attente, place un message dans une file d'attente et reçoit une réponse.

Le fragment de code C# suivant illustre une application qui effectue trois actions:

1. Connexion à un gestionnaire de files d'attente
2. Placez un message dans SYSTEM.DEFAULT.LOCAL.QUEUE
3. Récupérer le message

Il montre également comment modifier le type de connexion.

```
// ============================================================================
// Licensed Materials - Property of IBM
// 5724-H72
// (c) Copyright IBM Corp. 2003, 2024
// ============================================================================
using System;
using System.Collections;

using IBM.WMQ;

class MQSample
{
  // The type of connection to use, this can be:-
  // MQC.TRANSPORT_MQSERIES_BINDINGS for a server connection.
  // MQC.TRANSPORT_MQSERIES_CLIENT for a non-XA client connection
  // MQC.TRANSPORT_MQSERIES_XACLIENT for an XA client connection
  // MQC.TRANSPORT_MQSERIES_MANAGED for a managed client connection
  const String connectionType = MQC.TRANSPORT_MQSERIES_CLIENT;

  // Define the name of the queue manager to use (applies to all connections)
  const String qManager = "your_Q_manager";

  // Define the name of your host connection (applies to client connections only)
  const String hostName = "your_hostname";

  // Define the name of the channel to use (applies to client connections only)
  const String channel = "your_channelname";



  /// <summary>
  /// Initialise the connection properties for the connection type requested
```

```
/// </summary>
/// <param name="connectionType">One of the MQC.TRANSPORT_MQSERIES_ values</param>
static Hashtable init(String connectionType)
{
  Hashtable connectionProperties = new Hashtable();

  // Add the connection type
  connectionProperties.Add(MQC.TRANSPORT_PROPERTY, connectionType);

  // Set up the rest of the connection properties, based on the
  // connection type requested
  switch(connectionType)
  {
    case MQC.TRANSPORT_MQSERIES_BINDINGS:
      break;
    case MQC.TRANSPORT_MQSERIES_CLIENT:
    case MQC.TRANSPORT_MQSERIES_XACLIENT:
    case MQC.TRANSPORT_MQSERIES_MANAGED:
      connectionProperties.Add(MQC.HOST_NAME_PROPERTY, hostName);
      connectionProperties.Add(MQC.CHANNEL_PROPERTY, channel);
      break;
  }

  return connectionProperties;
}
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static int Main(string[] args)
{
  try
  {
    Hashtable connectionProperties = init(connectionType);

    // Create a connection to the queue manager using the connection
    // properties just defined
    MQQueueManager qMgr = new MQQueueManager(qManager, connectionProperties);

    // Set up the options on the queue we want to open
    int openOptions = MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT;

    // Now specify the queue that we want to open,and the open options
    MQQueue system_default_local_queue =
      qMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", openOptions);

    // Define an IBM MQ message, writing some text in UTF format
    MQMessage hello_world = new MQMessage();
    hello_world.WriteUTF("Hello World!");

    // Specify the message options
    MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
                                                         // same as MQPMO_DEFAULT

    // Put the message on the queue
    system_default_local_queue.Put(hello_world, pmo);



    // Get the message back again

    // First define an IBM MQ message buffer to receive the message
    MQMessage retrievedMessage =new MQMessage();
    retrievedMessage.MessageId =hello_world.MessageId;

    // Set the get message options
    MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                        //same as MQGMO_DEFAULT

    // Get the message off the queue
    system_default_local_queue.Get(retrievedMessage,gmo);

    // Prove we have the message by displaying the UTF message text
    String msgText = retrievedMessage.ReadUTF();
    Console.WriteLine("The message is: {0}", msgText);

    // Close the queue
    system_default_local_queue.Close();

    // Disconnect from the queue manager
    qMgr.Disconnect();
  }
```

```
    //If an error has occurred,try to identify what went wrong.

    //Was it an IBM MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("An IBM MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

Configuration de l'environnement IBM MQ

Avant d'utiliser la connexion client pour vous connecter à un gestionnaire de files d'attente, vous devez configurer l'environnement IBM MQ .

Remarque : Cette étape n'est pas nécessaire lors de l'utilisation de IBM MQ classes for .NET en mode de liaisons de serveur.

L'interface de programmation .NET permet d'utiliser la valeur de personnalisation NMQ_MQ_LIB, mais inclut également une classe MQEnvironment. Cette classe vous permet de spécifier les détails à utiliser lors de la tentative de connexion, tels que ceux de la liste suivante:

- Nom du canal
- Nom d'hôte
- Numéro de port
- Exits de canal
- Paramètres SSL
- ID utilisateur et mot de passe

Pour plus d'informations sur la classe MQEnvironment, voir Classe MQEnvironment.NET

Pour spécifier le nom de canal et le nom d'hôte, utilisez le code suivant:

```
MQEnvironment.Hostname = "host.domain.com";
MQEnvironment.Channel  = "client.channel";
```

Par défaut, les clients tentent de se connecter à un programme d'écoute IBM MQ sur le port 1414. Pour spécifier un port différent, utilisez le code suivant:

```
MQEnvironment.Port = nnnn;
```

Connexion et déconnexion d'un gestionnaire de files d'attente

Une fois que vous avez configuré l'environnement IBM MQ , vous êtes prêt à vous connecter à un gestionnaire de files d'attente.

Pour vous connecter à un gestionnaire de files d'attente, créez une nouvelle instance de la classe MQQueueManager :

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

Pour vous déconnecter d'un gestionnaire de files d'attente, appelez la méthode `Disconnect` sur le gestionnaire de files d'attente:

```
queueManager.Disconnect();
```

Vous devez disposer du droit d'interrogation (`inq`) sur le gestionnaire de files d'attente lorsque vous tentez de vous connecter au gestionnaire de files d'attente. Sans droit d'interrogation, la tentative de connexion échoue.

Si vous appelez la méthode `Disconnect` , toutes les files d'attente ouvertes et tous les processus auxquels vous avez accédé via ce gestionnaire de files d'attente sont fermés. Toutefois, il est recommandé de fermer ces ressources de manière explicite lorsque vous avez fini de les utiliser. Pour fermer les ressources, utilisez la méthode `Close` sur l'objet associé à chaque ressource.

Les méthodes `Commit` et `Backout` d'un gestionnaire de files d'attente remplacent les appels MQCMIT et MQBACK utilisés avec l'interface de procédure.

Accès aux files d'attente et aux rubriques

Vous pouvez accéder aux files d'attente et aux rubriques à l'aide des méthodes de MQQueueManager ou des constructeurs appropriés.

Pour accéder aux files d'attente, utilisez les méthodes de la classe MQQueueManager . Le MQOD (structure de descripteur d'objet) est réduit dans les paramètres de ces méthodes. Par exemple, pour ouvrir une file d'attente sur un gestionnaire de files d'attente représenté par un objet MQQueueManager appelé queueManager, utilisez le code suivant:

```
MQQueue queue = queueManager.AccessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

Le paramètre *options* est identique au paramètre Options de l'appel MQOPEN.

La méthode AccessQueue renvoie un nouvel objet de la classe MQQueue.

Une fois que vous avez fini d'utiliser la file d'attente, utilisez la méthode Close () pour la fermer, comme dans l'exemple suivant:

```
queue.Close();
```

Avec IBM MQ .NET, vous pouvez également créer une file d'attente à l'aide du constructeur MQQueue. Les paramètres sont exactement les mêmes que pour la méthode accessQueue , avec l'ajout d'un paramètre de gestionnaire de files d'attente spécifiant l'objet MQQueueManager instancié à utiliser. Exemple :

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

La construction d'un objet de file d'attente de cette manière vous permet d'écrire vos propres sous-classes de MQQueue.

De même, vous pouvez également accéder aux rubriques à l'aide des méthodes de la classe MQQueueManager . Utilisez une méthode AccessTopic() pour ouvrir une rubrique. Retourne un nouvel objet de la classe MQTopic. Une fois que vous avez fini d'utiliser la rubrique, utilisez la méthode Close () de MQTopic pour la fermer.

Vous pouvez également créer une rubrique à l'aide d'un constructeur MQTopic. Il existe un certain nombre de constructeurs pour les rubriques ; pour plus d'informations, voir Classe `MQTopic`.NET.

Traitement des messages

Les messages sont traités à l'aide des méthodes des classes de file d'attente ou de rubrique. Pour générer un nouveau message, créez un nouvel objet MQMessageObject.

Insérez des messages dans des files d'attente ou des rubriques à l'aide de la méthode Put () de la classe MQQueue ou MQTopic. Extrayez les messages des files d'attente ou des rubriques à l'aide de la méthode Get () de la classe MQQueue ou MQTopic. Contrairement à l'interface de procédure, où MQPUT et MQGET placent et obtiennent des tableaux d'octets, IBM MQ classes for .NET place et obtient des instances de la classe MQMessage. La classe MQMessage encapsule la mémoire tampon de données qui contient les données de message réelles, ainsi que tous les paramètres MQMD (descripteur de message) qui décrivent ce message.

Pour générer un nouveau message, créez une nouvelle instance de la classe MQMessage et utilisez les méthodes WriteXXX pour placer des données dans la mémoire tampon de messages.

Lorsque la nouvelle instance de message est créée, tous les paramètres MQMD sont automatiquement définis sur leurs valeurs par défaut, comme défini dans Valeurs initiales et déclarations de langage pour MQMD. La méthode Put () de MQQueue utilise également une instance de la classe d'options MQPutMessagecomme paramètre. Cette classe représente la structure MQPMO. L'exemple suivant crée un message et le place dans une file d'attente:

```
// 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);
```

La méthode Get () de MQQueue renvoie une nouvelle instance de MQMessage, qui représente le message qui vient d'être extrait de la file d'attente. Il prend également une instance de la classe MQGetMessageOptions comme paramètre. Cette classe représente la structure MQGMO.

Vous n'avez pas besoin de spécifier une taille de message maximale, car la méthode Get () ajuste automatiquement la taille de sa mémoire tampon interne en fonction du message entrant. Utilisez les méthodes ReadXXX de la classe MQMessage pour accéder aux données du message renvoyé.

L'exemple suivant montre comment extraire un message d'une file d'attente:

```
// 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();
```

Vous pouvez modifier le format numérique utilisé par les méthodes de lecture et d'écriture en définissant la variable de membre *encoding* .

Vous pouvez modifier le jeu de caractères à utiliser pour la lecture et l'écriture de chaînes en définissant la variable de membre *characterSet* .

Voir Classe MQMessage.NET pour plus de détails.

Remarque : La méthode WriteUTF() de MQMessage code automatiquement la longueur de la chaîne ainsi que les octets Unicode qu'elle contient. Lorsque votre message sera lu par un autre programme .NET (à l'aide de ReadUTF()), il s'agit du moyen le plus simple d'envoyer des informations de chaîne.

Traitement des propriétés de message

Les propriétés de message vous permettent de sélectionner des messages ou d'extraire des informations sur un message sans accéder à ses en-têtes. La classe MQMessage contient des méthodes permettant d'obtenir et de définir des propriétés.

Vous pouvez utiliser les propriétés de message pour permettre à une application de sélectionner les messages à traiter ou d'extraire des informations sur un message sans accéder aux en-têtes MQMD ou MQRFH2 . Ils facilitent également la communication entre les applications IBM MQ et JMS . Pour plus d'informations sur les propriétés de message dans IBM MQ, voir Propriétés de message.

La classe MQMessage fournit un certain nombre de méthodes permettant d'obtenir et de définir des propriétés, en fonction du type de données de la propriété. Les méthodes get ont des noms de format Get * Property et les méthodes set ont des noms de format Set * Property, où l'astérisque (*) représente l'une des chaînes suivantes:

- Boolean
- Octet
- Octets
- Double
- Flottant
- Ent
- Int2
- Int4
- Int8
- Long
- Objet
- Court
- String

Par exemple, pour obtenir la propriété IBM MQ myproperty (chaîne de caractères), utilisez l'appel `message.GetStringProperty('myproperty')`. Vous pouvez éventuellement transmettre un descripteur de propriété, qui sera terminé par IBM MQ .

Traitement des erreurs

Gérez les erreurs provenant de IBM MQ classes for .NET à l'aide de blocs `try` et `catch` .

Les méthodes de l'interface .NET ne renvoient pas de code achèvement ni de code anomalie. Au lieu de cela, ils émettent une exception chaque fois que le code achèvement et le code anomalie résultant d'un appel IBM MQ ne sont pas tous les deux nuls. Cela simplifie la logique du programme de sorte que vous n'ayez pas à vérifier les codes retour après chaque appel à IBM MQ. Vous pouvez décider à quels points de votre programme vous souhaitez faire face à la possibilité d'un échec. A ces points, vous pouvez entourer votre code de blocs `try` et `catch` , comme dans l'exemple suivant:

```
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 );
}
```

Obtention et définition des valeurs d'attribut

Les classes MQManagedObject, MQQueue et MQQueueManager contiennent des méthodes qui permettent d'obtenir et de définir leurs valeurs d'attribut. Notez que pour MQQueue, les méthodes ne fonctionnent que si vous spécifiez les indicateurs d'interrogation et de définition appropriés lorsque vous ouvrez la file d'attente.

Pour les attributs communs, les classes MQQueueManager et MQQueue héritent d'une classe appelée MQManagedObject. Cette classe définit les interfaces Inquire () et Set ().

Lorsque vous créez un objet de gestionnaire de files d'attente à l'aide de l'opérateur *new* , il est automatiquement ouvert pour l'interrogation. Lorsque vous utilisez la méthode AccessQueue() pour accéder à un objet file d'attente, cet objet n'est pas automatiquement ouvert pour les opérations d'interrogation ou de définition, ce qui peut entraîner des problèmes avec certains types de files d'attente éloignées. Pour utiliser les méthodes Inquire et Set et pour définir les propriétés d'une file d'attente, vous devez spécifier les indicateurs d'interrogation et de définition appropriés dans le paramètre openOptions de la méthode AccessQueue().

Les méthodes d'interrogation et de définition prennent trois paramètres:

- tableau de sélecteurs
- Tableau intAttrs
- Tableau charAttrs

Vous n'avez pas besoin des paramètres SelectorCount, IntAttrCount et CharAttrLength qui se trouvent dans MQINQ, car la longueur d'un tableau est toujours connue. L'exemple suivant montre comment effectuer une interrogation sur une file d'attente:

```
//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);
```

Tous les attributs de ces objets peuvent être recherchés. Un sous-ensemble d'attributs est exposé en tant que propriétés d'un objet. Pour obtenir la liste des attributs d'objet, voir Attributs des objets. Pour les propriétés d'objet, voir la description de classe appropriée.

Programmes à unités d'exécution multiples

L'environnement d'exécution .NET est intrinsèquement à unités d'exécution multiples. IBM MQ classes for .NET permet de partager un objet de gestionnaire de files d'attente entre plusieurs unités d'exécution, mais garantit que tous les accès au gestionnaire de files d'attente cible sont synchronisés.

Prenons l'exemple d'un programme simple qui se connecte à un gestionnaire de files d'attente et ouvre une file d'attente au démarrage. Le programme affiche un seul bouton à l'écran. Lorsqu'un utilisateur clique sur ce bouton, le programme extrait un message de la file d'attente. Dans cette situation, l'initialisation de l'application se produit dans une unité d'exécution et le code qui s'exécute en réponse à la pression du bouton s'exécute dans une unité d'exécution distincte (l'unité d'exécution de l'interface utilisateur).

L'implémentation de IBM MQ .NET garantit que, pour une connexion particulière (instance d'objetMQQueueManager), tous les accès au gestionnaire de files d'attente IBM MQ cible sont synchronisés. Par défaut, une unité d'exécution qui souhaite émettre un appel à un gestionnaire de files d'attente est bloquée jusqu'à ce que tous les autres appels en cours pour cette connexion soient terminés. Si vous avez besoin d'un accès simultané au même gestionnaire de files d'attente à partir de plusieurs unités d'exécution de votre programme, créez un nouvel objet MQQueueManager pour chaque

unité d'exécution nécessitant un accès simultané. (Cela revient à émettre un appel MQCONN distinct pour chaque unité d'exécution.)

Si les options de connexion par défaut sont remplacées par MQC.MQCNO_HANDLE_SHARE_NONE ou MQC.MQCNO_SHARE_NO_BLOCK , le gestionnaire de files d'attente n'est plus synchronisé.

Utilisation d'une table de définition de canal du client avec .NET

Vous pouvez utiliser une table de définition de canal du client (CCDT) avec IBM MQ classes for .NET. Vous spécifiez l'emplacement de la table de définition de canal du client de différentes manières, selon que vous utilisez une connexion gérée ou non.

Type de connexion client non gérée XA ou XA

Avec un type de connexion non gérée, vous pouvez spécifier l'emplacement de la table de définition de canal du client de deux manières:

- Utilisation des variables d'environnement MQCHLLIB pour spécifier le répertoire dans lequel se trouve la table et de MQCHLTAB pour spécifier le nom de fichier de la table.
- Utilisation du fichier de configuration du client. Dans la strophe CHANNELS, utilisez les attributs **ChannelDefinitionDirectory** pour spécifier le répertoire dans lequel se trouve la table et **ChannelDefinitionFile** pour spécifier le nom de fichier.

Si l'emplacement est spécifié à la fois dans le fichier de configuration du client et à l'aide de variables d'environnement, les variables d'environnement sont prioritaires. Vous pouvez utiliser cette fonction pour spécifier un emplacement standard dans le fichier de configuration du client et le remplacer à l'aide de variables d'environnement si nécessaire.

Type de connexion du client géré

Avec un type de connexion gérée, vous pouvez spécifier l'emplacement de la table de définition de canal du client de trois manières:

- Utilisation du fichier de configuration de l'application .NET . Dans la section CHANNELS, utilisez les clés **ChannelDefinitionDirectory** pour spécifier le répertoire dans lequel se trouve la table et **ChannelDefinitionFile** pour spécifier le nom de fichier.
- Utilisation des variables d'environnement MQCHLLIB pour spécifier le répertoire dans lequel se trouve la table et de MQCHLTAB pour spécifier le nom de fichier de la table.
- Utilisation du fichier de configuration du client. Dans la strophe CHANNELS, utilisez les attributs **ChannelDefinitionDirectory** pour spécifier le répertoire dans lequel se trouve la table et **ChannelDefinitionFile** pour spécifier le nom de fichier.

Si l'emplacement est spécifié de plusieurs manières, les variables d'environnement sont prioritaires sur le fichier de configuration du client et le fichier de configuration de l'application .NET est prioritaire sur les deux autres méthodes. Vous pouvez utiliser cette fonction pour spécifier un emplacement standard dans le fichier de configuration du client et le remplacer à l'aide de variables d'environnement ou du fichier de configuration de l'application si nécessaire.

Depuis la IBM MQ 9.3.0, le client .NET se comporte de la même manière que les clients C et Java et renvoie MQRC_Q_MGR_NAME_ERROR lors de l'utilisation d'une table de définition de canal du client avec le regroupement de gestionnaires de files d'attente.

Comment une application .NET détermine la définition de canal à utiliser

Dans l'environnement client IBM MQ .NET , la définition de canal à utiliser peut être spécifiée de différentes manières. Plusieurs spécifications de la définition de canal peuvent exister. Une application dérive la définition de canal à partir d'une ou de plusieurs sources.

S'il existe plusieurs définitions de canal, celle utilisée est sélectionnée dans l'ordre de priorité suivant:

1. Propriétés spécifiées dans le constructeur MQQueueManager , explicitement ou en incluant *MQC.CHANNEL_PROPERTY* dans la table de hachage des propriétés
2. Propriété *MQC.CHANNEL_PROPERTY* dans la table de hachage MQEnvironment.properties
3. La propriété *Channel* dans MQEnvironment
4. Le fichier de configuration de l'application .NET , le nom de section CHANNELS, la clé ServerConnectionParms (s'applique uniquement aux connexions gérées)
5. Variable d'environnement *MQSERVER*
6. Le fichier de configuration client, strophe CHANNELS, attribut ServerConnectionParms
7. Table de définition de canal du client (CCDT). L'emplacement de la table de définition de canal du client est spécifié dans le fichier de configuration de l'application .NET (s'applique uniquement aux connexions gérées)
8. Table de définition de canal du client (CCDT). L'emplacement de la table de définition de canal du client est spécifié à l'aide des variables d'environnement *MQCHLIB* et *MQCHLTAB*
9. Table de définition de canal du client (CCDT). L'emplacement de la table de définition de canal du client est spécifié à l'aide du fichier de configuration du client

Pour les articles 1 à 3, la définition de canal est créée zone par zone à partir des valeurs fournies par l'application. Ces valeurs peuvent être fournies à l'aide d'interfaces différentes et plusieurs valeurs peuvent exister pour chacune d'elles. Les valeurs de zone sont ajoutées à la définition de canal suivant l'ordre de priorité donné:

1. Valeur de *connName* sur le constructeur MQQueueManager
2. Valeurs des propriétés de la table de hachage MQQueueManager.properties
3. Valeurs des propriétés de la table de hachage MQEnvironment.properties
4. Valeurs définies en tant que zones MQEnvironment (par exemple, MQEnvironment.Hostname, MQEnvironment.Port)

Pour les éléments 4 à 6, la définition de canal complète est fournie en tant que valeur. Les zones non spécifiées dans la définition de canal prennent les valeurs par défaut du système. Aucune valeur des autres méthodes de définition des canaux et de leurs champs n'est fusionnée avec ces spécifications.

Pour les articles 7 à 9, la définition de canal complète est extraite de la table de définition de canal du client. Les zones qui n'ont pas été spécifiées explicitement lors de la définition du canal prennent les valeurs par défaut du système. Aucune valeur des autres méthodes de définition des canaux et de leurs champs n'est fusionnée avec ces spécifications.

Utilisation des exits de canal dans IBM MQ .NET

Si vous utilisez des liaisons client, vous pouvez utiliser des exits de canal comme pour toute autre connexion client. Si vous utilisez des liaisons gérées, vous devez écrire un programme d'exit qui implémente une interface appropriée.

Liaisons client

Si vous utilisez des liaisons client, vous pouvez utiliser des exits de canal comme décrit dans exits de canal. Vous ne pouvez pas utiliser les exits de canal écrits pour les liaisons gérées.

Liaisons gérées

Si vous utilisez une connexion gérée, pour implémenter un exit, vous définissez une nouvelle classe .NET qui implémente l'interface appropriée. Trois interfaces d'exit sont définies dans le package IBM MQ :

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Remarque : Les exits utilisateur écrits à l'aide de ces interfaces ne sont pas pris en charge en tant qu'exits de canal dans l'environnement non géré.

L'exemple suivant définit une classe qui implémente les trois:

```
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
  }

}
```

Chaque exit reçoit un MQChannelExit et une instance d'objet MQChannelDefinition . Ces objets représentent les structures MQCXP et MQCD définies dans l'interface de procédure.

Les données à envoyer par un exit d'émission et les données reçues dans un exit de sécurité ou de réception sont spécifiées à l'aide des paramètres de l'exit.

A l'entrée, les données au décalage *dataOffset* avec la longueur *dataLength* dans le tableau d'octets *dataBuffer* sont les données qui sont sur le point d'être envoyées par un exit d'émission et les données reçues dans un exit de sécurité ou de réception. Le paramètre *dataMaxLength* donne la longueur maximale (à partir de *dataOffset*) disponible pour l'exit dans *dataBuffer*. Remarque: pour un exit de sécurité, il est possible que la valeur de dataBuffer soit null, s'il s'agit de la première fois que l'exit est appelé ou que l'extrémité partenaire est sélectionnée pour ne pas envoyer de données.

En cas de retour, la valeur de *dataOffset* et *dataLength* doit être définie pour pointer vers le décalage et la longueur dans le tableau d'octets renvoyé que les classes .NET doivent ensuite utiliser. Pour un exit d'émission, indique les données qu'il doit envoyer, et pour un exit de sécurité ou de réception, les données qui doivent être interprétées. L'exit doit normalement renvoyer un tableau d'octets ; les exceptions sont un exit de sécurité qui peut choisir de ne pas envoyer de données et tout exit appelé avec les raisons INIT ou TERM. La forme d'exit la plus simple qui peut être écrite est donc celle qui ne fait rien de plus que renvoyer dataBuffer:

Le corps de sortie le plus simple possible est:

```
{
  return dataBuffer;
}
```

Classe MQChannelDefinition

L'ID utilisateur et le mot de passe spécifiés avec l'application client .NET gérée sont définis dans la classe IBM MQ .NET MQChannelDefinition qui est transmise à l'exit de sécurité client. L'exit de sécurité copie l'ID utilisateur et le mot de passe dans MQCD.RemoteUserIdentifier et MQCD.RemotePassword (voir «Ecriture d'un exit de sécurité», à la page 1001).

Spécification des exits de canal (client géré)

Si vous spécifiez un nom de canal et un nom de connexion lors de la création de l'objet MQQueueManager (dans l'environnement MQEnvironment ou dans le constructeur MQQueueManager), vous pouvez spécifier les exits de canal de deux manières.

Par ordre de priorité, il s'agit des éléments suivants:

1. Transmission des propriétés de table de hachage MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY ou MQC.RECEIVE_EXIT_PROPERTY sur le constructeur MQQueueManager .
2. Définition des propriétés SecurityExitde MQEnvironment, SendExit ou ReceiveExit .

Si vous n'indiquez pas de nom de canal et de nom de connexion, les exits de canal à utiliser proviennent de la définition de canal sélectionnée dans une table de définition de canal du client (CCDT). Il n'est pas possible de remplacer les valeurs stockées dans la définition de canal. Voir Table de définition de canal du client et «Utilisation d'une table de définition de canal du client avec .NET», à la page 605 pour plus d'informations sur les tables de définition de canal.

Dans chaque cas, la spécification prend la forme d'une chaîne au format suivant:

```
Assembly_name(Class_name)
```

nom_classe est le nom qualifié complet, y compris la spécification de l'espace de nom, d'une classe .NET qui implémente IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit ou IBM.WMQ.MQReceiveExit (selon le cas). *nom_assemblage* est l'emplacement qualifié complet, y compris l'extension de fichier, de l'assemblage qui héberge la classe. La longueur de la chaîne est limitée à 999 caractères si vous utilisez les propriétés de MQEnvironment ou MQQueueManager. Toutefois, si le nom de l'exit de canal est spécifié dans la table de définition de canal du client, il est limité à 128 caractères. Si nécessaire, le code client .NET charge et crée une instance de la classe spécifiée en analysant la spécification de chaîne.

Spécification des données utilisateur d'exit de canal (client géré)

Les exits de canal peuvent être associés à des données utilisateur. Si vous spécifiez un nom de canal et un nom de connexion lors de la création de l'objet MQQueueManager (dans l'environnement MQEnvironment ou dans le constructeur MQQueueManager), vous pouvez spécifier les données utilisateur de deux manières.

Par ordre de priorité, il s'agit des éléments suivants:

1. Transmission des propriétés de table de hachage MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY ou MQC.RECEIVE_USERDATA_PROPERTY sur le constructeur MQQueueManager .
2. Définition des propriétés de données SecurityUserde MQEnvironment, SendUserData ou ReceiveUserData.

Si vous n'indiquez pas de nom de canal et de nom de connexion, les valeurs de données utilisateur d'exit à utiliser proviennent de la définition de canal sélectionnée dans la table de définition de canal du client (CCDT). Il n'est pas possible de remplacer les valeurs stockées dans la définition de canal. Voir Table de définition de canal du client et «Utilisation d'une table de définition de canal du client avec .NET», à la page 605 pour plus d'informations sur les tables de définition de canal.

Dans chaque cas, la spécification est une chaîne, limitée à 32 caractères.

Reconnexion automatique du client dans .NET

Vous pouvez faire en sorte que votre client se reconnecte automatiquement à un gestionnaire de files d'attente lors d'une interruption de connexion inattendue.

Un client peut être déconnecté de manière inattendue d'un gestionnaire de files d'attente si, par exemple, le gestionnaire de files d'attente s'arrête ou si le réseau ou le serveur est défaillant.

Sans reconnexion automatique du client, une erreur est générée lorsque la connexion échoue. Vous pouvez utiliser le code d'erreur pour vous aider à rétablir la connexion.

Un client qui utilise la fonction de reconnexion automatique du client est appelé client reconnectable. Pour créer un client reconnectable, spécifiez certaines options appelées options de reconnexion lors de la connexion au gestionnaire de files d'attente.

Si l'application client est un client IBM MQ .NET , elle peut choisir d'obtenir une reconnexion client automatique en spécifiant une valeur appropriée pour CONNECT_OPTIONS_PROPERTY lorsque vous utilisez la classe `MQQueueManager` pour créer un gestionnaire de files d'attente. Voir Options de reconnexion pour plus de détails sur les valeurs de CONNECT_OPTIONS_PROPERTY.

Vous pouvez indiquer si l'application client se connecte et se reconnecte toujours à un gestionnaire de files d'attente du même nom, au même gestionnaire de files d'attente ou à un ensemble de gestionnaires de files d'attente définis avec le même `QMNAME` dans la table des connexions client (voir Groupes de gestionnaires de files d'attente dans CCDT pour plus de détails).

Prise en charge de TLS (Transport Layer Security) pour .NET

Les applications client IBM MQ classes for .NET prennent en charge le chiffrement TLS (Transport Layer Security). Le protocole TLS assure la sécurité des communications sur Internet et permet aux applications client/serveur de communiquer de manière confidentielle et fiable.

Concepts associés

Prise en charge de TLS par le client géré IBM MQ.NET

Protocole de sécurité cryptographique TLS

Prise en charge de TLS pour le client .NET non géré

La prise en charge de TLS pour le client .NET non géré est basée sur C MQI et IBM Global Security Kit (GSKit). L'interface MQI C gère les opérations TLS et GSKit implémente les protocoles de socket sécurisé TLS.

Activation de TLS pour le client .NET non géré

TLS est pris en charge uniquement pour les connexions client. Pour activer TLS, vous devez spécifier le CipherSpec à utiliser lors de la communication avec le gestionnaire de files d'attente, qui doit correspondre au CipherSpec défini sur le canal cible.

Pour activer TLS, spécifiez le CipherSpec à l'aide de la variable de membre statique SSLCipherSpec de MQEnvironment. L'exemple suivant se connecte à un canal SVRCONN nommé SECURE.SVRCONN.CHANNEL, qui a été configuré pour exiger TLS avec une CipherSpec de TLS_RSA_WITH_AES_128_CBC_SHA:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key.kdb";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Voir Spécification de CipherSpecs pour obtenir la liste des CipherSpecs.

La propriété SSLCipherSpec peut également être définie à l'aide de la propriété MQC.SSL_CIPHER_SPEC_PROPERTY dans la table de hachage des propriétés de connexion.

Pour que la connexion à l'aide de TLS aboutisse, le magasin de clés du client doit être configuré avec la chaîne de certificats racine de l'autorité de certification à partir de laquelle le certificat présenté par le gestionnaire de files d'attente peut être authentifié. De même, si SSLClientAuth sur le canal SVRCONN a

été défini sur MQSSL_CLIENT_AUTH_REQUIRED, le magasin de clés du client doit contenir un certificat personnel d'identification approuvé par le gestionnaire de files d'attente.

Utilisation du nom distinctif du gestionnaire de files d'attente
Le gestionnaire de files d'attente s'identifie à l'aide d'un certificat TLS, qui contient un *nom distinctif* (DN).

Une application client IBM MQ .NET peut utiliser ce nom distinctif pour s'assurer qu'elle communique avec le gestionnaire de files d'attente approprié. Un modèle de nom distinctif est spécifié à l'aide de la variable de nom sslPeerde MQEnvironment. Par exemple, en définissant:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

permet à la connexion d'aboutir uniquement si le gestionnaire de files d'attente présente un certificat dont le nom usuel commence par QMGR., et au moins deux noms d'unité organisationnelle, dont le premier doit être IBM et le second WEBSPHERE.

La propriété SSLPeerName peut également être définie à l'aide de la propriété MQC.SSL_PEER_NAME_PROPERTY dans la table de hachage des propriétés de connexion. Pour plus d'informations sur les noms distinctifs et les règles de définition des noms d'homologue, voir Sécurisation de IBM MQ.

Si SSLPeerName est défini, les connexions aboutissent uniquement si un modèle valide est défini et que le gestionnaire de files d'attente présente un certificat correspondant.

Traitement des erreurs lors de l'utilisation de TLS

Les codes anomalie suivants peuvent être émis par IBM MQ classes for .NET lors de la connexion à un gestionnaire de files d'attente à l'aide de TLS:

MQRC_SSL_NOT_ALLOWED
La propriété SSLCipherSpec a été définie, mais la connexion des liaisons a été utilisée. Seule la connexion client prend en charge TLS.

MQRC_SSL_PEER_NAME_MISMATCH
Le modèle de nom distinctif spécifié dans la propriété SSLPeerName ne correspond pas au nom distinctif présenté par le gestionnaire de files d'attente.

MQRC_SSL_PEER_NAME_ERROR
Le modèle de nom distinctif spécifié dans la propriété SSLPeerName n'est pas valide.

MQRC_KEY_REPOSITORY_ERROR
L'emplacement du référentiel de clés n'est pas spécifié, n'est pas valide ou est inaccessible.

Prise en charge de TLS pour le client .NET géré

Le client .NET géré utilise les bibliothèques Microsoft .NET Framework pour implémenter les protocoles de socket sécurisé TLS. La classe Microsoft System.Net.SecuritySslStream agit comme un flux via des sockets TCP connectés et envoie et reçoit des données via cette connexion socket.

Le niveau .NET Framework minimum requis est .NET Framework v3.5. Le niveau de prise en charge de l'algorithme de chiffrement est basé sur le niveau .NET Framework utilisé par l'application:

- Pour les applications basées sur les .NET Framework niveaux 3.5 et 4.0, les protocoles de socket sécurisé disponibles sont SSL 3.0 et TSL 1.0.
- Pour les applications basées sur .NET Framework niveau 4.5, les protocoles de socket sécurisé disponibles sont SSL 3.0, TLS 1.1 et TLS 1.2.

Il peut être nécessaire de déplacer des applications qui attendent une prise en charge plus élevée du protocole TLS vers une version plus récente de l'infrastructure, comme défini pour la prise en charge de la sécurité Microsoft dans le .NET Framework.

Les principales fonctions de la prise en charge de TLS pour le client .NET géré sont les suivantes:

Prise en charge du protocole TLS
La prise en charge de TLS pour le client géré .NET est définie via la classe SSLStream .NET et dépend de l'infrastructure .NET que l'application utilise. Pour plus d'informations, voir «Prise en charge du protocole TLS pour le client .NET géré», à la page 612.

Prise En Charge CipherSpec
Les paramètres TLS du client géré .NET sont ceux des steams TLS Microsoft.NET . Pour plus d'informations, voir «Prise en charge de CipherSpec pour le client .NET géré», à la page 613 et «CipherSpec mappages pour le client .NET géré», à la page 614.

Référentiels de clés
Le référentiel de clés côté client est un magasin de clés Windows . Le référentiel côté serveur est un référentiel de typeCMS(Cryptographic Message Syntax). Pour plus d'informations, voir «Référentiels de clés pour le client .NET géré», à la page 616.

Certificats
Vous pouvez utiliser des certificats TLS autosignés pour implémenter l'authentification mutuelle entre un client et un gestionnaire de files d'attente. Pour plus d'informations, voir «Utilisation de certificats pour le client géré .NET», à la page 616.

SSLPEERNAME
Dans .NET, les applications peuvent utiliser l'attribut SSLPEERNAME facultatif pour spécifier un modèle de nom distinctif (DN). Pour plus d'informations, voir «SSLPEERNAME», à la page 617.

Conformité à la norme FIPS
L'activation de la norme FIPS à l'aide d'un programme n'est pas prise en charge par la bibliothèque de sécurité Microsoft.NET . L'activation de la norme FIPS est contrôlée par le paramètre de stratégie de groupe Windows .

Conformité NSA Suite B
IBM MQ implémente RFC 6460. L'implémentation Microsoft.NET pour la suite NSA B est 5430. Cette fonction est prise en charge à partir de la version .NET Framework 3.5 .

Réinitialisation ou renégociation de la clé secrète
Bien que la classe SSLStream ne prenne pas en charge la réinitialisation ou la renégociation des clés secrètes, pour des raisons de cohérence avec les autres clients IBM MQ , le client géré .NET permet aux applications de définir `SSLKeyResetCount`. Pour plus d'informations, voir «Réinitialisation ou renégociation de clé secrète pour le client .NET géré», à la page 618.

Vérification de révocation
La classe SSLStream prend en charge la vérification de la révocation de certificat, qui est automatiquement effectuée par le moteur de chaînage de certificats. Pour plus d'informations, voir «Vérification de révocation», à la page 618.

Prise en charge des exits de sécurité IBM MQ
La classe `SSLStream` fournit une prise en charge limitée pour les exits de sécurité IBM MQ . L'interrogation des certificats locaux et distants pour obtenir `SSLPeerNamePtr`(nom distinctif du sujet) et `SSLRemCertIssNamePtr` (nom distinctif de l'émetteur) est possible car elle est prise en charge dans Microsoft.NET. Toutefois, il n'est pas possible d'obtenir des attributs tels que DNQ, UNSTRUCTUREDNAME et UNSTRUCTUREDADDRESS, de sorte que ces valeurs ne peuvent pas être extraites à l'aide des exits.

Prise en charge du matériel de cryptographie
Le matériel de cryptographie n'est pas pris en charge pour le client .NET géré.

Prise en charge de TLS1.3 sur les clients IBM MQ .NET et XMS .NET gérés

V 9.4.0

Depuis IBM MQ 9.4.0, les clients IBM MQ .NET et XMS .NET prennent en charge TLS1.3 à condition que le système d'exploitation prenne en charge TLS1.3.

Le client .NET géré utilise les bibliothèques Microsoft .NET Framework pour implémenter les protocoles de socket sécurisé TLS. La classe Microsoft `System.Net.SecuritySslStream` fonctionne comme un flux sur des sockets TCP connectés et envoie et reçoit des données sur cette connexion socket.

Sous Windows, .NET utilise SCHANNEL et sous Linux , .NET utilise OpenSSL pour la communication SSL.

Windows Pour les applications client IBM MQ .NET s'exécutant sous Windows

Microsoft a annoncé que Windows 11 et Windows Server 2022 prennent en charge les chiffrements TSL1.3 par défaut.

Les suites de chiffrement TLS_AES_128_GCM_SHA256 et TLS_AES_256_GCM_SHA384 sont activées par défaut sur les deux versions de Windows.

Avertissement :

- TLS_CHACHA20_POLY1305_SHA256 La suite de chiffrement n'est pas activée par défaut, mais elle est prise en charge.
- Pour un client IBM MQ .NET avec TLS1.3 activé, pour que la connexion à un gestionnaire de files d'attente aboutisse, IBM Global Security Kit (GSKit) 8.0.55.29 est la version minimale requise côté gestionnaire de files d'attente.

Linux Pour les applications client IBM MQ .NET s'exécutant sous Linux

Comme .NET utilise OpenSSL sur Linux for SSL Communication, pour utiliser TLS1.3, OpenSSL v1.1.1 est la configuration minimale requise.

De plus, comme .NET utilise OpenSSL sous Linux, tous les chiffrements pris en charge par OpenSSL doivent également fonctionner pour .NET .

OpenSSL prend en charge les CipherSpecs suivantes pour TLS1.3:

- TLS_AES_256_GCM_SHA384
- TLS_CHACHA20_POLY1305_SHA256
- TLS_AES_128_GCM_SHA256
- TLS_AES_128_CCM_8_SHA256
- TLS_AES_128_CCM_SHA256

Concepts associés

«CipherSpec mappages pour le client .NET géré», à la page 614
L'interface IBM MQ.NET gère une table de mappage IBM MQ vers Microsoft.NET qui est utilisée pour déterminer la version du protocole TLS que le client géré doit utiliser pour établir une connexion sécurisée avec un gestionnaire de files d'attente.

Prise en charge du protocole TLS pour le client .NET géré
La prise en charge de TLS dans IBM MQ.NET est basée sur la classe SSLStream .NET .

Remarque : La prise en charge du protocole TLS pour le client .NET géré dépend du niveau .NET Framework utilisé par l'application. Pour plus d'informations, voir «Prise en charge de TLS pour le client .NET géré», à la page 610.

Pour que la classe SSLStream Microsoft.NET initialise TLS et effectue un établissement manuel avec le gestionnaire de files d'attente, l'un des paramètres requis que vous devez définir est **SSLProtocol**, où vous devez spécifier le numéro de version TLS, qui doit être l'une des valeurs suivantes:

- SSL3.0
- TLS1.0
- TLS1.2

La valeur de ce paramètre est étroitement liée à la famille de protocoles à laquelle appartient le CipherSpec préféré. Lorsque SSLStream démarre un établissement de liaison TLS avec le serveur (gestionnaire de files d'attente), il utilise la version TLS spécifiée dans **SSLProtocol** pour identifier la liste des CipherSpecs à utiliser pour la négociation.

IBM MQ.NET ne met aucune propriété à la disposition des applications pour qu'elles l'utilisent pour définir cette valeur. A la place, IBM MQ utilise une table de mappage pour mapper en interne l'ensemble CipherSpec à la famille de protocoles et identifie la version SSLProtocol à utiliser. Ce tableau montre le mappage de chaque CipherSpec pris en charge entre Microsoft.NET et IBM MQ, ainsi que la version de protocole à laquelle ils appartiennent. Pour plus d'informations, voir «CipherSpec mappages pour le client .NET géré», à la page 614.

Prise en charge de CipherSpec pour le client .NET géré
Les paramètres CipherSpec d'une application sont utilisés lors de l'établissement de liaison avec le serveur.

Les clients IBM MQ vous permettent de définir une valeur CipherSpec qui est utilisée lors de l'établissement de liaison avec le gestionnaire de files d'attente. Les clients IBM MQ doivent définir un CipherSpec valide pour la connexion sécurisée à établir, de préférence le CipherSpec spécifié dans la stratégie de groupe Windows . Si vous ne renseignez pas cette zone, cela indique un canal en texte en clair sans aucune sécurité sur les sockets.

Pour le client géré IBM MQ.NET , les paramètres TLS concernent la classe Microsoft.NET SSLStream. Pour SSLStream, un CipherSpecou une liste de préférences de CipherSpecspeut être défini uniquement dans la règle de groupe Windows , qui est un paramètre à l'échelle de l'ordinateur. SSLStream utilise ensuite la CipherSpec spécifiée ou la liste de préférences lors de l'établissement de liaison avec le serveur. Dans le cas d'autres clients IBM MQ , la propriété CipherSpec peut être définie dans l'application sur la définition de canal IBM MQ et le même paramètre est utilisé pour la négociation TLS. En raison de cette restriction, l'établissement de liaison TLS peut négocier n'importe quel CipherSpec pris en charge, quel que soit ce qui est spécifié dans la configuration de canal IBM MQ . Par conséquent, il est probable que l'erreur AMQ9631 se produise sur le gestionnaire de files d'attente. Pour éviter cette erreur, définissez le même CipherSpec que celui que vous avez défini dans l'application en tant que configuration TLS dans la stratégie de groupe Windows .

Le nouveau code client TLS IBM MQ.NET vérifie uniquement que la version de protocole correcte a été négociée. La version du protocole TLS est dérivée du CipherSpec que l'application définit et est utilisée pour l'établissement de liaison TLS avec le serveur (gestionnaire de files d'attente). Par conséquent, il est nécessaire, en raison de sa conception, de définir le CipherSpec dans l'application client gérée par IBM MQ.NET . Si le CipherSpec défini par le client IBM MQ est autre que celui des protocoles SSL 3.0, TLS 1.0 et TLS 1.2 , le client IBM MQ géré .NET négocie par défaut avec l'un des chiffrements des protocoles SSL 3.0 ou TLS 1.0 et ne signale pas d'erreur.

Remarque : Si la valeur CipherSpec fournie par l'application n'est pas une valeur CipherSpec connue de IBM MQ, le client .NET géré IBM MQ l'ignore et négocie la connexion en fonction de la règle de groupe du système Windows .

Définition d'un CipherSpec

Il existe trois façons de définir un CipherSpec:

Classe .NET MQEnvironment
L'exemple suivant montre comment définir un CipherSpec avec la classe MQEnvironment.

```
MQEnvironment.SSLKeyRepository = "*USER";
MQEnvironment.ConnectionName = connectionName;
MQEnvironment.Channel = channelName;
MQEnvironment.properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA";
```

Propriété TLS CipherSpec
L'exemple suivant montre comment définir un CipherSpec en ajoutant un paramètre de table de hachage dans le constructeur MQQueueManager .

```
properties = new Hashtable();
properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
```

```
properties.Add(MQC.PORT_PROPERTY, port);
properties.Add(MQC.CHANNEL_PROPERTY, channelName);
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
queueManager = new MQQueueManager(queueManagerName, properties);
```

Windows politique de groupe

Lorsqu'une liste de suites de chiffrement est configurée via la console de gestion des règles de groupe Windows , la définition de canal SVRCONN doit spécifier un CipherSpeccorrespondant. Une valeur CipherSpec correspondante peut être une valeur générique telle que "ANY_TLS12_OR_HIGHER" ou une valeur spécifique mappée à la suite de chiffrement la plus élevée qui serait négociée à partir de la liste ordonnée. L'utilisation de valeurs CipherSpec génériques est recommandée pour une utilisation avec les clients .NET car elle évite d'avoir à modifier la configuration CipherSpec SVRCONN si l'ordre de la liste des clients change.

Utilisation de CCDT

IBM MQ.NET prend uniquement en charge les tables de définition de canal du client (fichiers .TAB) qui se trouvent sur un ordinateur local. Les fichiers CCDT existants dont la valeur CipherSpec est définie peuvent être utilisés pour les connexions IBM MQ.NET . Toutefois, la valeur CipherSpec définie sur le canal de connexion client détermine la version du protocole TLS et doit également correspondre à la valeur CipherSpec définie dans la règle de groupe Windows .

Concepts associés

«Configuration de l'environnement IBM MQ», à la page 600
Avant d'utiliser la connexion client pour vous connecter à un gestionnaire de files d'attente, vous devez configurer l'environnement IBM MQ .

«Prise en charge de TLS pour le client .NET géré», à la page 610
Le client .NET géré utilise les bibliothèques Microsoft .NET Framework pour implémenter les protocoles de socket sécurisé TLS. La classe Microsoft System.Net.SecuritySslStream agit comme un flux via des sockets TCP connectés et envoie et reçoit des données via cette connexion socket.

Tâches associées

Définition des spécifications CipherSpec

Référence associée

Classe .NET MQEnvironment

CipherSpec mappages pour le client .NET géré

L'interface IBM MQ.NET gère une table de mappage IBM MQ vers Microsoft.NET qui est utilisée pour déterminer la version du protocole TLS que le client géré doit utiliser pour établir une connexion sécurisée avec un gestionnaire de files d'attente.

Si un CipherSpec est spécifié sur le canal SVRCONN, une fois l'établissement de liaison TLS terminé, le gestionnaire de files d'attente tente de faire correspondre ce CipherSpec avec le CipherSpec négocié utilisé par l'application client. Si le gestionnaire de files d'attente ne trouve pas de CipherSpeccorrespondant, la communication échoue avec l'erreur AMQ9631.

L'interface IBM MQ.NET gère une table de mappage IBM MQ vers Microsoft.NET CipherSpec . Cette table permet de déterminer la version du protocole TLS que le client souhaite utiliser pour établir une connexion socket sécurisée avec le gestionnaire de files d'attente. En fonction de la valeur SSLCipherSpec , la version SSLProtocol peut être TLS 1.0ou TLS 1.2, selon la version de l'infrastructure Microsoft.NET que vous utilisez.

Veillez à fournir la valeur SSLCipherSpec correcte car la spécification d'une valeur incorrecte peut entraîner l'utilisation des protocoles SSL 3.0 ou TLS 1.0 .

Tableau 78. Table de mappage IBM MQ et Microsoft.NET

IBM MQ CipherSpec	Microsoft.NET CipherSpec	Version TLS
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA	TLS 1.0
TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS_RSA_WITH_3DES_EDE_CBC_SHA [1]	TLS 1.0
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS_RSA_WITH_AES_128_CBC_SHA256	TLS 1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_RSA_AES_128_CBC_SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_CBC_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_CBC_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384_P521	TLS 1.2
ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P256	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P384	TLS 1.2
ECDHE_ECDSA_AES_128_GCM_SHA256	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256_P521	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P384	TLS 1.2
ECDHE_ECDSA_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384_P521	TLS 1.2
V 9.4.0 TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3

Tableau 78. Table de mappage IBM MQ et Microsoft.NET (suite)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Version TLS**
V 9.4.0 TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3
V 9.4.0 TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3

Remarques :

1. Deprecated Ce CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA est obsolète. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.

Concepts associés

«Prise en charge de TLS pour le client .NET géré», à la page 610
Le client .NET géré utilise les bibliothèques Microsoft .NET Framework pour implémenter les protocoles de socket sécurisé TLS. La classe Microsoft System.Net.SecuritySslStream agit comme un flux via des sockets TCP connectés et envoie et reçoit des données via cette connexion socket.

Référentiels de clés pour le client .NET géré
Le référentiel de clés utilisé par les clients .NET gérés est le magasin de clés Windows . Les certificats et les clés privées doivent être disponibles dans le magasin de clés de l'utilisateur ou du système pour pouvoir être utilisés par l'application client à la fois pour l'identité et la confiance lors de l'établissement d'une liaison TLS.

Côté client

Dans l'application, vous pouvez définir l'une des valeurs suivantes pour le référentiel de clés:

- "*USER": IBM MQ.NET accède au magasin de certificats de l'utilisateur en cours pour extraire les certificats client.
- "*SYSTEM": IBM MQ.NET accède au compte de l'ordinateur local pour extraire les certificats.

Les certificats du client doivent être stockés dans le magasin de certificats My de l'utilisateur ou du compte d'ordinateur. Tous les certificats du serveur (CA) doivent être stockés dans le répertoire racine du magasin de certificats.

Remarque : Vous pouvez stocker plusieurs certificats dans un même fichier dans les formats suivants:

- Personal Information Exchange-PKCS #12 (.PFX, .P12)
- Cryptographic Message Syntax Standard-Certificats PKCS #7 (.P7B)
- Microsoft Magasin de certificats sérialisés (.SST)

Utilisation de certificats pour le client géré .NET
Pour les certificats client, le client IBM MQ géré .NET accède au magasin de clés Windows et charge tous les certificats du client qui sont mis en correspondance par un libellé de certificat ou par la chaîne.

Lors de la sélection d'un certificat à utiliser, le client IBM MQ géré .NET utilise toujours le premier certificat correspondant pour l'établissement de liaison SSLStream TLS.

Mise en correspondance des certificats par label de certificat

Si vous définissez le libellé de certificat, le client IBM MQ géré .NET effectue une recherche dans le magasin de certificats Windows avec le nom de libellé indiqué pour identifier le certificat client. Il charge tous les certificats correspondants et utilise le premier certificat de la liste. Il existe deux options pour définir le libellé de certificat:

- Le libellé de certificat peut être défini sur la classe MQEnvironment accédant à MQEnvironment.CertificateLabel.
- Le libellé de certificat peut également être défini dans une propriété de table de hachage, fournie en tant que paramètre d'entrée avec le constructeur MQQueueManager , comme illustré dans l'exemple suivant.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

Le nom ("CertificateLabel") et la valeur sont sensibles à la casse.

Mise en correspondance des certificats par chaîne

Si le libellé de certificat n'est pas défini, le certificat qui correspond à la chaîne "`ibmwebspheremq`" et l'utilisateur actuellement connecté (en minuscules) sont recherchés et utilisés.

Tâches associées

Connexion sécurisée d'un client à un gestionnaire de files d'attente

Référence associée

Classe .NET MQEnvironment

SSLPEERNAME
L'attribut SSLPEERNAME est utilisé pour vérifier le nom distinctif (DN) du certificat provenant du gestionnaire de files d'attente homologue.

Dans IBM MQ.NET, les applications peuvent utiliser SSLPEERNAME pour spécifier un modèle de nom distinctif, comme illustré dans l'exemple suivant.

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Comme pour les autres clients IBM MQ , SSLPEERNAME est un paramètre facultatif.

Si la valeur SSLPEERNAME n'est pas définie, le client géré IBM MQ.NET n'effectue aucune validation de certificat distant (serveur) et le client géré accepte simplement le certificat distant (/serveur) en l'état.

La manière dont vous définissez SSLPEERNAME dépend des offres de pile IBM MQ que vous utilisez.

IBM MQ classes for .NET

Il existe trois options:

1. Définissez MQEnvironment.SSLPeerName dans la classe MQEnvironment.
2. `MQEnvironment.properties.Add(MQC.SSL_PEER_NAME_PROPERTY, value)`
3. Utilisez le constructeur de gestionnaire de files d'attente `MQQueueManager (String queueManagerName, Hashtable properties)`. Indiquez SSLPEERNAME dans le fichier `Hashtable properties` comme pour l'option 2.

XMS .NET

Définissez le nom de l'homologue SSL dans la fabrique de connexions:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

Fonction WCF

Incluez le nom SslPeeren tant que zone séparée par des points-virgules dans l'URI.

Référence associée

Classe .NET MQEnvironment

Réinitialisation ou renégociation de clé secrète pour le client .NET géré
La classe SSLStream ne prend pas en charge la réinitialisation / renégociation des clés secrètes. Toutefois, pour être cohérent avec les autres clients IBM MQ , le client IBM MQ géré .NET permet aux applications de définir **SSLKeyResetCount**.

Lorsque la limite est atteinte, IBM MQ.NET se déconnecte du gestionnaire de files d'attente et l'application en est informée en tant qu'exception avec MQRC_CONNECTION_BROKEN comme code anomalie. Les applications peuvent choisir de traiter l'exception et de rétablir les connexions ou d'activer l'option MQCNO_RECONNECT pour que IBM MQ.NET se reconnecte automatiquement au gestionnaire de files d'attente.

L'activation de la fonction de reconnexion automatique du client signifie que, lorsque le nombre de réinitialisations de clé est atteint, toutes les connexions existantes sont interrompues et le client IBM MQ.NET recrée toutes les connexions à nouveau. Pour plus d'informations sur la reconnexion automatique du client, voir Reconnexion automatique du client.

Concepts associés

Réinitialisation des clés secrètes SSL et TLS

Vérification de révocation
La classe SSLStream prend en charge la vérification de la révocation de certificat.

La vérification de révocation est automatiquement effectuée par le moteur de chaînage de certificats. Cela s'applique à la fois au protocole OCSP (Online Certificate Status Protocol) et aux listes de révocation de certificat (CRL). La classe SSLStream utilise la révocation de certificat qui utilise uniquement le serveur spécifié dans le certificat, c'est-à-dire que le serveur est dicté par le certificat lui-même. Il est possible pour les extensions HTTP CDP et les demandes HTTP OCSP de passer par le serveur proxy HTTP.

La manière dont vous définissez la vérification de révocation dépend des offres de pile IBM MQ que vous utilisez.

IBM MQ.NET
La vérification de révocation peut être définie en accédant à la propriété **MQEnvironment.SSLCertRevocationCheck** dans le fichier de classe `MQEnvironment.cs` .

XMS .NET
La vérification de révocation peut être définie sur le contexte de la propriété de la fabrique de connexions, comme illustré dans l'exemple suivant.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

Fonction WCF
La vérification de révocation peut être définie sur l'URI à l'aide de la convention de dénomination suivante.

```
"SslCertRevocationCheck=true"
```

Configuration de TLS pour IBM MQ .NET géré
La configuration de TLS pour IBM MQ .NET géré consiste à créer les certificats de signataire, puis à configurer le côté serveur, le côté client et le programme d'application.

Pourquoi et quand exécuter cette tâche

Pour configurer TLS, vous devez d'abord créer les certificats de signataire appropriés. Les certificats de signataire peuvent être des certificats autosignés ou fournis par une autorité de certification. Bien que les certificats autosignés puissent être utilisés sur un système de développement, de test ou de préproduction, ne les utilisez pas sur un système de production. Sur un système de production,

utilisez les certificats que vous avez obtenus auprès d'une autorité de certification externe (CA) digne de confiance.

Procédure

1. Créez les certificats de signataire.
   a) Pour créer des certificats autosignés, utilisez les commandes **runmqakm** ou V 9.4.0 V 9.4.0 **runmqktool** .

      Pour plus d'informations, voir Creating a self-signed personal certificate on AIX, Linux, and Windows.
   b) Pour obtenir des certificats pour le gestionnaire de files d'attente et les clients auprès d'une autorité de certification, suivez les instructions de la rubrique Obtention de certificats personnels auprès d'une autorité de certification.
2. Configurez le côté serveur.
   a) Configurez TLS sur le gestionnaire de files d'attente, à l'aide de IBM Global Security Kit (GSKit), comme décrit dans Connexion sécurisée d'un client à un gestionnaire de files d'attente.
   b) Définissez les attributs TLS du canal SVRCONN:
      - Définissez **SSLCAUTH** sur REQUIRED ou OPTIONAL.
      - Définissez **SSLCIPH** sur un CipherSpecapproprié.

      Pour plus d'informations, voir «Activation de TLS pour le client .NET non géré», à la page 609.
3. Configurez le côté client.
   a) Importez les certificats client dans le magasin de certificats Windows (sous le compte utilisateur / ordinateur).

      IBM MQ .NET accède aux certificats client à partir du magasin de certificats Windows . Par conséquent, vous devez importer vos certificats dans le magasin de certificats Windows pour établir une connexion socket sécurisée à IBM MQ . Pour plus d'informations sur l'accès au magasin de clés Windows et l'importation des certificats côté client, voir Importation ou exportation de certificats et de clés privées.
   b) Indiquez CertificateLabel comme décrit dans la rubrique Connexion d'un client à un gestionnaire de files d'attente en toute sécurité.
   c) Si nécessaire, éditez la stratégie de groupe Windows pour définir le CipherSpec, puis, pour que les mises à jour de la stratégie de groupe Windows prennent effet, redémarrez l'ordinateur.
4. Configurez le programme d'application.
   a) Définissez l'environnement MQEnvironment ou la valeur SSLCipherSpec pour indiquer que la connexion est sécurisée.

      La valeur que vous spécifiez est utilisée pour identifier le protocole utilisé (TLS). L'ensemble CipherSpec doit être l'un des CipherSpecs de la version SSLProtocol prise en charge et il peut de préférence être identique à celui spécifié dans la stratégie de groupe Windows . (La version SSLProtocol prise en charge dépend de l'infrastructure .NET utilisée. La version SSLProtocol peut être TLS 1.0ou TLS 1.2, selon la version de l'infrastructure Microsoft .NET que vous utilisez.)

      Remarque : Si la valeur CipherSpec fournie par l'application n'est pas une valeur CipherSpec connue de IBM MQ, le client .NET géré IBM MQ l'ignore et négocie la connexion en fonction de la règle de groupe du système Windows .
   b) Définissez la propriété SSLKeyRepository sur "*SYSTEM" ou "*USER".
   c) Facultatif : Définissez SSLPEERNAME sur le nom distinctif (DN) du certificat serveur.
   d) Indiquez CertificateLabel comme décrit dans la rubrique Connexion d'un client à un gestionnaire de files d'attente en toute sécurité.
   e) Définissez d'autres paramètres facultatifs dont vous avez besoin, tels que KeyResetCount, CertificationRevocationCheck, et activez FIPS.

Exemples de définition du protocole TLS et du référentiel de clés TLS

Pour Base .NET, vous pouvez définir le protocole TLS et le référentiel de clés TLS via la classe MQEnvironment, comme illustré dans l'exemple suivant:

```
MQEnvironment.SSLCipherSpec = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "*USER";

MQEnvironment.properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Vous pouvez également définir le protocole TLS et le référentiel de clés TLS en fournissant une table de hachage dans le cadre du constructeur MQQueueManager , comme illustré dans l'exemple suivant.

```
Hashtable properties = new Hashtable();
properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, "TLS_RSA_WITH_AES_128_CBC_SHA256")
```

Que faire ensuite

Pour plus d'informations sur l'initiation au développement d'applications TLS gérées par IBM MQ .NET , voir«Ecriture d'une application simple», à la page 620.

Référence associée

Classe .NET MQEnvironment

KeyReset(MQLONG)

FIPS (Federal Information Processing Standards) pour AIX, Linux, and Windows

Ecriture d'une application simple
Conseils pour l'écriture d'une application TLS IBM MQ gérée .NET simple, incluant des exemples de définition des propriétés SSL pour les fabriques de connexions, la création d'une instance de gestionnaire de files d'attente, d'une connexion, d'une session et d'une destination, ainsi que l'envoi d'un message de test.

Avant de commencer

Vous devez d'abord configurer TLS pour les IBM MQ.NET gérées, comme décrit dans «Configuration de TLS pour IBM MQ .NET géré», à la page 618.

Pour la configuration de programme d'application dans le .NETde base, définissez les propriétés SSL à l'aide de la classe MQEnvironment ou en fournissant une table de hachage dans le cadre du constructeur MQQueueManager .

Pour la configuration de programme d'application dans XMS .NET, vous définissez les propriétés SSL dans le contexte de propriété des fabriques de connexions.

Procédure

1. Définissez les propriétés SSL pour les fabriques de connexions comme indiqué dans les exemples suivants.

   Exemple pour IBM MQ.NET

   ```
   properties = new Hashtable();
   properties.Add(MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES_MANAGED);
   properties.Add(MQC.HOST_NAME_PROPERTY, hostName);
   properties.Add(MQC.PORT_PROPERTY, port);
   properties.Add(MQC.CHANNEL_PROPERTY, channelName);
   properties.Add(MQC.SSL_CERT_STORE_PROPERTY, sslKeyRepository);
   properties.Add(MQC.SSL_CIPHER_SPEC_PROPERTY, cipherSpec);
   properties.Add(MQC.SSL_PEER_NAME_PROPERTY, sslPeerName);
   properties.Add(MQC.SSL_RESET_COUNT_PROPERTY, keyResetCount);
   ```

```
properties.Add("CertificateLabel", "ibmwebspheremq");
MQEnvironment.SSLCertRevocationCheck = sslCertRevocationCheck;
```

Exemple pour XMS .NET

```
cf.SetStringProperty(XMSC.WMQ_SSL_KEY_REPOSITORY, "sslKeyRepository");
cf.SetStringProperty(XMSC.WMQ_SSL_CIPHER_SPEC, cipherSpec);
cf.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, sslPeerName);
cf.SetIntProperty(XMSC.WMQ_SSL_KEY_RESETCOUNT, keyResetCount);
cf.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

2. Créez l'instance de gestionnaire de files d'attente, les connexions, la session et la destination, comme illustré dans les exemples suivants.

   Exemple pour MQ .NET

   ```
   queueManager = new MQQueueManager(queueManagerName, properties);
   Console.WriteLine("done");

   // accessing queue
   Console.Write("Accessing queue " + queueName + ".. ");
   queue = queueManager.AccessQueue(queueName, MQC.MQOO_OUTPUT +
   MQC.MQOO_FAIL_IF_QUIESCING);
   Console.WriteLine("done");
   ```

   Exemple pour XMS .NET

   ```
   connectionWMQ = cf.CreateConnection();
   // Create session
   sessionWMQ = connectionWMQ.CreateSession(false, AcknowledgeMode.AutoAcknowledge);

   // Create destination
   destination = sessionWMQ.CreateQueue(destinationName);

   // Create producer
   producer = sessionWMQ.CreateProducer(destination);
   ```

3. Envoyez un message comme illustré dans les exemples suivants.

   Exemple pour MQ .NET

   ```
   // creating a message object
   message = new MQMessage();
   message.WriteString(messageString);

   // putting messages continuously
   for (int i = 1; i <= numberOfMsgs; i++)
   {
   Console.Write("Message " + i + " <" + messageString + ">.. ");
   queue.Put(message);
   Console.WriteLine("put");
   }
   ```

   Exemple pour XMS .NET

   ```
   textMessage = sessionWMQ.CreateTextMessage();
   textMessage.Text = simpleMessage;
   producer.Send(textMessage);
   ```

4. Vérifiez la connexion TLS.

   Vérifiez le statut du canal pour vous assurer que la connexion TLS a été établie et qu'elle fonctionne correctement.

Configuration de la trace pour SSLStream

Pour capturer les événements de trace et les messages liés à la classe SSLStream, vous devez ajouter une section de configuration pour les diagnostics système au fichier de configuration d'application de votre application.

Pourquoi et quand exécuter cette tâche

Remarque :

Cette tâche s'applique à IBM MQ classes for .NET Framework uniquement. Le fichier de configuration d'application n'est pas pris en charge dans les bibliothèques IBM MQ classes for .NET (.NET Standard et .NET 6).

Si vous n'ajoutez pas de section de configuration pour les diagnostics système au fichier de configuration de l'application, le client IBM MQ géré .NET ne capturera pas d'événements, de traces ou de points de débogage liés à TLS et à la classe SSLStream.

Remarque : Le démarrage de la fonction de trace IBM MQ à l'aide de **strmqtrc** ne capture pas toutes les fonctions de trace TLS requises.

Procédure

1. Créez un fichier de configuration d'application (App.Config) pour votre projet d'application.
2. Ajoutez une section de configuration des diagnostics système, comme illustré dans l'exemple suivant.

```
<system.diagnostics>
           <sources>
                 <source name="System.Net" tracemode="includehex">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Sockets">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Cache">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Net.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
                 <source name="System.Security">
                       <listeners>
                             <add name="ExternalSourceTrace"/>
                       </listeners>
                 </source>
           </sources>
           <switches>
                 <add name="System.Net" value="Verbose"/>
                 <add name="System.Net.Sockets" value="Verbose"/>
                 <add name="System.Net.Cache" value="Verbose"/>
                 <add name="System.Security" value="Verbose"/>
                 <add name="System.Net.Security" value="Verbose"/>
           </switches>

           <sharedListeners>
                 <add name="ExternalSourceTrace" type="IBM.WMQ.ExternalSourceTrace,
amqmdnet, Version=n.n.n.n, Culture=neutral, PublicKeyToken=dd3cb1c9aae9ec97" />
           </sharedListeners>
         <trace autoflush="true"/>
     </system.diagnostics>
```

Avertissement : La zone `Version` de l'entrée add name doit correspondre à la version du fichier .net `amqmdnet.dll` utilisée.

Tâches associées

Traçage des clients IBM MQ classes for .NET Framework à l'aide d'un fichier de configuration d'application

Exemples d'applications pour l'implémentation de TLS dans .NET géré
Des exemples d'application sont fournis pour illustrer l'implémentation de TLS pour .NET géré dans IBM MQ classes for .NET, XMS .NET et le canal personnalisé IBM MQ pour WCF.

Le tableau suivant indique l'emplacement des exemples d'application. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Tableau 79. Emplacement des modèles d'application pour l'implémentation de TLS dans .NET géré

Offre de pile IBM MQ.NET	**Emplacement des échantillons**
De base.NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
Canal personnalisé IBM MQ pour WCF	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\MQMessagingOneWayService.cs

Windows Utilisation du moniteur .NET

Le moniteur .NET est une application similaire à un moniteur de déclenchement IBM MQ .

Important : Pour plus d'informations, voir Fonctions disponibles uniquement avec l'installation principale sous Windows .

Vous pouvez créer des composants .NET qui sont instanciés chaque fois qu'un message est reçu dans une file d'attente surveillée, puis qui traitent ce message. Le moniteur .NET est démarré par la commande **runmqdnm** et arrêté par la commande **endmqdnm** . Pour plus de détails sur ces commandes, voir runmqdnm et endmqdnm.

Pour utiliser le moniteur .NET , vous écrivez un composant qui implémente l'interface IMQObjectTrigger , qui est définie dans amqmdnm.dll.

Les composants peuvent être transactionnels ou non transactionnels. Un composant transactionnel doit hériter de System.EnterpriseServices.ServicedComponent et être enregistré en tant que RequiresTransaction ou SupportsTransaction. Il ne doit pas être enregistré en tant que RequiresNew car le moniteur .NET a déjà lancé une transaction.

Le composant reçoit les objets MQQueueManager, MQQueue et MQMessage de **runmqdnm**. Il peut également recevoir une chaîne de paramètre utilisateur si elle a été spécifiée, à l'aide de l'option de ligne de commande *-u* , lorsque runmqdnm a été démarré. Notez que votre composant reçoit le contenu d'un message qui est arrivé dans la file d'attente surveillée dans un objet MQMessage. Il n'a pas besoin de se connecter au gestionnaire de files d'attente, d'ouvrir la file d'attente ou d'obtenir le message lui-même. Le composant doit ensuite traiter le message comme il convient et renvoyer le contrôle au moniteur .NET .

Si votre composant a été écrit en tant que composant transactionnel, il s'enregistre pour valider ou annuler la transaction à l'aide des fonctions fournies par System.EnterpriseServices.ServicedComponent.

Comme le composant reçoit les objets MQQueueManager et MQQueue ainsi que le message, il dispose d'informations de contexte complètes pour ce message et peut, par exemple, ouvrir une autre file d'attente sur le même gestionnaire de files d'attente sans avoir à se connecter séparément à IBM MQ.

Windows *Exemples de fragments de code*

Cette rubrique contient deux exemples de composants qui obtiennent un message du moniteur .NET et l'impriment, l'un utilisant le traitement transactionnel et l'autre le traitement non transactionnel. Un troisième exemple montre des routines d'utilitaire communes, applicables aux deux premiers exemples. Tous les exemples sont en C#.

Exemple 1: Traitement transactionnel

```
/*********************************************************************/
/* 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;
    }
  }
}
```

Exemple 2: Traitement non transactionnel

```
/*********************************************************************/
/* 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());
      }
    }
  }
}
```

Exemple 3: routines communes

```
/*********************************************************************/
/* 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;
    }
  }
}
```

Compilation de programmes IBM MQ .NET

Exemple de commandes permettant de compiler des applications .NET écrites dans différents langages.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour générer une application C# à l'aide de IBM MQ classes for .NET, utilisez la commande suivante:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin /out:MyProg.exe
MyProg.cs
```

Pour générer une application Visual Basic à l'aide de IBM MQ classes for .NET, utilisez la commande suivante:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:MyProg.exe MyProg.vb
```

Pour générer une application C++ gérée à l'aide de IBM MQ classes for .NET, utilisez la commande suivante:

```
cl /clr MQ_INSTALLATION_PATH\bin Myprog.cpp
```

Pour les autres langues, consultez la documentation fournie par le fournisseur de la langue.

Utilisation du client IBM MQ .NET autonome

Le client IBM MQ .NET vous permet de conditionner et de déployer un assemblage IBM MQ .NET sans avoir à utiliser l'installation complète du client IBM MQ sur les systèmes de production pour l'exécution de vos applications.

Avant de commencer

V 9.4.0 Depuis la IBM MQ 9.4.0, la bibliothèque client `amqmdnetstd.dll` installée à l'emplacement par défaut est basée sur .NET 6.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for .NET. Si vous utilisez une application .NET 6 , vous pouvez exécuter cette application sans qu'aucune recompilation ne soit nécessaire en effectuant une petite modification dans le fichier `runtimeconfig` pour définir `targetframeworkversion` sur `"net8.0"`.

V 9.4.0 V 9.4.0 Removed La bibliothèque client IBM MQ .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit IBM MQ 9.4.0.

Stabilized LTS La bibliothèque `amqmdnet.dll` est toujours fournie, mais cette bibliothèque est stabilisée, c'est-à-dire qu'aucune nouvelle fonction n'y sera introduite. Pour les fonctions les plus récentes, vous devez migrer vers la bibliothèque `amqmdnetstd.dll` . Toutefois, vous pouvez continuer à utiliser la bibliothèque `amqmdnet.dll` sur les éditions IBM MQ 9.1 Long Term Support ou Continuous Delivery .

Pourquoi et quand exécuter cette tâche

Vous pouvez générer vos applications IBM MQ .NET sur une machine où le client IBM MQ complet est installé et conditionner ultérieurement l'assemblage IBM MQ .NET , c'est-à-dire `amqmdnetstd.dll`, avec votre application et le déployer sur des systèmes de production.

Les applications que vous générez et déployez peuvent être les applications .NET traditionnelles, les services ou les applications Microsoft Azure Web / Worker

Dans de tels déploiements, le client IBM MQ .NET prend en charge uniquement le mode géré de connectivité à un gestionnaire de files d'attente. Les liaisons serveur et la connectivité en mode de client géré ne sont pas disponibles, car ces modes nécessitent une installation client IBM MQ complète. Toute tentative d'utilisation de ces deux autres modes entraîne une exception d'application.

Procédure

Référencement de l'assemblage client IBM MQ .NET dans les applications

- Référencez l'assemblage `amqmdnetstd.dll` dans votre application de la même manière que pour les versions précédentes.

  Définissez la propriété **`CopyLocal`** de l'assemblage `amqmdnetstd.dll` sur `True` pour vous assurer que l'assemblage `amqmdnetstd.dll` est copié dans le répertoire `bin` de l'application. La définition de cette propriété permet également à l'outil de conditionnement d'application de conditionner les fichiers binaires requis pour le déploiement sur les systèmes de production ainsi que les environnements de cloud Microsoft Azure PaaS .

Ajout de la prise en charge des transactions globales

- Assurez-vous que votre application déploie l'application de surveillance WMQDotnetXAMonitor sur la machine avec l'application elle-même.

  Si une application utilise la fonction de transaction globale gérée par IBM MQ .NET , elle doit également déployer WMQDotnetXAMonitor sur la machine avec l'application elle-même. Cet utilitaire est nécessaire pour récupérer les transactions en attente de validation.

Démarrage et arrêt d'une trace

- Pour IBM MQ classes for .NET Framework uniquement, pour démarrer et arrêter la trace à l'aide du fichier de configuration d'application et d'un fichier de configuration de trace spécifique à IBM MQ , voir Traçage d'un client IBM MQ classes for .NET Framework à l'aide d'un fichier de configuration d'application.

Vous devez utiliser le fichier de configuration d'application et un fichier de configuration de trace spécifique à IBM MQ car, en l'absence d'installation complète du client IBM MQ , les outils standard utilisés pour le démarrage et l'arrêt de la trace, **strmqtrc** et **endmqtrc**, ne sont pas disponibles.

Remarques :

- Cette méthode de génération de trace s'applique au client .NET géré redistribuable ainsi qu'au client .NET autonome. Voir .NET application runtime- Windows uniquement.
- Le fichier de configuration d'application n'est pas pris en charge dans les bibliothèques IBM MQ classes for .NET (.NET Standard et .NET 6). Pour activer la trace pour IBM MQ classes for .NET (bibliothèques.NET Standard et .NET 6), utilisez la variable d'environnement **MQDOTNET_TRACE_ON** . Voir Traçage des applications IBM MQ .NET à l'aide de variables d'environnement.

- V 9.4.0

  Démarrez et arrêtez la trace à l'aide du fichier `mqclient.ini` et définissez les propriétés appropriées de la strophe Trace.

  Voir Traçage des applications IBM MQ .NET avec mqclient.ini.

  Depuis IBM MQ 9.4.0, vous pouvez configurer la trace à l'aide du fichier `mqclient.ini` et en définissant les propriétés appropriées de la strophe Trace. Vous pouvez également activer et désactiver la fonction de trace de manière dynamique avec le fichier `mqclient.ini` .

Activation de la redirection de liaison dans le fichier de configuration de l'application

- Pour activer la référence de liaison de compilation de l'assemblage IBM MQ .NET à une version ultérieure de l'assemblage, ajoutez la propriété <dependentAssembly> au fichier de configuration de l'application.

  L'exemple de fragment suivant dans le fichier `app.config` redirige une application qui a été compilée à l'aide de la version IBM MQ 8.0.0 Fix Pack 2 (`8.0.0.2`) de l'assemblage IBM MQ .NET , mais plus tard, un groupe de correctifs, IBM MQ 8.0.0 Fix Pack 3, a ensuite été appliqué pour mettre à jour l'assemblage IBM MQ.NET vers `8.0.0.3`.

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- amqmdnet related binding redirect -->
    <dependentAssembly>
      <assemblyIdentity name="amqmdnet"
                        publicKeyToken="dd3cb1c9aae9ec97"
                        culture="neutral" />
      <codeBase version="8.0.0.2"
            href="file:///amqmdnet.dll"/>
      <bindingRedirect oldVersion="1.0.0.3-8.0.0.2"
                       newVersion="8.0.0.3"/>
      <publisherPolicy apply="no" />
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Concepts associés

«Installation de IBM MQ classes for .NET», à la page 569
IBM MQ classes for .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux

Clients redistribuables

Environnement d'application .NET - Windows uniquement

Tâches associées

«Utilisation de l'application WMQDotnetXAMonitor», à la page 589
Le client IBM MQ .NET fournit une application de surveillance XA, WmqDotnetXAMonitor, que vous pouvez utiliser pour récupérer toutes les transactions distribuées incomplètes. L'application XAMonitor WmqDotnetétablit une connexion au gestionnaire de files d'attente dans lequel les transactions sont en attente de validation, puis résout la transaction en fonction des paramètres que vous avez définis.

Traçage des applications IBM MQ .NET

OutboundSNI Propriété

Vous pouvez définir la propriété **OutboundSNI** dans une application à l'aide d'une propriété ou d'une variable d'environnement.

Depuis IBM MQ 9.3.0, vous pouvez définir le MQC MQC.OUTBOUND_SNI_PROPERTY dans l'application, en utilisant une table de hachage lors de l'utilisation de la classe `MQQueueManager` pour la connexion au gestionnaire de files d'attente.

Le MQC MQC.OUTBOUND_SNI_PROPERTY prend les valeurs suivantes:

- MQC.OUTBOUND_SNI_CHANNEL, qui est mappé à "CHANNEL"
- MQC.OUTBOUND_SNI_HOSTNAME, qui est mappé à "HOSTNAME"
- MQC.OUTBOUND_SNI_ASTERISK, qui correspond à "*"

En outre, vous pouvez définir la propriété **OutboundSNI** à l'aide de la variable d'environnement MQOUTBOUND_SNI, qui prend les valeurs suivantes:

- Canal
- HOSTNAME
- *

et définissez la valeur **OutboundSNI** dans le fichier `App.config` , comme avec toute autre propriété `mqclient.ini` .

Remarque : La valeur par défaut de la propriété est MQC.OUTBOUND_SNI_CHANNEL si aucune valeur spécifique n'est définie.

L'ordre de priorité pour la définition de la propriété **OutboundSNI** dans le noeud géré est le suivant:

1. Propriété de niveau application
2. Variable d'environnement

Pour la propriété **OutboundSNI** dans un noeud non géré, `mqclient.ini` uniquement est pris en charge.

Les propriétés définies dans le fichier `App.config` s'appliquent uniquement aux applications .NET Framework .

Si vous indiquez une valeur qui n'est pas valide au niveau de l'application ou dans le fichier `App.config` , le code retour MQRC_OUTBOUND_SNI_NOT_VALID est émis.

Si vous définissez une variable d'environnement qui n'est pas valide ou fournissez une valeur qui n'est pas valide dans le fichier `mqclient.ini` , la valeur par défaut `CHANNEL` est utilisée.

OutboundSNI et plusieurs certificats

IBM MQ utilise l'en-tête SNI pour fournir plusieurs fonctionnalités de certificats. Si une application se connecte à un canal IBM MQ configuré pour utiliser un certificat différent via la zone CERTLABL, l'application doit se connecter avec le paramètre **OutboundSNI** CHANNEL.

Si une application avec un paramètre **OutboundSNI** autre que CHANNEL se connecte à un canal avec un libellé de certificat configuré, l'application est rejetée avec une erreur MQRC_SSL_INITIALIZATION_ERROR et un message AMQ9673 est imprimé dans les journaux d'erreurs du gestionnaire de files d'attente.

Pour plus d'informations sur la façon dont IBM MQ fournit plusieurs fonctionnalités de certificat, voir How IBM MQ fournit plusieurs fonctionnalités de certificat .

Développement d'applications XMS .NET

IBM MQ Message Service Client (XMS) for .NET (XMS .NET) fournit une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que Java Message Service (JMS) API. IBM MQ Message Service Client (XMS) for .NET comprend une implémentation complète de XMS, qui peut être utilisée par tout langage compatible avec .NET.

Avant de commencer

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for XMS .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client XMS .NET générée à l'aide de.NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Pourquoi et quand exécuter cette tâche

XMS prend en charge :

- Messagerie point-à-point
- Messagerie de type publication/abonnement
- Distribution synchrone des messages
- une distribution asynchrone de messages

Une application XMS peut échanger des messages avec les types suivants d'application :

- une application XMS
- une application IBM MQ classes for JMS
- une application IBM MQ native
- Une application JMS qui utilise le fournisseur de messagerie par défaut IBM MQ

Une application XMS peut se connecter et utiliser les ressources des serveurs de messagerie suivants :

Gestionnaire de files d'attente IBM MQ
L'application peut se connecter en mode liaisons ou client.

WebSphere Application Server service integration bus
L'application peut utiliser une connexion TCP/IP directe ou HTTP via TCP/IP.

IBM Integration Bus
Les messages sont transportés entre l'application et le courtier via WebSphere MQ Real-Time Transport. Les messages peuvent être distribués à l'application à l'aide de WebSphere MQ Multicast Transport.

En se connectant à un gestionnaire de files d'attente IBM MQ , une application XMS peut utiliser WebSphere MQ Enterprise Transport pour communiquer avec IBM Integration Bus. Une application XMS peut également effectuer des publications et s'abonner en se connectant à IBM MQ.

V 9.4.0 IBM MQ 9.4.0 fournit une bibliothèque client XMS .NET générée avec .NET 6 comme infrastructure cible. Pour plus d'informations, voir «Installation de IBM MQ classes for XMS .NET», à la page 634.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for XMS .NET. Pour plus d'informations, voir «Installation de IBM MQ classes for XMS .NET», à la page 634.

Les applications gérées par XMS .NET peuvent équilibrer automatiquement les connexions entre les gestionnaires de files d'attente en cluster. Les bibliothèques IBM MQ classes for XMS .NET et IBM MQ classes for XMS .NET Framework sont prises en charge. Pour plus d'informations, voir A propos des clusters uniformes et Equilibrage automatique des applications.

Pour plus d'informations sur les différences entre IBM MQ classes for XMS .NET Framework et IBM MQ classes for XMS .NET, voir «Installation de IBM MQ classes for XMS .NET», à la page 634.

Tâches associées
Contacter le support IBM

Traitement des incidents liés à XMS .NETproblems

Styles de messagerie pris en charge par XMS

XMS prend en charge les styles de messagerie point-à-point et de publication / abonnement.

Les styles de messagerie sont également appelés domaines de messagerie.

Messagerie point-à-point

Une forme courante de messagerie point-à-point utilise la mise en file d'attente. Dans le cas le plus simple, une application envoie un message à une autre application en identifiant, de manière implicite ou explicite, une file d'attente de destination. Le système de messagerie et de mise en file d'attente sous-jacent reçoit le message de l'application émettrice et l'achemine vers la file d'attente de destination. L'application de réception peut alors extraire le message de la file d'attente.

Si le système de messagerie et de mise en file d'attente sous-jacent contient IBM Integration Bus, IBM Integration Bus peut répliquer un message et acheminer des copies du message vers différentes files d'attente. Plusieurs applications peuvent donc recevoir le message. IBM Integration Bus peut également transformer un message et y ajouter des données.

Une caractéristique clé de la messagerie point-à-point est qu'une application place un message dans une file d'attente locale lorsqu'elle envoie un message. Le système de messagerie et de mise en file d'attente sous-jacent détermine la file d'attente de destination à laquelle le message est envoyé. L'application de réception extrait le message de la file d'attente de destination.

Messagerie de type publication/abonnement

Dans la messagerie de publication / abonnement, il existe deux types d'application: le diffuseur de publications et l'abonné.

Un *diffuseur* fournit des informations sous la forme de messages de publication. Lorsqu'un diffuseur de publications publie un message, il spécifie une rubrique, laquelle identifie l'objet des informations contenues dans le message.

Un *abonné* est un consommateur des informations publiées. Un abonné spécifie les rubriques qui l'intéressent en créant des abonnements.

Le système de publication/abonnement reçoit des publications des diffuseurs et des abonnements des abonnés. Il achemine les publications vers les abonnés. Un abonné reçoit uniquement les publications relatives aux rubriques auxquelles il s'est abonné.

Une caractéristique essentielle de la messagerie de publication / abonnement est qu'un diffuseur de publications identifie une rubrique lorsqu'il publie un message. Il n'identifie pas les abonnés. Si un message est publié sur une rubrique pour laquelle il n'y a aucun abonné, aucune application ne reçoit le message.

Une application peut être à la fois un diffuseur de publications et un abonné.

Modèle d'objet XMS

L'API XMS est une interface orientée objet. Le modèle d'objet XMS s'appuie sur le modèle d'objet JMS 1.1.

Classes XMS principales

Les principales classes XMS ou types d'objet sont les suivants:

ConnectionFactory
Un objet `ConnectionFactory` encapsule un ensemble de paramètres pour une connexion. Une application utilise `ConnectionFactory` pour créer une connexion. Une application peut fournir les paramètres lors de l'exécution et créer un objet `ConnectionFactory`. Les paramètres de connexion peuvent également être stockés dans un référentiel d'objets gérés. Une application peut extraire un objet du référentiel et créer un objet `ConnectionFactory` à partir de celui-ci.

Connexion
Un objet `Connection` encapsule une connexion active d'une application vers un serveur de messagerie. Une application utilise une connexion pour créer des sessions.

Destination
Une application envoie ou reçoit des messages via un objet `Destination`. Dans le domaine de publication / abonnement, un objet `Destination` encapsule une rubrique et, dans le domaine point à point, un objet `Destination` encapsule une file d'attente. Une application peut fournir les paramètres pour créer un objet `Destination` lors de l'exécution. Elle peut également créer un objet `Destination` à partir d'une définition d'objet stockée dans un référentiel d'objets gérés.

Session
Un objet `Session` est un contexte à unité d'exécution unique pour l'envoi et la réception de messages. Une application utilise un objet `Session` pour créer des objets `Message`, `MessageProducer` et `MessageConsumer`.

Message
Un objet `Message` encapsule l'objet `Message` qu'une application envoie via un objet `MessageProducer` ou reçoit via un objet `MessageConsumer`.

MessageProducer
Un objet `MessageProducer` est utilisé par une application pour envoyer des messages vers une destination.

MessageConsumer
Un objet `MessageConsumer` est utilisé par une application pour recevoir des messages envoyés vers une destination.

Objets XMS et leurs relations

La Figure 52, à la page 632 présente les principaux types d'objet XMS : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message et Destination. Une application utilise une fabrique de connexions pour créer une connexion et utilise une connexion pour créer des sessions. L'application peut ensuite utiliser une session pour créer des messages, des expéditeurs de message et des consommateurs de message. L'application utilise un expéditeur de message pour envoyer des messages vers une destination et utilise un consommateur de message pour recevoir des messages envoyés vers une destination.

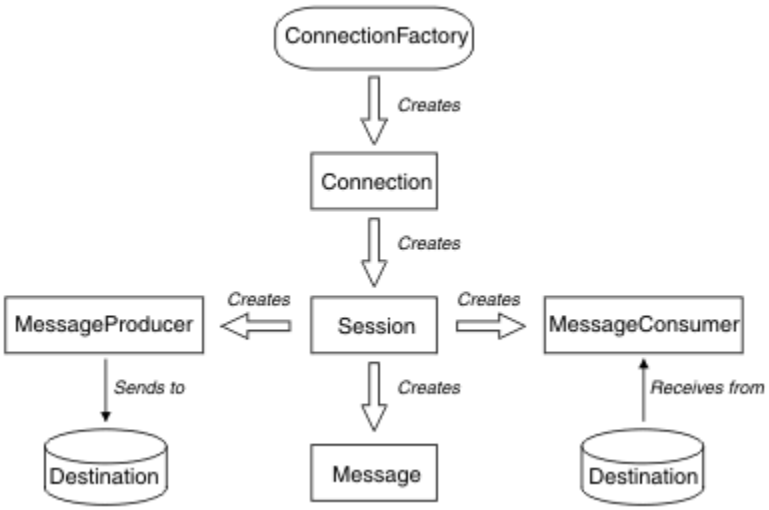


Figure 52. Objets XMS et leurs relations

Dans XMS .NET, les classes XMS sont définies en tant qu'ensemble d'interfaces .NET. Lorsque vous codez des applications XMS .NET , vous n'avez besoin que des interfaces déclarées.

Le modèle d'objet XMS est basé sur les interfaces indépendantes du domaine décrites dans Java Message Service Specification, Version 1.1. Les classes spécifiques à un domaine, par exemple `Topic`, `TopicPublisher` et `TopicSubscriber`, ne sont pas fournies.

Attributs et propriétés des objets XMS

Un objet XMS peut avoir des attributs et des propriétés, qui sont des caractéristiques de l'objet, qui sont implémentées de différentes manières:

Attributs
Un attribut est une caractéristique d'objet, toujours présente et occupant de l'espace de stockage, même si aucune valeur ne lui est attribuée. Un attribut est donc similaire à une zone dans une structure de données de longueur fixe. Une des caractéristiques typiques des attributs est que chacun d'eux dispose de ses propres méthodes de définition et d'obtention de sa valeur.

Propriétés
Une propriété d'objet est présente et occupe de l'espace de stockage une fois que sa valeur a été définie. Une propriété ne peut pas être supprimée ou son espace de stockage récupéré une fois que sa valeur a été définie. Vous pouvez modifier cette valeur. XMS fournit un ensemble de méthodes génériques pour définir et obtenir des valeurs de propriété.

Objets gérés

Les objets gérés permettent d'administrer les paramètres de connexion utilisés à partir d'un référentiel central. Une application extrait des définitions d'objet depuis le référentiel central et les utilise pour créer des objets `ConnectionFactory` et `Destination`. Les objets gérés permettent de découpler les applications des ressources qu'elles utilisent lors de leur exécution.

Par exemple, des applications XMS peuvent être écrites et testées avec des objets gérés qui font référence à un ensemble de connexions et de destinations dans un environnement de test. Lorsque les applications sont déployées, les objets gérés peuvent être modifiés pour que les applications soient configurées de manière à faire référence aux connexions et aux destinations de l'environnement de production.

XMS prend en charge deux types d'objet géré :

- L'objet `ConnectionFactory`, utilisé par les applications pour établir la connexion initiale au serveur.
- L'objet `Destination`, utilisé par les applications pour spécifier la destination des messages envoyés et la source des messages reçus. Une destination est une rubrique ou une file d'attente sur le serveur auquel l'application se connecte.

L'outil d'administration **JMSAdmin** est intégré à IBM MQ. Il permet de créer et de gérer des objets gérés dans un référentiel central d'objets gérés.

Les objets gérés du référentiel peuvent être utilisés par les applications IBM MQ classes for JMS et XMS. Les applications XMS peuvent utiliser les objets `ConnectionFactory` et `Destination` pour se connecter à un IBM MQ gestionnaire de files d'attente. Un administrateur peut modifier les définitions d'objet conservées dans le référentiel sans affecter le code d'application.

Le diagramme suivant illustre la manière dont une application XMS utilise généralement les objets gérés. La partie gauche du diagramme présente un référentiel contenant des définitions d'objet ConnectionFactory et Destination gérés par l'intermédiaire d'une console d'administration. La partie droite du diagramme présente une application XMS qui recherche des définitions d'objet dans le référentiel, puis les utilise lors de la connexion à un serveur de messagerie.

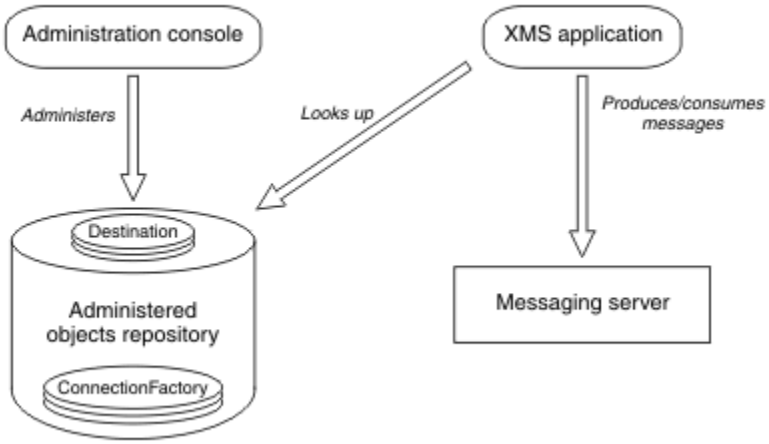


Figure 53. Utilisation standard d'objets gérés par une application XMS

Modèle de message XMS

Le modèle de message XMS est identique au modèle de message IBM MQ classes for JMS.

XMS implémente les mêmes zones d'en-tête et les mêmes propriétés de message que IBM MQ classes for JMS :

- Zones d'en-tête JMS. Ces zones portent un nom qui commence par le préfixe JMS.
- Propriétés définies par JMS. Ces zones contiennent des propriétés dont le nom commence par le préfixe JMSX.
- Propriétés définies par IBM. Ces zones contiennent des propriétés dont le nom commence par le préfixe JMS_IBM_.

En conséquence, les applications XMS peuvent échanger des messages avec des applications IBM MQ classes for JMS. Dans chaque message, certaines zones d'en-tête et certaines propriétés sont définies par l'application et d'autres, par XMS ou IBM MQ classes for JMS. Certaines zones définies par XMS ou IBM MQ classes for JMS le sont lors de l'envoi du message, d'autres lors de sa réception. Les zones d'en-tête et les propriétés sont propagées avec un message via un serveur de messagerie. Elles sont disponibles pour chaque application qui reçoit le message.

Concepts associés

IBM MQ classes for JMS

Windows Linux Installation de IBM MQ classes for XMS .NET

IBM MQ classes for XMS .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux.

Installation

V 9.4.0 IBM MQ 9.4.0 fournit une bibliothèque client XMS .NET générée avec .NET 6 comme infrastructure cible. Depuis la IBM MQ 9.4.0, Microsoft .NET 6.0 est la version minimale requise pour l'exécution d'applications à l'aide de bibliothèques IBM MQ générées à l'aide de .NET 6 comme infrastructure cible. La bibliothèque client XMS .NET générée à l'aide de .NET 6 comme infrastructure cible est disponible sous *MQ_INSTALLATION_PATH*/`bin` sous Windows et sous *MQ_INSTALLATION_PATH*/`lib64` sous Linux.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for XMS .NET. Si vous utilisez une application .NET 6 , vous pouvez exécuter

cette application sans qu'aucune recompilation ne soit nécessaire en effectuant une petite modification dans le fichier `runtimeconfig` pour définir `targetframeworkversion` sur `"net8.0"`.

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for XMS .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client XMS .NET générée à l'aide de.NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Bibliothèque `amqmxmsstd.dll`

V 9.4.0 Depuis la IBM MQ 9.4.0, la bibliothèque `amqmxmsstd.dll` générée à l'aide de .NET 6 comme infrastructure cible est disponible aux emplacements suivants:

- Windows Sous Windows: *MQ_INSTALLATION_PATH*`\bin`. Les exemples d'application sont installés dans *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.
- Linux Sous Linux: *MQ_INSTALLATION_PATH*`\lib64`. Les exemples .NET se trouvent dans *MQ_INSTALLATION_PATH*`/samp/dotnet/samples/cs/core/base`.

V 9.4.0 V 9.4.0 Removed La bibliothèque client XMS .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Avertissement : V 9.4.0 V 9.4.0 Removed A partir de IBM MQ 9.4.0, les bibliothèques client XMS .NET générées en utilisant .NET Standard 2.0 comme infrastructure cible sont supprimées. Ces bibliothèques sont obsolètes dans IBM MQ 9.3.1.

Stabilized Toutes les bibliothèques `IBM.XMS.*` sont toujours fournies, mais ces bibliothèques sont stabilisées, c'est-à-dire qu'aucune nouvelle fonction ne leur est introduite. Pour les fonctions les plus récentes, vous devez effectuer une migration vers la bibliothèque `amqmxmsstd.dll` . Toutefois, vous pouvez continuer à utiliser les bibliothèques existantes dans les éditions IBM MQ 9.1 Long Term Support ou Continuous Delivery .

V 9.4.0 V 9.4.0 Voici deux scénarios que vous pouvez rencontrer après la suppression des bibliothèques `netstandard2.0` :

- Si vous utilisez une application IBM MQ classes for XMS .NET Framework générée à l'aide des bibliothèques `netstandard2.0` telles que `amqmdnetstd.dll`, vous devez régénérer votre application avec les bibliothèques Microsoft.NET Framework 4.7.2 telles que `amqmdnet.dll`, afin que votre application s'exécute correctement. Si vous ne régénérez pas votre application, vous risquez d'obtenir un System.IO.Unexceptionable non exceptionnel:

  ```
  Exception interceptée: System.IO.FileLoadException: Impossible de charger le fichier ou
  l'assemblage'amqmdnetstd, Version=9.3.5.0, Culture=neutre, PublicKeyToken=23d6cb914eeaac0e'ou
  l'une de ses dépendances. La définition de manifeste de l'assemblage localisé ne correspond pas
  à la référence d'assemblage. (Exception de HRESULT: 0x80131040)
  Nom de fichier:'amqmdnetstd, Version=9.3.5.0, Culture=neutral, PublicKeyToken=23d6cb914eeaac0e'
     à SimplePut.SimplePut.PutMessages()
     sur SimplePut.SimplePut.Main (String [ ] args) dans C:\SampleCode\Program.cs:line 132
  ```

- Si vous utilisez une application .NET 6 générée à l'aide de bibliothèques `netstandard2.0` , il vous suffit de remplacer ces bibliothèques par les mêmes bibliothèques .NET 6 dans le dossier `bin` du répertoire d'exécution de l'application. Aucune régénération n'est requise.

  Remarque : Le niveau de la bibliothèque .NET 6 de remplacement doit toujours être identique ou supérieur à celui de la bibliothèque `netstandard2.0` remplacée.

Les IBM MQ classes for XMS .NET Standard sont disponibles pour téléchargement à partir du référentiel NuGet . Le package NuGet contient à la fois la bibliothèque `amqmxmsstd.dll` et la bibliothèque

amqmdnetstd.dll . amqmxmsstd.dll dépend de amqmdnetstd.dll et, lors du conditionnement de l'application XMS .NET Core, amqmxmsstd.dll et amqmdnetstd.dll doivent être conditionnés avec l'application XMS .NET Core. Pour plus d'informations, voir la section «Téléchargement d'IBM MQ classes for XMS .NET depuis le référentiel NuGet», à la page 638.

commande dspmqver

Vous pouvez utiliser la commande **dspmqver** pour afficher les informations de version et de génération pour le composant .NET Core .

Comparaison entre les bibliothèques IBM MQ classes for XMS .NET Framework et IBM MQ classes for XMS .NET Bibliothèques .NET 6 et .NET 6)

Le tableau suivant répertorie les fonctions de IBM MQ classes for XMS .NET Framework par rapport aux fonctions de IBM MQ classes for XMS .NET et .NET 6).

Tableau 80. Différences entre IBM MQ classes for XMS .NET Framework et IBM MQ classes for XMS .NET

Fonction	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET**
Noms de classe (API)	Toutes les classes restent identiques sur chaque réseau.	Toutes les classes restent identiques sur chaque réseau.
Système d'exploitation	Windows	Windows Conteneurs dockérisés Linux macOS
Fichier app.config (fichier de configuration permettant d'activer la trace dans le client redistribuable)	Le fichier app.config est utilisé pour activer la trace pour le package redistribuable.	app.config n'est pas pris en charge. Utilisez des variables d'environnement.

Tableau 80. Différences entre IBM MQ classes for XMS .NET Framework et IBM MQ classes for XMS .NET (suite)

Fonction	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET
Fonction de trace	Pour tracer le client XMS .NET , vous pouvez utiliser les variables d'environnement existantes, telles que la variable d'environnement **XMS_TRACE_ON** utilisée pour activer la trace. Pour plus d'informations, voir Configuration de la trace XMS .NET à l'aide de variables d'environnement XMS. Pour les clients redistribuables, le fichier `app.config` peut être utilisé pour activer la trace. V 9.4.0 Depuis IBM MQ 9.4.0, vous pouvez activer et désactiver la trace à l'aide du fichier `mqclient.ini` et en définissant les propriétés appropriées de la section Trace. Vous pouvez également activer et désactiver la fonction de trace de manière dynamique avec le fichier `mqclient.ini` . Pour plus d'informations, voir Suivi des applications IBM MQ .NET avec mqclient.ini.	Pour tracer le client XMS .NET , vous pouvez utiliser les variables d'environnement existantes, telles que la variable d'environnement **XMS_TRACE_ON** utilisée pour activer la trace. Pour plus d'informations, voir Configuration de la trace XMS .NET à l'aide de variables d'environnement XMS. V 9.4.0 Depuis IBM MQ 9.4.0, vous pouvez activer et désactiver la trace à l'aide du fichier `mqclient.ini` et en définissant les propriétés appropriées de la section Trace. Vous pouvez également activer et désactiver la fonction de trace de manière dynamique avec le fichier `mqclient.ini` . Pour plus d'informations, voir Suivi des applications IBM MQ .NET avec mqclient.ini.
Modes de transport	Géré, non géré, liaisons	Gérée
TLS	Le magasin de clés Windows est utilisé pour stocker les certificats.	Sous Windows, le magasin de clés doit être utilisé pour stocker les certificats. Les valeurs admises sont *USER et *SYSTEM. En fonction de l'entrée, le client IBM MQ .NET examine le magasin de clés Windows de l'utilisateur en cours ou de l'ensemble du système. Sous Linux, il est recommandé d'utiliser la classe X509Store pour installer les certificats et .NET Core installe les certificats à l'emplacement suivant: `".dotnet/corefx/cryptography/x509stores"`.
Table de définition de canal du client (CCDT)	Pris en charge	Prise en charge. Les paramètres du chemin de la table de définition de canal du client (CCDT) sont les mêmes que pour les classes .NET Framework.
Reconnexion automatique du client	Pris en charge	Pris en charge

Tableau 80. Différences entre IBM MQ classes for XMS .NET Framework et IBM MQ classes for XMS .NET (suite)		
Fonction	**IBM MQ classes for XMS .NET Framework**	**IBM MQ classes for XMS .NET**
Transactions réparties	Pris en charge	Non pris en charge
Installation de bibliothèques de liaison dynamique (dll) dans le cache d'assemblage global (GAC)	Les bibliothèques de liaison dynamique sont installées dans le cache d'assemblage global (GAC) dans le cadre de l'installation d'IBM MQ.	Les bibliothèques de liaison dynamique ne sont pas installées dans le cache d'assemblage global (GAC) dans le cadre de l'installation d'IBM MQ.
Prise en charge des types de connexion WMQ, WPM et RTT	Prend en charge les types de connexion WMQ, WPM et RTT	Prise en charge de WMQ uniquement
Objets gérés JNDI	Prend en charge LDAP et FileSystem	Prend en charge FileSystem uniquement

Depuis IBM MQ 9.3.0, pour exécuter IBM MQ classes for XMS .NET Framework , vous devez installer Microsoft.NET Framework V4.7.2 ou ultérieure.

Tâches associées

«Utilisation des modèles d'application XMS», à la page 645
Les exemples d'application XMS .NET fournissent une présentation des fonctions communes de chaque API. Vous pouvez les utiliser pour vérifier votre installation et votre configuration de serveur de messagerie, ou pour vous aider à construire vos propres applications.

Windows Linux Téléchargement d'IBM MQ classes for XMS .NET depuis le référentiel NuGet

Les IBM MQ classes for XMS .NET sont disponibles pour téléchargement à partir du référentiel NuGet , afin qu'ils puissent être facilement consommés par les développeurs .NET .

Pourquoi et quand exécuter cette tâche

NuGet est le gestionnaire de package pour les plateformes de développement Microsoft , y compris .NET. Les outils client NuGet permettent de produire et de consommer des packages. Un package NuGet est un fichier compressé unique avec l'extension `.nupkg` qui contient du code compilé (DLL), d'autres fichiers associés à ce code et un manifeste descriptif qui inclut des informations telles que le numéro de version du package.

Vous pouvez télécharger le package `IBMXMSDotnetClient` NuGet , qui contient à la fois la bibliothèque `amqmdnetstd.dll` et la bibliothèque `amqmxmsstd.dll` , à partir de la galerie NuGet , qui est le référentiel de package central utilisé par tous les auteurs et destinataires de package.

Remarque : V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, le package NuGet contient des bibliothèques générées à l'aide de .NET 6 comme infrastructure cible.

Removed La bibliothèque client XMS .NET générée à l'aide de .NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

V 9.4.0 V 9.4.0 Depuis la IBM MQ 9.4.0, IBM MQ prend en charge les applications .NET 8 à l'aide de IBM MQ classes for XMS .NET. Si vous utilisez une application .NET 6 , vous pouvez exécuter cette application sans qu'aucune recompilation ne soit nécessaire en effectuant une petite modification dans le fichier `runtimeconfig` pour définir `targetframeworkversion` sur `"net8.0"`.

Il existe trois façons de télécharger le package `IBMXMSDotnetClient` :

- En utilisant Microsoft Visual Studio. NuGet est distribué en tant qu'extension Microsoft Visual Studio . Depuis Microsoft Visual Studio 2012, NuGet est préinstallé par défaut.
- A partir de la ligne de commande, à l'aide du gestionnaire de package NuGet ou de l'interface de ligne de commande .NET .
- En utilisant un navigateur Web.

Comme pour le package redistribuable, vous activez la trace à l'aide de la variable d'environnement **XMS_TRACE_ON**.

Procédure

- Pour télécharger le package `IBMXMSDotnetClient` à l'aide de l'interface utilisateur de Package Manager dans Microsoft Visual Studio, procédez comme suit:
  a) Cliquez avec le bouton droit de la souris sur le projet .NET , puis cliquez sur **Manage Nuget Packages**.
  b) Cliquez sur l'onglet **Parcourir** et recherchez `"IBMXMSDotnetClient"`.
  c) Sélectionnez le package et cliquez sur **Installer**.

  Lors de l'installation, le gestionnaire de packages fournit des informations de progression sous la forme d'instructions de console.
- Pour télécharger le package `IBMXMSDotnetClient` à partir de la ligne de commande, choisissez l'une des options suivantes:
  - A l'aide de NuGet Package Manager, entrez la commande suivante:

    ```
    Install-Package IBMXMSDotnetClient -Version 9.1.4.0
    ```

    Lors de l'installation, le gestionnaire de packages fournit des informations de progression sous la forme d'instructions de console. Vous pouvez rediriger la sortie vers un fichier journal.
  - A l'aide de l'interface de ligne de commande .NET , entrez la commande suivante:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- A l'aide d'un navigateur Web, téléchargez le package `IBMXMSDotnetClient` à partir de https://www.nuget.org/packages/IBMXMSDotnetClient.

Concepts associés

«Installation de IBM MQ classes for .NET», à la page 569
IBM MQ classes for .NET, y compris les exemples, sont installés avec IBM MQ sous Windows et Linux

Informations sur la licence de IBM MQ Client for .NET

Tâches associées

«Téléchargement de IBM MQ classes for .NET à partir du référentiel NuGet», à la page 574
Le fichier IBM MQ classes for .NET peut être téléchargé à partir du référentiel NuGet afin de pouvoir être facilement utilisé par les développeurs .NET .

Configuration de l'environnement d'un serveur de messagerie

Les rubriques de cette section expliquent comment configurer l'environnement d'un serveur de messagerie pour permettre aux applications XMS de se connecter à un serveur.

Pourquoi et quand exécuter cette tâche

Pour les applications qui se connectent à un gestionnaire de files d'attente IBM MQ, le client IBM MQ (ou le gestionnaire de files d'attente pour le mode liaisons) est requis.

Il n'y a aucun prérequis pour les applications qui utilisent une connexion en temps réel à un courtier.

Vous devez configurer l'environnement du serveur de messagerie avant d'exécuter des applications XMS, y compris les modèles d'application fournis avec XMS.

Cette section contient les rubriques suivantes :

- «Configuration du gestionnaire de files d'attente et du courtier pour une application se connectant à un gestionnaire de files d'attente IBM MQ», à la page 642
- «Installation de IBM MQ classes for XMS .NET», à la page 634
- «Configuration d'un courtier pour une application utilisant une connexion en temps réel à un courtier», à la page 643
- «Configuration du bus d'intégration de services pour une application se connectant à WebSphere Application Server», à la page 644

Programmes d'écoute de message dans XMS .NET

Un programme d'écoute de message est utilisé pour recevoir des messages de manière asynchrone. Contrairement à l'appel `MessageConsumer.receive()` , le programme d'écoute des messages ne bloque pas l'unité d'exécution appelante, mais distribue les messages à une méthode de rappel spécifiée par l'application, généralement la méthode `onMessage` .

La distribution des messages démarre une fois que la méthode `Connection.Start()` est appelée. La distribution des messages peut être arrêtée et reprise à tout moment à l'aide des méthodes `Connection.Stop()` et `Connection.Start()` respectivement.

Une fois que la méthode `Connection.Start()` est appelée après avoir défini un programme d'écoute de message sur au moins un consommateur dans une session, cette session devient une session asynchrone. Une fois qu'une session devient asynchrone, il n'est plus possible d'appeler une méthode synchrone XMS .NET . Par exemple, `MessageProducer.Send()`. Cela génère une exception avec le code anomalie IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

Appels synchrones dans une session asynchrone

`Session.Close` est le seul appel synchrone autorisé dans une session asynchrone. Les applications peuvent également effectuer des appels synchrones (sauf `Session.Close`) à l'aide de la méthode de rappel du programme d'écoute des messages, c'est-à-dire la méthode `onMessage` .

Outre ces deux options, vous devez arrêter la connexion à l'aide de la méthode `Connection.Stop()` pour qu'une application effectue un appel synchrone. Une fois les appels effectués, vous devez reprendre la connexion à l'aide de la méthode `Connection.Start()` . qui redémarre la distribution des messages.

Combien de consommateurs de messages asynchrones une session peut-elle avoir?

Une session peut comporter plusieurs destinataires de messages asynchrones. Mais à tout moment, un message est distribué à un seul consommateur. En pratique, cela signifie que lorsqu'un deuxième message arrive alors que XMS .NET a appelé la méthode `onMessage()` d'un consommateur pour distribuer le premier message, le deuxième message n'est pas distribué à un consommateur dans la session tant que la méthode `onMessage()` n'est pas renvoyée.

Le deuxième message est distribué à un consommateur dans la session uniquement après le retour de la méthode `onMessage()` . En effet, une session gère la distribution des messages aux consommateurs à l'aide d'une seule unité d'exécution. Cela signifie qu'un seul message peut être distribué à la fois et que le consommateur peut être n'importe quel message.

Si une application requiert la distribution simultanée de messages, c'est-à-dire que tous les destinataires doivent recevoir des messages en même temps, l'application doit créer plusieurs sessions et chacune doit avoir un destinataire de message asynchrone.

Les exemples suivants illustrent cette fonction plus clairement.

Dans le premier exemple, il existe plusieurs consommateurs de messages asynchrones dans une session. Une session S comporte trois consommateurs de messages asynchrones: AMC1, AMC2 et AMC3 qui reçoivent des messages provenant de trois destinations différentes Q1, Q2 et Q3.

Etant donné qu'il n'y a qu'une seule session S, il n'y a qu'une seule unité d'exécution de distribution de messages pour distribuer les messages aux consommateurs AMC1, AMC2et AMC3. Lorsque la session distribue le message à AMC1, les deux autres destinataires AMC2 et AMC3 attendent, même s'il existe des messages dans Q2 et Q3 prêts pour la distribution.

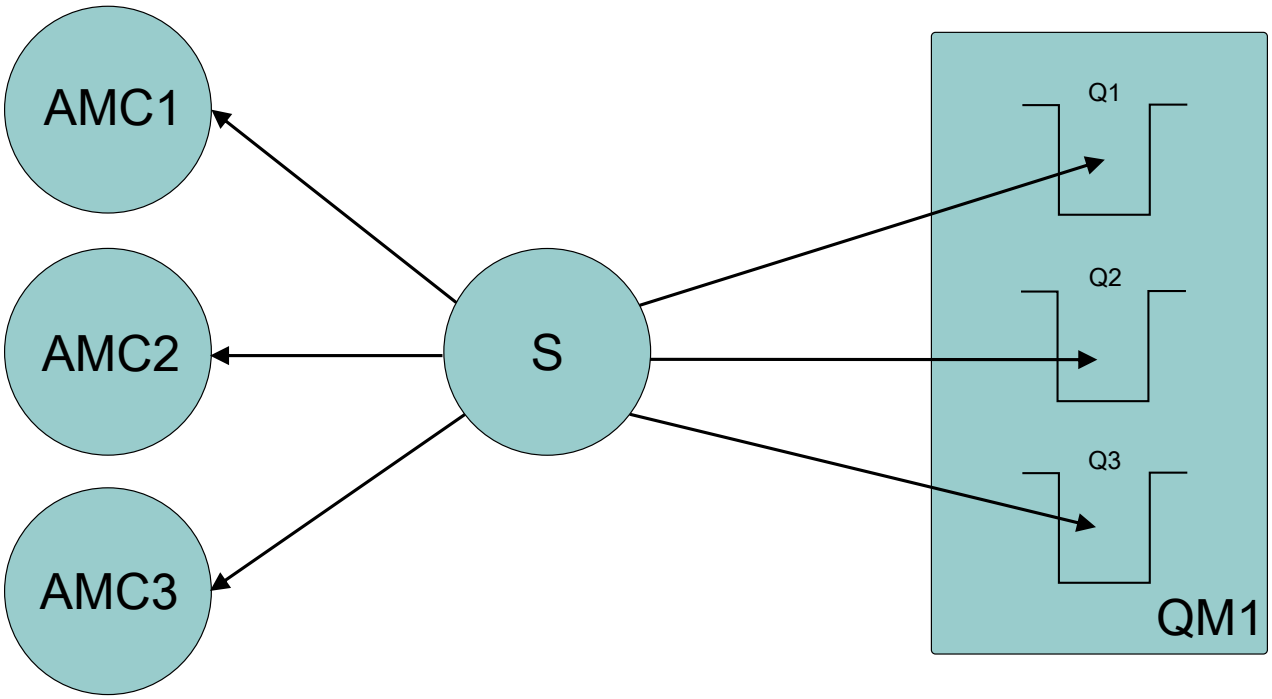


Figure 54. Une session avec trois consommateurs de messages asynchrones

Dans le second cas, il existe plusieurs sessions S1, S2et S3, chacune ayant un consommateur de message asynchrone AMC1, AMC2et AMC3 respectivement. Comme il existe un consommateur pour chaque session, les messages sont distribués simultanément aux consommateurs.

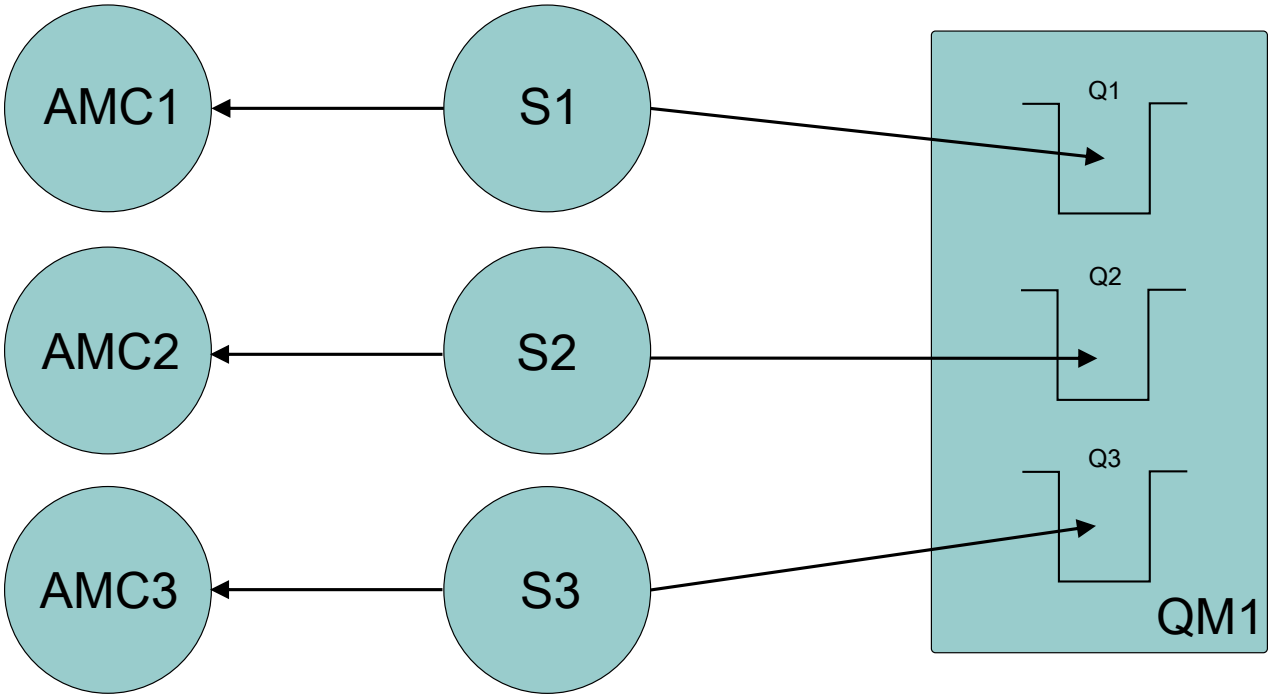


Figure 55. Plusieurs sessions, chacune avec un consommateur de message asynchrone

Cela indique que si vous avez besoin d'une distribution simultanée des messages, vous avez besoin de plusieurs sessions.

Configuration du gestionnaire de files d'attente et du courtier pour une application se connectant à un gestionnaire de files d'attente IBM MQ

Cette section suppose que vous utilisez IBM WebSphere MQ 7.0.1 ou une version ultérieure. Pour pouvoir exécuter une application qui se connecte à un gestionnaire de files d'attente IBM MQ, vous devez configurer le gestionnaire de files d'attente. Pour une application de publication/abonnement, une configuration supplémentaire est requise si vous utilisez l'interface de publication/abonnement en file d'attente.

Avant de commencer

XMS fonctionne avec IBM Integration Bus ou WebSphere Message Broker 6.1 ou version ultérieure

Avant de commencer cette tâche, procédez comme suit:

- Assurez-vous que votre application a accès à un gestionnaire de files d'attente en cours d'exécution.
- Si votre application est une application de publication / abonnement et qu'elle utilise l'interface de publication / abonnement en file d'attente, assurez-vous que l'attribut **PSMODE** est défini sur ENABLED sur le gestionnaire de files d'attente.
- Assurez-vous que votre application utilise une fabrique de connexions dont les propriétés sont définies de manière appropriée pour la connexion au gestionnaire de files d'attente. Si votre application est de type publication/abonnement, assurez-vous que les propriétés de fabrique de connexions appropriées sont définies pour l'utilisation du courtier. Pour plus d'informations sur les propriétés d'une fabrique de connexions, voir Propriétés de ConnectionFactory.

Pourquoi et quand exécuter cette tâche

Vous configurez le gestionnaire de files d'attente et le courtier pour exécuter des applications XMS de la même manière que vous configurez le gestionnaire de files d'attente et l'interface de publication / abonnement en file d'attente pour exécuter des applications IBM MQ JMS . Les étapes suivantes récapitulent la procédure à suivre.

Procédure

1. Dans le gestionnaire de files d'attente, créez les files d'attente dont votre application a besoin.

   Pour plus d'informations sur la création de files d'attente, voir Définition de files d'attente.

   Si votre application est une application de publication/abonnement et utilise une interface de publication/abonnement en file d'attente qui doit accéder aux files d'attente système IBM MQ classes for JMS, attendez jusqu'à l'étape 4a avant de créer les files d'attente.
2. Accordez à l'ID utilisateur associé à votre application le droit de connexion au gestionnaire de files d'attente, ainsi que les droits requis pour accéder aux files d'attente.

   Pour plus d'informations sur les droits, voir Sécurisation. Si votre application se connecte au gestionnaire de files d'attente en mode client, voir aussi Clients et serveurs.
3. Si votre application se connecte au gestionnaire de files d'attente en mode client, assurez-vous qu'un canal de connexion serveur est défini sur le gestionnaire de files d'attente et qu'un programme d'écoute est démarré.

   Il n'est pas nécessaire de répéter cette étape pour chaque application qui se connecte au gestionnaire de files d'attente. Une définition de canal de connexion serveur et un programme d'écoute peuvent prendre en charge toutes les applications qui se connectent en mode client.
4. Si votre application est une application de publication/abonnement et qu'elle utilise l'interface de publication/abonnement en file d'attente, effectuez les étapes ci-dessous.
   a) Dans le gestionnaire de files d'attente, créez les files d'attente système IBM MQ classes for JMS en exécutant le script des commandes MQSC fourni avec IBM MQ. Assurez-vous que l'ID utilisateur associé à IBM Integration Bus ou WebSphere Message Broker dispose des droits requis pour accéder aux files d'attente.

Pour plus d'informations sur le script, voir Utilisation de IBM MQ classes for Java.

Effectuez cette étape une seule fois pour le gestionnaire de files d'attente. Le même ensemble de files d'attente système IBM MQ classes for JMS peut prendre en charge toutes les applications XMS et IBM MQ classes for JMS se connectant au gestionnaire de files d'attente.

b) Accordez à l'ID utilisateur associé à votre application le droits d'accès aux files d'attente système IBM MQ classes for JMS.

Pour plus d'informations sur les droits dont l'ID utilisateur a besoin, voir Utilisation de IBM MQ classes for JMS.

c) Pour un courtier IBM Integration Bus ou WebSphere Message Broker, créez et déployez un flux de messages pour servir la file d'attente sur laquelle les applications envoient les messages qu'elles publient.

Le flux de messages de base comprend un noeud de traitement de messages MQInput pour lire les messages publiés et un noeud de traitement de messages Publication pour publier les messages.

Pour des informations sur la création et le déploiement d'un flux de messages, voir la documentation du produit IBM Integration Bus ou WebSphere Message Broker disponible depuis la Page Web de la bibliothèque de documentation du produit IBM Integration Bus.

Cette étape n'est pas utile si un flux de messages approprié est déjà déployé sur le courtier.

Résultats

Vous pouvez à présent démarrer votre application.

Configuration d'un courtier pour une application utilisant une connexion en temps réel à un courtier

Avant de pouvoir exécuter une application utilisant une connexion en temps réel à un courtier, vous devez configurer ce dernier.

Avant de commencer

Avant de démarrer cette tâche, vous devez effectuer les étapes suivantes :

- Assurez-vous que votre application a accès à un courtier en cours d'exécution.
- Assurez-vous que votre application utilise une fabrique de connexions dont les propriétés sont correctement définies pour une connexion en temps réel à un courtier. Pour plus d'informations sur les propriétés d'une fabrique de connexions, voir Propriétés de ConnectionFactory.

Pourquoi et quand exécuter cette tâche

La configuration d'un courtier pour exécuter des applications XMS s'effectue de manière identique à la configuration d'un courtier pour exécuter des applications IBM MQ classes for JMS. Les étapes suivantes récapitulent la procédure à suivre :

Procédure

1. Créez et déployez un flux de messages pour lire des messages depuis le port TCP/IP sur lequel un courtier est à l'écoute et publiez les messages.

   Pour ce faire, vous pouvez procéder selon l'une des manières suivantes :

   - Créer un flux de messages qui contient un noeud de traitement de message **Real-timeOptimizedFlow**.
   - Créer un flux de messages qui contient un noeud de traitement de message **Real-timeInput** et un noeud de traitement de message Publication.

Vous devez configurer le noeud **Real-timeOptimizedFlow** ou **Real-timeInput** pour écouter sur le port utilisé pour les connexions en temps réel. Dans XMS, le numéro de port par défaut pour les connexions en temps réel est 1506.

Cette étape n'est pas utile si un flux de messages approprié est déjà déployé sur le courtier.

2. Si vous avez besoin que des messages soient livrés à votre application à l'aide de IBM MQ classes for JMS, configurez le courtier de manière à activer la multidiffusion. Configurez les rubriques pour lesquelles la multidiffusion doit être activée, en spécifiant une qualité de service fiable pour ces rubriques nécessitant un mode multidiffusion fiable.
3. Si votre application fournit un ID utilisateur et un mot de passe lorsqu'elle se connecte au courtier, et si vous souhaitez que le courtier authentifie votre application par l'intermédiaire de ces informations, configurez le serveur et le courtier de manière à activer une authentification simple par mot de passe de type telnet.

Résultats

Vous pouvez à présent démarrer votre application.

Configuration du bus d'intégration de services pour une application se connectant à WebSphere Application Server

Pour pouvoir exécuter une application qui se connecte à un bus d'intégration de services WebSphere Application Server service integration technologies, vous devez configurer l'intégration de services de la même façon que vous configurez le bus d'intégration de services pour exécuter des applications JMS utilisant le fournisseur de messagerie par défaut.

Avant de commencer

Avant de démarrer cette tâche, vous devez effectuer les étapes suivantes :

- Assurez-vous qu'un bus de messagerie est créé et que votre serveur est ajouté au bus en tant que membre de bus.
- Assurez-vous que votre application a accès à un bus d'intégration de services qui contient au moins un moteur de messagerie en cours d'exécution.
- Si le mode HTTP est requis, un canal de transport entrant de moteur de messagerie HTTP doit être défini. Par défaut, les canaux pour SSL et TCP sont définis au cours de l'installation du serveur.
- Assurez-vous que votre application utilise une fabrique de connexions dont les propriétés sont définies de manière appropriée pour la connexion au bus d'intégration de services à l'aide d'un serveur d'amorçage. Les informations minimales requises sont :
  - Le noeud final du fournisseur, qui décrit l'emplacement et le protocole à utiliser lors de la négociation d'une connexion au serveur de messagerie (c'est-à-dire, le serveur d'amorçage). Dans son format le plus simple, pour un serveur installé avec des paramètres par défaut, le noeud final peut être défini par le nom d'hôte du serveur.
  - Le nom du bus par lequel les messages sont envoyés.

  Pour plus d'informations sur les propriétés d'une fabrique de connexions, voir Propriétés de ConnectionFactory.

Pourquoi et quand exécuter cette tâche

Les espaces de rubrique ou de file d'attente dont vous avez besoin doivent être définis. Par défaut, un espace de rubrique appelé Default.Topic.Space est défini au cours de l'installation du serveur mais, si vous avez besoin d'espaces de rubrique supplémentaires, vous devez les créer vous-même. Vous n'avez pas besoin de prédéfinir des rubriques individuelles dans un espace de rubriques, car le serveur instancie de manière dynamique les rubriques individuelles si nécessaire.

Les étapes suivantes récapitulent la procédure à suivre.

Procédure

1. Créez les files d'attente dont votre application a besoin pour la messagerie point-à-point.
2. Créez les espaces de sujet supplémentaires dont votre application a besoin pour la messagerie de publication/abonnement.

Résultats

Vous pouvez à présent démarrer votre application.

Utilisation des modèles d'application XMS

Les exemples d'application XMS .NET fournissent une présentation des fonctions communes de chaque API. Vous pouvez les utiliser pour vérifier votre installation et votre configuration de serveur de messagerie, ou pour vous aider à construire vos propres applications.

Pourquoi et quand exécuter cette tâche

Si vous avez besoin d'aide pour créer vos propres applications, vous pouvez utiliser les modèles d'application comme base de départ. La version source et la version compilée sont fournies pour chaque application. Utilisez le code source exemple pour identifier les étapes clé permettant de créer chaque objet requis pour votre application (ConnectionFactory, Connection, Session, Destination, mais aussi un expéditeur et/ou un consommateur) et de définir les propriétés spécifiques dont vous avez besoin pour spécifier la manière dont l'application doit fonctionner. Pour plus d'informations, voir «Ecriture d'applications XMS .NET», à la page 648. Les exemples sont susceptibles de changer dans les éditions ultérieures de XMS.

Le tableau suivant présente les ensembles d'exemples d'application (un pour chaque API) fournis avec XMS.

Tableau 81. Exemples d'application pour XMS .NET

Nom du modèle	Description
SampleConsumerCS	Application consommateur de message qui prend les messages d'une file d'attente ou s'abonne à une rubrique.
SampleProducerCS	Application expéditeur de message qui fournit des messages à une file d'attente ou à une rubrique.
SampleConfigCS	Application de configuration que vous pouvez utiliser pour créer un référentiel d'objets gérés à partir de fichiers. L'application contient une fabrique de connexions et une destination pour vos paramètres de connexion spécifiques. Ce référentiel d'objets gérés peut ensuite être utilisé avec chaque modèle d'application consommateur et expéditeur.

Les modèles qui prennent en charge les mêmes fonctions dans les diverses API présentent des différences au niveau de la syntaxe.

- Les modèles d'application consommateur et expéditeur prennent en charge les fonctions suivantes :
  - Connexion à IBM MQ, IBM Integration Bus (via une connexion en temps réel à un courtier) et à un WebSphere Application Server service integration bus
  - Recherche dans le référentiel d'objets gérés via l'interface de contexte initial
  - Connexion aux files d'attente (IBM MQ et WebSphere Application Server service integration bus) et aux rubriques (IBM MQ, connexion en temps réel à un courtier, et à WebSphere Application Server service integration bus)
  - Messages de base, d'octets, de mappe, d'objet, de flux et texte
- Le modèle d'application consommateur de message prend en charge les modes de réception synchrone et asynchrone, et les instructions SQL Selector.

- Le modèle d'application expéditeur de message prend en charge les modes de livraison permanent et non permanent.

Les modèles peuvent fonctionner dans deux modes :

Mode simple
Vous pouvez exécuter le modèle avec un minimum d'entrée utilisateur.

Mode avancé
Vous pouvez personnaliser plus en profondeur la manière dont le modèle fonctionne.

Tous les modèles sont compatibles et peuvent fonctionner quelle que soit la langue.

IBM MQ prend en charge les applications .NET Core for XMS .NET dans les environnements Windows . IBM MQ classes for .NET Standard, y compris les exemples, sont installés par défaut dans le cadre de l'installation standard de IBM MQ .

IBM MQ prend également en charge .NET Core pour les applications dans les environnements Linux .

Les exemples d'application pour XMS .NET sont installés dans `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Pour plus d'informations, voir «Installation de IBM MQ classes for XMS .NET», à la page 634.

Exécution des modèles d'application .NET

Vous pouvez exécuter les modèles d'application .NET de manière interactive en mode simple ou avancé, ou de manière non interactive à l'aide des fichiers de réponse générés automatiquement ou personnalisés.

Avant de commencer

Avant d'exécuter un des modèles d'application fournis, vous devez commencer par configurer l'environnement du serveur de messagerie, de sorte que les applications puissent se connecter à un serveur. Voir «Configuration de l'environnement d'un serveur de messagerie», à la page 639.

Procédure

Pour exécuter un modèle d'application .NET, effectuez les étapes suivantes :

Conseil : Lorsque vous exécutez un exemple d'application, entrez? à tout moment pour obtenir de l'aide sur ce qu'il faut faire ensuite.

1. Sélectionnez le mode dans lequel vous voulez exécuter le modèle d'application.
   Entrez `Advanced` ou `Simple`.
2. Répondez aux questions.
   Pour sélectionner la valeur par défaut, qui apparaît entre crochets à la fin de la question, appuyez sur Entrée. Pour sélectionner une autre valeur, tapez celle-ci et appuyez sur Entrée.

   Voici un exemple de question :

   ```
   Enter connection type [wpm]:
   ```

   Dans ce cas, la valeur par défaut est `wpm` (connexion à un WebSphere Application Server service integration bus).

Résultats

Lorsque vous exécutez les modèles d'application, des fichiers de réponses sont générés automatiquement dans le répertoire de travail en cours. Les noms de fichier de réponses sont au format *`connection_type-sample_type`*`.rsp`; par exemple, `wpm-producer.rsp`. Si besoin, vous pouvez

utiliser les fichiers de réponses générés pour exécuter à nouveau les modèles d'application avec les mêmes options, de sorte que vous n'avez pas besoin de les entrer une nouvelle fois.

Tâches associées

Génération des modèles d'application .NET
Lorsque vous générez un modèle d'application .NET, une version exécutable du modèle choisi est créée.

Génération de vos propres applications
Vous pouvez générer vos propres applications comme vous générez les modèles d'application.

Génération des modèles d'application .NET

Lorsque vous générez un modèle d'application .NET, une version exécutable du modèle choisi est créée.

Avant de commencer

Installez le compilateur approprié. Cette tâche suppose que vous avez installé Microsoft Visual Studio 2012 et que vous savez l'utiliser.

Procédure

Pour générer un modèle d'application .NET, effectuez les étapes suivantes :

1. Cliquez sur le fichier de solution `Samples.sln` fourni avec les modèles .NET.
2. Cliquez avec le bouton droit de la souris sur la solution `Exemples` dans la fenêtre Explorateur de solutions et sélectionnez **Générer**.

Résultats

Un programme exécutable est créé dans le sous-dossier approprié du modèle, `bin/Debug` ou `bin/Release` selon la configuration choisie. Ce programme a le même nom que le dossier, avec le suffixe CS. Par exemple, si vous générez la version C# du modèle d'application d'expéditeur de message, le fichier `SampleProducerCS.exe` est créé dans le dossier `SampleProducer`.

Tâches associées

Exécution des modèles d'application .NET
Vous pouvez exécuter les modèles d'application .NET de manière interactive en mode simple ou avancé, ou de manière non interactive à l'aide des fichiers de réponse générés automatiquement ou personnalisés.

Génération de vos propres applications
Vous pouvez générer vos propres applications comme vous générez les modèles d'application.

«Génération de vos propres applications», à la page 647
Vous pouvez générer vos propres applications comme vous générez les modèles d'application.

Génération de vos propres applications

Vous pouvez générer vos propres applications comme vous générez les modèles d'application.

Avant de commencer

Installez le compilateur approprié. Cette tâche suppose que vous avez installé Microsoft Visual Studio 2012 et que vous savez l'utiliser.

Procédure

- Générez votre application .NET , comme décrit dans «Génération des modèles d'application .NET», à la page 647.

  Pour plus de conseils sur la manière de générer vos propres applications, utilisez les fichiers makefile fournis avec chaque modèle d'application.

Conseil : Pour vous aider à diagnostiquer les problèmes en cas de panne, vous pouvez compiler vos applications avec les symboles inclus.

Tâches associées

Exécution des modèles d'application .NET
Vous pouvez exécuter les modèles d'application .NET de manière interactive en mode simple ou avancé, ou de manière non interactive à l'aide des fichiers de réponse générés automatiquement ou personnalisés.

Génération des modèles d'application .NET
Lorsque vous générez un modèle d'application .NET, une version exécutable du modèle choisi est créée.

Ecriture d'applications XMS .NET

Cette section fournit des informations pour vous aider à écrire des applications XMS .NET , y compris des informations sur les propriétés, les types de données et le traitement des erreurs.

Avant de commencer

Deprecated V 9.4.0 V 9.4.0 Depuis IBM MQ 9.4.0, dans IBM MQ classes for XMS .NET, les méthodes WriteObject(), ReadObject(), CreateObjectMessage () et les classes ObjectMessage et XmsObjectMessageImpl utilisées pour la sérialisation et la désérialisation des données sont obsolètes.

V 9.4.0 V 9.4.0 Removed La bibliothèque client XMS .NET générée à l'aide de.NET Standard 2.0, qui a été dépréciée dans la IBM MQ 9.3.1, a été retirée du produit à l'adresse IBM MQ 9.4.0.

Depuis la IBM MQ 9.2.0, le nombre de bibliothèques de liens dynamiques XMS .NET a été réduit de manière significative, pour atteindre un total de cinq. Les cinq bibliothèques de liaison dynamique sont les suivantes :

- `IBM.XMS.dll` - inclut tous les messages dans la langue nationale
- `IBM.XMS.Comms.RMM.dll`
- Trois bibliothèques de liaison dynamique de règle :
  - `policy.8.0.IBM.XMS.dll`
  - `policy.9.0.IBM.XMS.dll`
  - `policy.9.1.IBM.XMS.dll`

Dans XMS .NET, toutes les chaînes sont transmises à l'aide de la chaîne .NET native. Celle-ci utilisant un codage fixe, aucune information supplémentaire n'est requise pour son interprétation. Par conséquent, la propriété XMSC_CLIENT_CCSID n'est pas requise pour les applications XMS .NET .

Pourquoi et quand exécuter cette tâche

Cette section contient les rubriques suivantes :

- «Suppression d'objet», à la page 666
- «Types de données pour XMS.NET», à la page 666
- «Types primitifs XMS», à la page 667
- «Conversion implicite d'une valeur de propriété d'un type de données à un autre», à la page 668
- «Itérateurs», à la page 670
- «Traitement des erreurs dans XMS .NET», à la page 671
- «Utilisation des programmes d'écoute des messages et des exceptions dans .NET», à la page 671
- «Reconnexion automatique du client IBM MQ via XMS», à la page 672

Opérations gérées et non gérées dans .NET

Le code géré est exécuté exclusivement dans la routine d'exécution du langage commun .NET ; il est totalement dépendant des services fournis par cette routine. Une application est classée comme non gérée si une partie de celle-ci exécute ou appelle des services externes à la routine d'exécution du langage commun .NET.

Certaines fonctionnalités avancées ne sont pas prises en charge dans l'environnement géré .NET.

Si votre application requiert une fonctionnalité qui n'est pas prise en charge dans l'environnement géré, vous pouvez modifier l'application de sorte qu'elle utilise l'environnement non géré sans y apporter de modifications substantielles. Toutefois, vous devez prendre en compte le fait que la pile XMS utilise des codes non gérés lorsque cette sélection est effectuée.

Connexions à un gestionnaire de files d'attente IBM MQ

Les connexions gérées à WMQ_CM_CLIENT ne prennent pas en charge les communications non TCP et la compression de canal. Toutefois, ces connexions peuvent être prises en charge via l'utilisation d'une connexion non gérée (WMQ_CM_CLIENT_UNMANAGED). Pour plus d'informations, voir «Développement d'applications .NET», à la page 567.

Si vous créez une fabrique de connexions à partir d'un objet administré dans un environnement non géré, vous devez modifier manuellement la valeur du mode de connexion à XMSC_WMQ_CM_CLIENT_UNMANAGED.

Connexions à un moteur de messagerie de bus d'intégration de services WebSphere Application Server

Les connexions à un moteur de messagerie de bus d'intégration de services WebSphere Application Server nécessitant l'utilisation du protocole SSL (HTTPS y compris) ne sont pas prises en charge en tant que code géré.

Windows Utilisation du canevas de projet IBM MQ XMS .NET

Le client IBM MQ XMS .NET vous permet d'utiliser un modèle de projet pour vous aider à développer vos applications XMS .NET Core .

Avant de commencer

Vous devez disposer de Microsoft Visual Studio 2017ou version ultérieure et de .NET Core 2.1 sur votre système.

Vous devez copier le modèle XMS .NET à partir du

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip
```

vers le répertoire

&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates

répertoire, où:

- *& MQ_INSTALL_ROOT* est le répertoire racine de votre installation
- *& USER_HOME_DIRECTORY* est votre répertoire de base.

Vous devez arrêter et redémarrer Microsoft Visual Studio pour récupérer le modèle.

Pourquoi et quand exécuter cette tâche

Le canevas de projet XMS .NET inclut du code commun que vous pouvez utiliser pour vous aider à développer vos applications. Avec le code intégré, vous pouvez vous connecter au gestionnaire de files d'attente IBM MQ et effectuer une opération d'insertion ou d'extraction en modifiant simplement les propriétés dans le code intégré.

Procédure

1. Ouvrez Microsoft Visual Studio.
2. Cliquez sur **Fichier**, puis sur **Nouveau** , puis sur **Projet**.
3. Dans la *fenêtre Créer un projet*, sélectionnez `IBM XMS .NET Client App (.NET Core)` et cliquez sur **Suivant**.
4. Dans la fenêtre *Configurer votre nouveau projet* , modifiez le *nom de projet* de votre projet si vous le souhaitez, puis cliquez sur **Créer** pour créer le projet XMS .NET .

   `XMSDotnetApp.cs` est le fichier créé avec le fichier de projet. Ce fichier contient le code qui se connecte au gestionnaire de files d'attente et effectue une opération d'envoi et de réception.

   Les propriétés de connexion sont définies sur les valeurs par défaut:

   - WMQ_CONNECTION_NAME_LIST est défini sur *localhost (1414)*
   - XMSC.WMQ_CHANNEL est défini sur *DOTNET.SVRCONN*

   La file d'attente est définie sur *Q1*et vous pouvez modifier ces propriétés en conséquence.
5. Compilez et exécutez l'application.

Concepts associés

Composants et fonctions d'IBM MQ

Environnement d'application .NET - Windows uniquement

Le modèle d'unités d'exécution

Des règles générales déterminent comment une application à unités d'exécution multiples peut utiliser des objets XMS.

- Seuls les objets des types suivants peuvent être utilisés simultanément sur des unités d'exécution différentes :
  - ConnectionFactory
  - Connexion
  - ConnectionMetaData
  - Destination
- Un objet Session peut être utilisé sur une seule unité d'exécution à un moment donné.

Les exceptions à ces règles sont indiquées par des entrées libellées "Contexte d'unité d'exécution" dans les définitions d'interface des méthodes dans Référence d'IBM Message Service Client for .NET.

Propriétés dans XMS.NET

Une application .NET utilise les méthodes dans l'interface PropertyContext pour extraire et définir les propriétés des objets. La gestion des propriétés non existantes dans XMS .NET est globalement cohérente avec la spécification JMS, mais aussi avec les implémentations C et C++ de XMS.

Propriétés XMS .NET et leurs valeurs

L'interface PropertyContext encapsule les méthodes pour extraire et définir les propriétés. Ces méthodes sont héritées, directement ou indirectement, par les classes suivantes :

- BytesMessage
- connexion
- ConnectionFactory
- ConnectionMetaData
- Destination
- MapMessage
- Message
- MessageConsumer
- MessageProducer
- ObjectMessage
- QueueBrowser
- Session
- StreamMessage
- TextMessage

Si une application définit la valeur d'une propriété, la nouvelle valeur remplace toute valeur existante. Pour plus d'informations sur les propriétés XMS , voir Propriétés des objets XMS.

Dans XMS, par souci de simplification, les noms et les valeurs de propriété XMS sont prédéfinis en tant que constantes publiques dans une structure appelée XMSC. Les noms de ces constantes se présentent sous la forme XMSC.*constante*; par exemple, XMSC.USERID (constante de nom de propriété) et XMSC.DELIVERY_AS_APP (constante de valeur).

De plus, vous pouvez accéder aux constantes IBM MQ par l'intermédiaire de la structure IBM.XMS.MQC. Si l'espace de nom IBM.XMS est déjà importé, vous pouvez accéder aux valeurs de ces propriétés au format MQC.*constante*. Par exemple, MQC.MQRO_COA_WITH_FULL_DATA.

Si vous disposez d'une application hybride qui utilise à la fois les classes XMS .NET et IBM MQ pour .NET et qui importe les deux IBM.XMS et IBM.WMQ , puis vous devez qualifier complètement l'espace de nom de structure MQC pour vous assurer que chaque occurrence est unique.

Remarque : Certaines fonctionnalités avancées ne sont pas prises en charge dans l'environnement .NET géré. Pour plus d'informations, voir «Opérations gérées et non gérées dans .NET», à la page 649.

Traitement des propriétés inexistantes dans XMS .NET

Dans JMS, l'accès à une propriété non existante peut entraîner une exception système Java lorsqu'une méthode essaie de convertir la valeur non existante (null) dans le type requis. Si une propriété n'existe pas, les exceptions suivantes se produisent :

- getStringProperty et getObjectProperty renvoie une valeur null
- getBooleanProperty renvoie false car Boolean.valueOf(null) renvoie false
- getIntProperty etc. renvoie java.lang.NumberFormatException car Integer.valueOf(null) émet une exception

Si une propriété n'existe pas dans XMS .NET, les exceptions suivantes se produisent:

- GetStringProperty et GetObjectProperty (et GetBytesProperty) renvoient null (identique à Java)
- GetBooleanProperty émet System.NullReferenceException
- GetIntProperty etc. émet System.NullReferenceException

Cette implémentation est différente de Java, mais elle est globalement cohérente avec la spécification JMS et avec les interfaces XMS C et C++. Tout comme l'implémentation Java, XMS .NET propage les exceptions de l'appel System.Convert vers l'appelant. Toutefois, contrairement à Java, XMS émet explicitement NullReferenceExceptions plutôt que d'utiliser le comportement natif de l'infrastructure .NET en transmettant la valeur null aux routines de conversion système. Si votre application définit une chaîne telle que "abc" pour une propriété et appelle GetIntProperty, l'exception System.FormatException émise par Convert.ToInt32("abc") est propagée à l'appelant, ce qui est cohérent avec Java. L'exception MessageFormatException est émise uniquement si les types utilisés pour setProperty et getProperty ne sont pas compatibles. Ce comportement est également cohérent avec Java.

Objets ConnectionFactories et Connection

Un objet ConnectionFactory fournit un modèle utilisé par une application pour créer un objet Connection. L'application utilise l'objet Connection pour créer un objet Session.

Pour .NET, l'application XMS utilise tout d'abord un objet XMSFactoryFactory pour extraire une référence à un objet ConnectionFactory approprié pour le type de protocole requis. Cet objet ConnectionFactory peut ensuite produire des connexions uniquement pour ce type de protocole.

L'application XMS peut créer plusieurs connexions, et une application à plusieurs unités d'exécution peut utiliser un objet Connection unique simultanément sur plusieurs unités d'exécution. Un objet Connection encapsule une connexion de communications entre une application et un serveur de messagerie.

Une connexion a plusieurs objectifs :

- Lorsqu'une application crée une connexion, elle peut être authentifiée.
- Une application peut associer un identificateur de client unique à une connexion. L'identificateur de client est utilisé pour la prise en charge d'abonnements durables dans le domaine de publication/abonnement. L'identificateur de client peut être défini de deux manières :

  La meilleure manière d'affecter un identificateur de client de connexions consiste à configurer un objet ConnectionFactory propre au client et à l'affecter de manière transparente à la connexion qu'il crée.

  Une autre manière d'affecter un identificateur de client est d'utiliser une valeur propre au client définie sur l'objet Connection. Cette valeur ne se substitue pas à l'identificateur qui a été configuré de manière administrative. Elle est fournie au cas où il n'existe aucun identificateur spécifié de manière administrative. Si un identificateur spécifié de manière administrative existe, une tentative de substitution par une valeur propre au fournisseur génère l'émission d'une exception. Si une application définit un identificateur de manière explicite, elle doit le faire immédiatement après avoir créé la connexion ou avant toute autre action sur la connexion ; sinon, une exception est émise.

Généralement, l'application XMS crée une connexion, une ou plusieurs sessions, des expéditeurs de message et des consommateurs de message.

La création d'une connexion est relativement coûteuse en termes de ressources système, car cela implique d'établir une connexion de communications, mais aussi éventuellement d'authentifier l'application.

Mode démarré et arrêt d'une connexion

Une connexion peut fonctionner en mode démarré ou arrêté.

Lorsqu'une application crée une connexion, celle-ci est en mode arrêté. Dans ce cas, l'application peut initialiser des sessions et envoyer des messages, mais ne peut pas en recevoir, que ce soit en mode synchrone ou asynchrone.

Une application peut démarrer une connexion en appelant la méthode `Start Connection`. Lorsque la connexion est en mode démarré, l'application peut envoyer et recevoir des messages. L'application peut ensuite arrêter et redémarrer la connexion en appelant les méthodes Stop Connection et `Start Connection`.

Fermeture de connexion

Une application ferme une connexion en appelant la méthode Close Connection. Lorsqu'une application ferme une connexion, XMS effectue les actions suivantes :

- Il ferme toutes les sessions associées à la connexion et supprime certains objets associés à ces sessions. Pour plus d'informations sur les objets supprimés, voir «Suppression d'objet», à la page 666. Dans le même temps, XMS annule toutes les transactions en cours dans les sessions.
- Il met fin à la connexion de communications avec le serveur de messagerie.
- Il libère la mémoire et les autres ressources internes utilisées par la connexion.

XMS n'accuse pas réception des messages pour lesquels il n'est pas parvenu à effectuer cette opération avant la fermeture de la connexion. Pour plus d'informations sur l'accusé de réception des messages, voir «Accusé de réception de message», à la page 655.

Gestion des exceptions

Les exceptions XMS .NET sont toutes dérivées de System.Exception. Pour plus d'informations, voir «Traitement des erreurs dans XMS .NET», à la page 671.

Connexion à un bus d'intégration de services

Une application XMS peut se connecter à un bus d'intégration de services WebSphere Application Server via une connexion TCP/IP directe ou via HTTP sur TCP/IP.

Le protocole HTTP peut être utilisé dans les situations où une connexion TCP/IP directe n'est pas possible. Une situation courante est la communication via un pare-feu, par exemple lorsque deux entreprises échangent des messages. L'utilisation de HTTP pour communiquer via un pare-feu est souvent appelée *tunnellisation HTTP*. Toutefois, cette méthode est par nature plus lente que l'utilisation d'une connexion TCP/IP directe, car les en-têtes HTTP s'ajoutent de manière significative à la quantité de données transférées ; de plus, le protocole HTTP nécessite plus de flux de communication que TCP/IP.

Pour créer une connexion TCP/IP, une application peut utiliser une fabrique de connexions dont la propriétéXMSC_WPM_TARGET_TRANSPORT_CHAIN est définie sur XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Il s'agit de la valeur par défaut de la propriété. Si la création de la connexion abouti, la propriétéXMSC_WPM_CONNECTION_PROTOCOL de la connexion est définie sur XMSC_WPM_CP_TCP.

Pour créer une connexion qui utilise le protocole HTTP, une application doit utiliser une fabrique de connexions dont la propriété XMSC_WPM_TARGET_TRANSPORT_CHAIN est définie sur le nom d'une chaîne de transport de communications entrante, c'est-à-dire configurée pour utiliser un canal de transport HTTP. Si la création de la connexion abouti, la propriété XMSC_WPM_CONNECTION_PROTOCOL de la connexion est définie sur XMSC_WPM_CP_HTTP. Pour des informations sur la configuration des chaînes de transport, voir Configuration des chaînes de transport dans la documentation du produit WebSphere Application Server.

Une application a un choix similaire de protocoles de communication pour une connexion à un serveur d'amorçage. La propriété XMSC_WPM_PROVIDER_ENDPOINTS d'une fabrique de connexions est une séquence de une ou plusieurs adresses de noeud final de serveurs d'amorçage. Le composant de chaîne de transport d'amorçage de chaque adresse de noeud final peut être XMSC_WPM_BOOTSTRAP_TCP, pour une connexion TCP/IP à un serveur d'amorçage, ou XMSC_WPM_BOOTSTRAP_HTTP pour une connexion via HTTP.

Sessions

Une session est un contexte à unité d'exécution unique pour l'envoi et la réception de messages.

Une application peut utiliser une session pour créer des messages, des expéditeurs de messages, des consommateurs de messages, des navigateurs de files d'attente et des destinations temporaires. Une application peut également utiliser une session pour exécuter des transactions locales.

Une application peut créer des sessions multiples, où chaque session produit et consomme des messages indépendamment des autres sessions. Si deux consommateurs de message dans des sessions distinctes (ou même dans la même session) s'abonnent à la même rubrique, chacun reçoit une copie de tout message publié sur ce sujet.

Contrairement à un objet Connexion, un objet Session ne peut pas être utilisé simultanément sur plusieurs unités d'exécution. Seule la méthode Close Session d'un objet Session peut être appelée à partir d'une unité d'exécution autre que celle utilisée au même moment par l'objet Session. La méthode Close Session met fin à une session et libère les ressources système qui y étaient allouées.

Si une application doit traiter des messages simultanément sur plusieurs unités d'exécution, l'application doit créer une session sur chaque unité d'exécution, puis utiliser cette session pour toute opération d'envoi ou de réception au sein de cette unité d'exécution.

Sessions transactionnelles

Les applications XMS peuvent exécuter des transactions locales. Une *transaction locale* est une transaction qui implique uniquement les modifications apportées aux ressources du gestionnaire de files d'attente ou du bus d'intégration de services auquel l'application est connectée.

Les informations de cette rubrique sont pertinentes uniquement si une application se connecte à un gestionnaire de files d'attente IBM MQ ou à un bus d'intégration de services WebSphere Application Server. Elles ne le sont pas pour une connexion en temps réel à un courtier.

Pour exécuter des transactions locales, une application doit tout d'abord créer une session transactionnelle en appelant la méthode Create Session d'un objet Connection, et en spécifiant sous forme de paramètre que la session est transactionnelle. Par la suite, tous les messages envoyés et reçus au cours de la session sont groupés dans une séquence de transactions. Une transaction se termine lorsque l'application valide ou annule les messages qu'il a envoyés et reçus depuis le début de la transaction.

Pour valider une transaction, une application appelle la méthode Commit de l'objet Session. Lorsqu'une transaction est validée, tous les messages envoyés au cours de cette transaction deviennent disponibles pour la distribution à d'autres applications ; de même, un accusé de réception est envoyé pour tous les messages reçus au cours de cette transaction, de sorte que le serveur de messagerie n'essaie pas de les distribuer à nouveau à l'application. Dans le domaine point-à-point, le serveur de messagerie retire également les messages reçus de leurs files d'attente.

Pour annuler une transaction, une application appelle la méthode Rollback de l'objet Session. Lorsqu'une transaction est annulée, tous les messages envoyés au cours de cette transaction sont supprimés du serveur de messagerie, et tous les messages reçus au cours de cette transaction sont à nouveau disponible pour la distribution. Dans le domaine point-à-point, les messages reçus sont replacés dans leurs filesd'attente et les autres applications peuvent à nouveau les voir.

Une nouvelle transaction commence automatiquement lorsqu'une application crée une session transactionnelle ou appelle la méthode Commit ou Rollback. Une session transactionnelle a donc toujours une transaction active.

Lorsqu'une application ferme une session transactionnelle, une annulation implicite se produit. Lorsqu'une application ferme une connexion, une annulation implicite de toutes les sessions transactionnelles de la connexion se produit.

Une transaction est entièrement contenue dans une session transactionnelle. Une transaction ne peut pas s'étendre à d'autres sessions. Cela signifie qu'une application ne peut pas envoyer ou recevoir des messages dans plusieurs sessions transactionnelles, puis valider ou annuler toutes ces actions comme une transaction unique.

Concepts associés

Accusé de réception de message
Chaque session non transactionnelle possède un mode d'accusé de réception qui détermine la façon dont l'application accuse réception des messages. Trois modes d'accusé de réception sont disponibles, et le choix du mode d'accusé de réception a un impact sur la conception de l'application.

Distribution des messages
XMS prend en charge les modes persistant et non persistant de distribution des messages, ainsi que la distribution asynchrone et synchrone des messages.

Transactions IBM MQ XA gérées via XMS
Les transactions IBM MQ XA gérées peuvent être utilisées via XMS.

Accusé de réception de message

Chaque session non transactionnelle possède un mode d'accusé de réception qui détermine la façon dont l'application accuse réception des messages. Trois modes d'accusé de réception sont disponibles, et le choix du mode d'accusé de réception a un impact sur la conception de l'application.

Remarque : Cette rubrique est pertinente uniquement si une application se connecte à un gestionnaire de files d'attente IBM MQ ou à un bus d'intégration de services WebSphere Application Server . Elles ne le sont pas pour une connexion en temps réel à un courtier.

XMS utilise le même mécanisme d'accusé de réception de messages que JMS.

Si une session n'est pas transactionnelle, la façon dont l'application accuse réception des messages dépend du mode d'accusé de réception de la session. Il existe trois modes d'accusé de réception:

XMSC_AUTO_ACKNOWLEDGE
La session accuse automatiquement réception de chaque message reçu par l'application.

Si des messages sont distribués de manière synchrone à l'application, la session accuse réception chaque fois qu'un appel Receive aboutit. Si l'application reçoit un message, mais qu'un incident empêche l'émission de l'accusé de réception, le message est à nouveau disponible pour la distribution. L'application doit donc être capable de gérer un message redistribué.

XMSC_DUPS_OK_ACKNOWLEDGE
La session accuse réception des messages reçus par l'application au moment de leur sélection.

Ce mode d'accusé de réception réduit le volume de travail que la session doit accomplir, mais si un incident empêche d'accuser réception du message, il se peut que plusieurs messages deviennent disponibles pour une nouvelle distribution. L'application doit donc être capable de gérer les messages redistribués.

XMSC_CLIENT_ACKNOWLEDGE
L'application accuse réception des messages qu'elle reçoit en appelant la méthode Acknowledge de la classe Message.

L'application peut accuser réception de chaque message individuellement ou recevoir un lot de messages et appeler la méthode Acknowledge seulement pour le dernier message reçu. Dans ce cas, l'accusé de réception est émis pour tous les messages reçus depuis l'appel précédent à cette méthode.

Conjointement avec ces modes d'accusé de réception, une application peut arrêter et redémarrer la distribution des messages dans une session en appelant la méthode Recover de la classe Session. Les messages dont l'accusé de réception n'a pas été envoyé sont redistribués. Toutefois, il est possible qu'ils ne soient pas redistribués dans le même ordre que la fois précédente. Entre temps, des messages de priorité plus élevée peuvent être arrivés, et certains messages originaux peuvent avoir expiré. Dans le domaine point-à-point, certains messages originaux peuvent avoir été consommés par une autre application.

Une application peut déterminer si un message est en cours de redistribution en examinant le contenu de la zone d'en-tête JMSRedelivered de celui-ci. L'application effectue cette opération en appelant la méthode Get JMSRedelivered de la classe Message.

Concepts associés

Sessions transactionnelles
Les applications XMS peuvent exécuter des transactions locales. Une *transaction locale* est une transaction qui implique uniquement les modifications apportées aux ressources du gestionnaire de files d'attente ou du bus d'intégration de services auquel l'application est connectée.

Distribution des messages
XMS prend en charge les modes persistant et non persistant de distribution des messages, ainsi que la distribution asynchrone et synchrone des messages.

Transactions IBM MQ XA gérées via XMS
Les transactions IBM MQ XA gérées peuvent être utilisées via XMS.

Distribution des messages

XMS prend en charge les modes persistant et non persistant de distribution des messages, ainsi que la distribution asynchrone et synchrone des messages.

Modes de distribution des messages

XMS prend en charge deux modes de distribution des messages:

- Persistante

  Les messages persistants sont distribués une seule fois. Un serveur de messagerie prend des précautions spécifiques, telles que la consignation des messages, pour s'assurer que les messages persistants ne sont pas perdus au cours du transfert, même en cas d'arrêt anormal.

- Non persistant

  Les messages non persistants ne sont distribués qu'une seule fois. Les messages non persistants sont moins fiables que les messages persistants car ils peuvent être perdus au cours du transfert en cas d'arrêt anormal.

Le choix du mode de livraison est un compromis entre les performances et la fiabilité. Les messages non persistants sont généralement transportés plus rapidement que les messages persistants.

une distribution asynchrone de messages

XMS utilise une unité d'exécution pour gérer les distributions asynchrones de messages pour une session. Cela signifie qu'une seule fonction de programme d'écoute de message ou une méthode `onMessage()` peut être exécutée à la fois.

Si, dans une session, plusieurs consommateurs de message reçoivent des messages de façon asynchrone et qu'une fonction de programme d'écoute de message ou une méthode `onMessage()` distribue un message à un consommateur, les consommateurs qui attendent le même message doivent patienter. Les autres messages qui attendent d'être distribués dans la session doivent également patienter.

Si une application requiert une distribution simultanée de messages, vous devez créer plusieurs sessions de sorte queXMS utilise plusieurs unités d'exécution pour gérer la distribution asynchrone. De cette manière, plusieurs fonctions de programme d'écoute de message ou plusieurs méthodes `onMessage()` peuvent être exécutées simultanément.

Une session n'est pas rendue asynchrone par l'affectation d'un programme d'écoute de message à un consommateur. Une session devient asynchrone uniquement lorsque la méthode `Connection.Start` est appelée. Tous les appels synchrones sont autorisés jusqu'à ce que la méthode `Connection.Start` soit appelée. La distribution des messages aux consommateurs commence lorsque le `Connection.Start` est appelé.

Les appels synchrones, par exemple une création de consommateur ou de fournisseur, doivent être faits sur une session asynchrone et `Connection.Stop` doit être appelée. Une session peut être reprise par appel de la méthode `Connection.Start` pour lancer la distribution des messages. La seule exception est l'unité d'exécution de distribution de messages Session, qui est l'unité d'exécution en charge de la

distribution des messages à la fonction de rappel. Cette unité d'exécution peut lancer n'importe quel appel sur une session (à l'exception de l'appel Close) dans la fonction de rappel de message.

Remarque : En mode non géré, l'appel MQDISC dans une fonction de rappel n'est pas pris en charge par le client IBM MQ .NET. Ainsi, l'application client ne peut pas créer ou fermer des sessions dans le rappel MessageListener en mode de réception asynchrone. Créez et supprimez la session en dehors de la méthode `MessageListener`.

Distribution synchrone des messages

Les messages sont distribués de manière synchrone à une application lorsque celle-ci utilise les méthodes Receive des objets MessageConsumer.

Les méthodes Receive permettent à une application d'attendre un message indéfiniment ou pendant une période de temps spécifiée. Si une application ne veut pas attendre de message, elle peut utiliser la méthode Receive with No Wait.

Concepts associés

Sessions transactionnelles
Les applications XMS peuvent exécuter des transactions locales. Une *transaction locale* est une transaction qui implique uniquement les modifications apportées aux ressources du gestionnaire de files d'attente ou du bus d'intégration de services auquel l'application est connectée.

Accusé de réception de message
Chaque session non transactionnelle possède un mode d'accusé de réception qui détermine la façon dont l'application accuse réception des messages. Trois modes d'accusé de réception sont disponibles, et le choix du mode d'accusé de réception a un impact sur la conception de l'application.

Transactions IBM MQ XA gérées via XMS
Les transactions IBM MQ XA gérées peuvent être utilisées via XMS.

Transactions IBM MQ XA gérées via XMS

Les transactions IBM MQ XA gérées peuvent être utilisées via XMS.

Pour utiliser des transactions XA via XMS, une session transactionnelle doit être créée. Lorsque la transaction XA est utilisée, le contrôle des transactions s'effectue via des transactions globales DTC (Distributed Transaction Coordinator) et non via des sessions XMS . Lors de l'utilisation de transactions XA, `Session.commit` ou `Session.rollback` ne peut pas être émis sur la session XMS . A la place, utilisez la méthode DTC `Transscope.Commit` ou `Transscope.Rollback` pour valider ou annuler les transactions. Si une session est utilisée pour une transaction XA, le fournisseur ou le consommateur créé à l'aide de la session doit faire partie de la transaction XA. Ils ne peuvent pas être utilisés pour des opérations externes à la transaction XA. Ils ne peuvent pas être utilisés pour des opérations telles que `Producer.send` or `Consumer.receive` en dehors de la transaction XA.

Un objet d'exception `IllegalStateException` est émis si:

- Une session transactionnelle XA est utilisée pour `Session.commit` ou `Session.rollback`.
- Les objets fournisseurs ou consommateurs utilisés une fois dans une session transactionnelle XA sont utilisés hors de la portée de la transaction XA.

Les transactions XA ne sont pas prises en charge dans les consommateurs asynchrones.

Remarque :

1. Une fermeture peut se produire sur l'objet `Producer`, `Consumer`, `Session` ou `Connection` avant la validation de la transaction XA. Dans ce cas, les messages de la transaction sont annulés. De même, si la connexion est interrompue avant la validation de la transaction XA, tous les messages de la transaction sont annulés. Pour un objet `Producer`, une annulation signifie que les messages ne sont pas placés en file d'attente. Pour un objet `Consumer`, une annulation signifie que les messages restent en file d'attente.

2. Si un objet `Producer` place un message avec `TimeToLive` dans `TransactionScope` et qu'un `commit` est émis une fois que le temps est écoulé, le message peut expirer avant que le `commit` soit émis. Dans ce cas, le message n'est pas disponible pour les objets `Consumer`.
3. Les objets `Session` ne sont pas pris en charge entre unités d'exécution. L'utilisation de transactions avec des objets `Session` partagés entre unités d'exécution n'est pas prise en charge.

Concepts associés

Sessions transactionnelles
Les applications XMS peuvent exécuter des transactions locales. Une *transaction locale* est une transaction qui implique uniquement les modifications apportées aux ressources du gestionnaire de files d'attente ou du bus d'intégration de services auquel l'application est connectée.

Accusé de réception de message
Chaque session non transactionnelle possède un mode d'accusé de réception qui détermine la façon dont l'application accuse réception des messages. Trois modes d'accusé de réception sont disponibles, et le choix du mode d'accusé de réception a un impact sur la conception de l'application.

Distribution des messages
XMS prend en charge les modes persistant et non persistant de distribution des messages, ainsi que la distribution asynchrone et synchrone des messages.

Destinations

Une application XMS utilise un objet Destination pour spécifier la destination des messages envoyés et la source des messages reçus.

Une application XMS peut créer un objet Destination au moment de l'exécution, ou obtenir une destination prédéfinie à partir du référentiel des objets gérés.

Dans une fabrique de connexions, la manière la plus souple pour une application XMS de spécifier une destination consiste à la définir en tant qu'objet géré. Ainsi, des applications écrites en langage C, C++ et .NET et des applications Java peuvent partager des définitions de destination. Les propriétés d'objets Destination gérés peuvent être changées sans modification du code.

Destinations dans .NET

Dans .NET, les destinations sont créées en fonction du type de protocole et peuvent être utilisées uniquement sur le type de protocole pour lequel elles ont été créées. Deux méthodes sont fournies pour créer des destinations, l'une pour les rubriques et l'autre pour les files d'attente:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Ces méthodes sont disponibles sur les deux objets suivants dans l'API .NET :

- ISession
- XMSFactoryFactory

Dans les deux cas, ces méthodes peuvent accepter une chaîne de style URI, qui peut inclure des paramètres, au format suivant:

```
"topic://some/topic/name?priority=5"
```

Ces méthodes acceptent également seulement un nom de destination, c'est-à-dire un nom sans préfixe topic:// ou queue:// ni paramètre. Ainsi, la chaîne de style identificateur URI

```
CreateTopic("topic://some/topic/name");
```

produit le même résultat que le nom de destination suivant :

```
CreateTopic("some/topic/name");
```

Pour plus d'informations, voir IDestination.

Comme pour le bus d'intégration de services WebSphere Application Server JMS, les rubriques peuvent également être spécifiées dans un format abrégé, qui inclut les éléments *topicname* et *topicspace*, sans paramètre :

```
CreateTopic("topicspace:topicname");
```

identificateur URI de rubrique

L'identificateur URI (Uniform Resource Identifier) de rubrique spécifie le nom de la rubrique ; il peut également en spécifier une ou plusieurs propriétés.

L'identificateur URI d'une rubrique commence par la séquence topic://, suivie du nom de la rubrique et, éventuellement, d'une liste de paires nom-valeur qui définissent les autres propriétés de la rubrique. Un nom de rubrique ne peut pas être vide.

Voici un exemple dans un fragment de code .NET :

```
topic = session.CreateTopic("topic://Sport/Football/Results?multicast=7");
```

Pour plus d'informations sur les propriétés d'une rubrique, y compris le nom et les valeurs valides que vous pouvez utiliser dans un URI, voir Propriétés de destination.

Lorsque vous spécifiez un identificateur URI de rubrique, vous pouvez utiliser des caractères génériques. La syntaxe de ces caractères génériques dépend du type de connexion et de la version du courtier ; l'option suivante est disponible:

- Bus d'intégration de services WebSphere Application Server

Bus d'intégration de services WebSphere Application Server

Le bus d'intégration de services WebSphere Application Server utilise les caractères génériques suivants :

* pour tous les caractères à un niveau dans la hiérarchie
// pour 0 niveau ou plus
//. pour faire correspondre 0 ou plusieurs niveaux (à la fin d'une expression Topic)

Le Tableau 82, à la page 659 contient quelques exemples d'utilisation de ce schéma de caractères génériques.

Tableau 82. Exemples d'URI utilisant le schéma de caractères génériques pour le bus d'intégration de services WebSphere Application Server

Identificateur URI	Correspondance	Exemples
"topic://Sport/*ball/Résultats"	Toutes les rubriques avec un nom de niveau hiérarchique unique se terminant par "ball" entre Sport et Résultats	"topic://Sport/Football/Résultats" et "topic://Sport/Netball/Résultats"
"topic://Sport//Résultats"	Toutes les rubriques commençant par "Sport" et se terminant par "/Résultats"	"topic://Sport/Football/Résultats" et "topic://Sport/Hockey/National/Div3/Résultats"
"topic://Sport/Football//."	Toutes les rubriques commençant par "Sport/Football/"	"topic://Sport/Football/Résultats" et "topic://Sport/Football/Informations/Signatures/Gestion"

Tableau 82. Exemples d'URI utilisant le schéma de caractères génériques pour le bus d'intégration de services WebSphere Application Server (suite)

Identificateur URI	Correspondance	Exemples
"topic://Sport/*ball//Résultats//."	Rubriques	"topic://Sport/Football/Résultats" et "topic://Sport/Netball/National/Div3/Résultats/2002/Novembre"

Concepts associés

Identificateurs URI de file d'attente
L'identificateur URI d'une file d'attente indique le nom de cette file d'attente ; il peut également en spécifier une ou plusieurs propriétés.

Destinations temporaires
Les applications XMS peuvent créer et utiliser des destinations temporaires.

Identificateurs URI de file d'attente

L'identificateur URI d'une file d'attente indique le nom de cette file d'attente ; il peut également en spécifier une ou plusieurs propriétés.

L'URI d'une file d'attente commence par la séquence `queue://`, suivie du nom de la file d'attente ; elle peut également inclure une liste de paires nom-valeur qui définissent les propriétés de file d'attente restantes.

Pour les files d'attente IBM MQ (mais pas pour les files d'attente de fournisseur de messagerie par défaut WebSphere Application Server), le gestionnaire sur lequel la file d'attente réside peut être spécifié avant la file d'attente, du moment qu'un caractère / (barre oblique) sépare le nom du gestionnaire de files d'attente et le nom de la file d'attente.

Si un gestionnaire de files d'attente est spécifié, il doit s'agir de celui auquel XMS est directement connecté pour la connexion utilisant cette file d'attente, ou il doit être accessible à partir de cette file d'attente. Les gestionnaires de files d'attente éloignées sont pris en charge uniquement pour extraire des messages des files d'attente, et non pour placer des messages dans des files d'attente. Pour plus de détails, reportez-vous à la documentation du gestionnaire de files d'attente IBM MQ.

Si aucun gestionnaire de files d'attente n'est spécifié, le séparateur / (barre oblique) supplémentaire est facultatif ; sa présence ou son absence ne fait aucune différence pour la définition de la file d'attente.

Les définitions de file d'attente suivantes sont toutes équivalentes pour une file d'attente IBM MQ appelée QB sur un gestionnaire de files d'attente appelé QM_A, auquel XMS est directement connecté:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Concepts associés

identificateur URI de rubrique
L'identificateur URI (Uniform Resource Identifier) de rubrique spécifie le nom de la rubrique ; il peut également en spécifier une ou plusieurs propriétés.

Destinations temporaires
Les applications XMS peuvent créer et utiliser des destinations temporaires.

Destinations temporaires

Les applications XMS peuvent créer et utiliser des destinations temporaires.

Généralement, une application utilise une destination temporaire pour recevoir des réponses à des messages de demande. Pour spécifier la destination où la réponse à un message de demande doit être envoyé, une application appelle la méthode Set JMSReplyTo de l'objet Message représentant le message de demande. La destination spécifiée sur l'appel peut être une destination temporaire.

Bien qu'une session soit utilisée pour créer une destination temporaire, la portée d'une destination temporaire est en fait la connexion qui a été utilisée pour créer la session. Toutes les sessions de la connexion peuvent créer des expéditeurs de message et des consommateurs de message pour la destination temporaire. La destination temporaire reste active jusqu'à sa suppression explicite ou la fin de la connexion.

Lorsqu'une application crée une file d'attente temporaire, une file d'attente est créé dans le serveur de messagerie auquel l'application est connectée. Si l'application est connectée à un gestionnaire de files d'attente, une file d'attente dynamique est créée à partir de la file d'attente modèle dont le nom est spécifié par la propriété XMSC_WMQ_TEMPORARY_MODEL et le préfixe utilisé pour former le nom de la file d'attente dynamique est spécifié par la propriété XMSC_WMQ_TEMP_Q_PREFIX . Si l'application est connectée à un bus d'intégration de services, une file d'attente temporaire est créée dans le bus et le préfixe utilisé pour former le nom de la file d'attente temporaire est spécifié par la propriété XMSC_WPM_TEMP_Q_PREFIX .

Lorsqu'une application connectée à un bus d'intégration de services crée une rubrique temporaire, le préfixe utilisé pour former le nom de la rubrique temporaire est spécifié par la propriété XMSC_WPM_TEMP_TOPIC_PREFIX .

Concepts associés

identificateur URI de rubrique
L'identificateur URI (Uniform Resource Identifier) de rubrique spécifie le nom de la rubrique ; il peut également en spécifier une ou plusieurs propriétés.

Identificateurs URI de file d'attente
L'identificateur URI d'une file d'attente indique le nom de cette file d'attente ; il peut également en spécifier une ou plusieurs propriétés.

Expéditeurs de message

Dans XMS, un expéditeur de message peut être créé avec ou sans destination associée. Si l'expéditeur de message est créé avec une destination null, une destination valide doit être spécifiée à l'envoi d'un message.

Expéditeurs de messages avec destination associée

Dans ce scénario, l'expéditeur de message est créé avec une destination valide. Au cours de l'opération d'envoi, la destination ne doit pas être indiquée.

Expéditeurs de messages sans destination associée

Dans XMS .NET, un expéditeur de message peut être créé avec une destination Null.

Pour créer un expéditeur de message sans destination associée lors de l'utilisation de l'API .NET , la valeur NULL doit être transmise en tant que paramètre dans la méthode `CreateProducer()` de l'objet ISession (par exemple, `session.CreateProducer(null)`). Toutefois, une destination valide doit être spécifiée lors de l'envoi du message.

Consommateurs de message

Les consommateurs de message peuvent être classés comme consommateurs durables ou non durables, et comme consommateurs de message synchrone ou asynchrone.

Abonnés durables

Un abonné durable est un consommateur de message qui reçoit tous les messages publiés sur une rubrique, y compris les messages publiés pendant qu'il est inactif.

Avertissement : Ces informations sont pertinentes uniquement si une application se connecte à un gestionnaire de files d'attente IBM MQ ou à un bus d'intégration de services WebSphere Application Server . Elles ne le sont pas pour une connexion en temps réel à un courtier.

Pour créer un abonné durable à une rubrique, une application appelle la méthode Create Durable Subscriber d'un objet Session, en spécifiant sous forme de paramètres un nom identifiant l'abonnement durable et un objet Destination représentant la rubrique. L'application peut créer un abonné durable avec ou sans sélecteur de message ; elle peut également spécifier si l'abonné durable peut recevoir des messages publiés par sa propre connexion.

La session utilisée pour créer un abonné durable doit avoir un identificateur de client associé. L'identificateur de client est le même que celui associé à la connexion utilisée pour créer la session ; il est spécifié selon la méthode décrite dans «Objets ConnectionFactories et Connection», à la page 652.

Le nom qui identifie l'abonnement durable doit être unique dans l'identificateur de client ; cet identificateur fait donc partie de l'identificateur unique de l'abonnement durable. Le serveur de messagerie conserve un enregistrement de l'abonnement durable et garantit que tous les messages publiés sur la rubrique sont conservés jusqu'à l'envoi d'un accusé de réception de la part de l'abonné durable ou jusqu'à leur expiration.

Le serveur de messagerie gère l'enregistrement de l'abonnement durable même après la fermeture de l'abonné durable. Pour réutiliser un abonnement durable précédemment créé, une application doit créer un abonné durable en spécifiant le même nom d'abonnement et en utilisant une session avec le même identificateur de client que ceux utilisés pour cet abonnement durable. Une seule session à la fois peut disposer d'un abonné durable pour un abonnement durable particulier.

La portée d'un abonnement durable est le serveur de messagerie qui gère un enregistrement de l'abonnement. Si deux applications connectées chacun à un serveur de messagerie distinct créent un abonné durable avec le même nom d'abonnement et le même identificateur de client, deux abonnements durables indépendants sont créés.

Pour supprimer un abonnement durable, une application appelle la méthode Unsubscribe d'un objet Session, en spécifiant sous forme de paramètre le nom qui identifie cet abonnement durable. L'identificateur de client associé à la session doit être le même que celui qui est associé à l'abonnement durable. Le serveur de messagerie supprime l'enregistrement de l'abonnement durable qu'il gère et n'envoie plus de messages à l'abonné durable.

Pour modifier un abonnement existant, une application peut créer un abonné durable avec le même nom d'abonnement et le même identificateur de client, mais en spécifiant une rubrique et/ou un sélecteur de message différents. Modifier un abonnement durable équivaut à supprimer cet abonnement et à en créer un nouveau.

Pour une application qui se connecte à un gestionnaire de files d'attente IBM MQ , XMS gère les files d'attente d'abonné. L'application ne doit donc pas nécessairement spécifier une file d'attente d'abonné. XMS ignore la file d'attente d'abonné lorsque celle-ci est spécifiée.

Notez que vous ne pouvez pas modifier la file d'attente d'abonnement pour un abonnement durable. La seule manière de procéder consiste à supprimer l'abonnement et à en créer un nouveau.

Pour une application qui se connecte à un bus d'intégration de services, chaque abonné durable doit avoir un accueil d'abonnement durable. Pour spécifier cet accueil pour tous les abonnés durables utilisant la même connexion, définissez la propriété XMSC_WPM_DUR_SUB_HOME de l'objet ConnectionFactory utilisé pour créer la connexion. Pour spécifier cet accueil pour une rubrique particulière, définissez la propriété XMSC_WPM_DUR_SUB_HOME de l'objet Destination représentant la rubrique. Un accueil d'abonnement durable doit être spécifié pour une connexion avant qu'une application puisse créer un abonné durable qui utilise la connexion. Toute valeur spécifiée pour une destination remplace la valeur spécifiée pour la connexion.

Consommateurs de messages synchrones

Le consommateur de message synchrone reçoit les messages d'une file d'attente de manière synchrone et reçoit un message à la fois. Lorsque la méthode `Receive(wait interval)` est utilisée, l'appel attend le message pendant une période de temps spécifiée (en millisecondes) ou jusqu'à la fermeture du consommateur de message.

Lorsque la méthode ReceiveNoWait() est utilisée, le consommateur de messages synchrones reçoit les messages sans délai ; si le message suivant est disponible, il est reçu immédiatement, sinon, un pointeur vers un objet Message Null est renvoyé.

Consommateurs de message asynchrone

Le consommateur de message asynchrone reçoit un message en provenance d'une file d'attente de manière asynchrone. Le programme d'écoute de message enregistré par l'application est appelé chaque fois qu'un nouveau message est disponible dans la file d'attente.

Messages incohérents dans XMS

Un message incohérent est un message qui ne peut pas être traité par une application MDB. Si un message incohérent est rencontré, l'objet XMS MessageConsumer peut le remettre en file d'attente en fonction de deux propriétés de file d'attente, **BOQUEUE**et **BOTHRESH**.

Dans certaines circonstances, un message distribué à une application MDB peut être remis dans une file d'attente IBM MQ. Cela peut se produire, par exemple, si un message est distribué au sein d'une unité de travail qui est ensuite annulée. Un message annulé est généralement redistribué, mais un message formaté de manière incorrecte peut provoquer des échecs répétés de l'application MDB et donc ne pas être distribué. Un message de ce type est appelé un message incohérent. Vous pouvez configurer IBM MQ de sorte que le message incohérent soit automatiquement transféré à une autre file d'attente pour investigation ou supprimé. Pour plus d'informations sur la configuration de IBM MQ de cette manière, voir «Gestion des messages incohérents dans ASF», à la page 665.

Parfois, un message formaté de manière incorrecte est placé dans une file d'attente. Dans ce contexte, cela signifie que l'application de réception ne peut pas traiter correctement le message. Un tel message peut provoquer l'échec de l'application de réception et l'annulation du message formaté de manière incorrecte. Le message peut ensuite être distribué plusieurs fois sur la file d'entrée et plusieurs fois refusé par l'application. Ces messages sont appelés des messages incohérents. L'objet XMS MessageConsumer détecte les messages incohérents et les réachemine vers une autre destination.

Le gestionnaire de files d'attente IBM MQ garde un enregistrement du nombre de fois où chaque message a été annulé. Lorsque ce nombre atteint une valeur de seuil configurable, le consommateur de message replace le message dans une file d'attente d'annulation nommée. Si cette opération échoue, le message est supprimé de la file d'attente d'entrée, puis placé dans une file d'attente de messages non livrés ou détruit.

Les objets XMS ConnectionConsumer gèrent les messages incohérents de la même manière et avec les mêmes propriétés de file d'attente. Si plusieurs consommateurs de connexion surveillent une même file d'attente, il est possible que le message incohérent soit distribué à une application un nombre de fois supérieur à la valeur de seuil avant la remise en file d'attente. Ce comportement s'explique par la manière dont les clients de connexion individuelle gèrent les files d'attente et remettre en file d'attente les messages incohérents.

La valeur de seuil et le nom de la file d'attente d'annulation sont des attributs d'une file d'attente IBM MQ . Les noms des attributs sont **BackoutThreshold** et **BackoutRequeueQName**. La file d'attente à laquelle ils s'appliquent est la suivante :

- Pour la messagerie point-à-point, il s'agit de la file d'attente locale sous-jacente. Ceci est important lorsque les consommateurs de message et les clients de connexion utilisent des alias de file d'attente.
- Pour la messagerie publication/abonnement en mode normal de fournisseur de messagerie IBM MQ, il s'agit de la file d'attente modèle à partir de laquelle la file d'attente gérée de Topic est créée.
- Pour la messagerie publication/abonnement en mode de migration de fournisseur de messagerie IBM MQ, il s'agit de la file d'attente CCSUB définie sur l'objet TopicConnectionFactory, ou de la file d'attente CCDSUB définie sur l'objet Topic.

Pour définir les attributs **BackoutThreshold** et **BackoutRequeueQName** , exécutez la commande MQSC suivante:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Pour la messagerie de publication / abonnement, si votre système crée une file d'attente dynamique pour chaque abonnement, ces valeurs d'attribut sont obtenues à partir de la file d'attente modèle IBM MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Pour modifier ces paramètres, utilisez :

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value)
BOQUEUE(your.backout.queue.name)
```

Si la valeur de seuil d'annulation est zéro, la gestion des messages incohérents est désactivée et ces messages restent en file d'attente d'entrée. Dans le cas contraire, lorsque le nombre d'annulations atteint la valeur de seuil, le message est envoyé à la file d'attente d'annulation nommée.

Si le nombre d'annulations atteint la valeur de seuil, mais que le message ne peut pas être placé dans la file d'attente d'annulation, il est envoyé à la file d'attente de messages non livrés, ou supprimé s'il s'agit d'un message non persistant.

Cette situation se produit si la file d'attente d'annulation n'est pas définie ou si l'objet MessageConsumer ne peut pas envoyer le message à la file d'attente d'annulation.

Configuration de votre système pour le traitement des messages incohérents

La file d'attente utilisée par XMS .NET lorsqu'il interroge les attributs **BOTHRESH** et **BOQNAME** dépend du style de messagerie :

- Pour la messagerie point-à-point, il s'agit de la file d'attente locale sous-jacente. Elle est importante lorsqu'une application XMS .NET consomme des messages provenant de files d'attente alias ou de files d'attente de cluster.
- Pour la messagerie de publication/abonnement, une file d'attente gérée est créée afin de contenir les messages pour une application. XMS .NET interroge la file d'attente gérée afin de déterminer les valeurs pour les attributs **BOTHRESH** et **BOQNAME**.

  La file d'attente gérée est créée à partir d'une file d'attente modèle associée à l'objet Topic auquel l'application s'est abonnée et hérite des valeurs des attributs **BOTHRESH** et **BOQNAME** spécifiés dans la file d'attente modèle. La file d'attente modèle qui est utilisée varie selon que l'application de réception a souscrit un abonnement durable ou non durable :

  – La file d'attente modèle utilisée pour les abonnements durables est spécifiée par l'attribut **MDURMDL** de l'objet Topic. La valeur par défaut de cet attribut est SYSTEM.DURABLE.MODEL.QUEUE.

  – Pour les abonnements non durables, la file d'attente modèle utilisée est spécifiée par l'attribut **MNDURMDL**. La valeur par défaut de l'attribut **MNDURMDL** est SYSTEM.NDURABLE.MODEL.QUEUE.

Lors de l'interrogation des attributs **BOTHRESH** et **BOQNAME**, XMS .NET :

- Ouvre la file d'attente locale, ou la file d'attente cible pour une file d'attente d'alias.
- Interroge les attributs **BOTHRESH** et **BOQNAME**.
- Ferme la file d'attente locale, ou la file d'attente cible pour une file d'attente alias.

Les options d'ouverture utilisées lors de l'ouverture d'une file d'attente locale, ou d'une file d'attente cible pour une file d'attente alias, dépendent de la version d'IBM MQ qui est utilisée :

- Pour IBM MQ 9.1.0 Fix Pack 4 Long Term Support et les versions antérieures et IBM MQ 9.1.4 Continuous Delivery et les versions antérieures: si la file d'attente locale ou la file d'attente cible d'une file d'attente alias est une file d'attente de cluster, XMS .NET ouvre la file d'attente avec les options MQOO_INPUT_AS_Q_DEF, MQOO_INQUIRE et MQOO_FAIL_IF_QUIESCING . Cela signifie que l'utilisateur qui exécute l'application de réception doit disposer de l'accès en interrogation et en obtention dans l'instance locale de la file d'attente de cluster.

XMS .NET ouvre tous les autres types de file d'attente locale avec les options d'ouverture `MQOO_INQUIRE` et `MQOO_FAIL_IF_QUIESCING`. Pour que XMS .NET interroge les valeurs des attributs, l'utilisateur qui exécute l'application de réception doit disposer de l'accès en interrogation dans la file d'attente locale.

Pour déplacer des messages incohérents dans une file d'attente de remise en file d'attente d'annulation ou dans la file d'attente de rebut du gestionnaire de files d'attente, vous devez accorder à l'utilisateur exécutant l'application les droits `put` et `passall` .

Gestion des messages incohérents dans ASF

Lorsque vous utilisez ASF (Application Server Facilities), les messages incohérents sont traités par l'objet ConnectionConsumer plutôt que par l'objet MessageConsumer. ConnectionConsumer replace les messages en fonction des propriétés **`BackoutThreshold`** et **`BackoutRequeueQName`** de la file d'attente.

Lorsqu'une application utilise ConnectionConsumers, les circonstances dans lesquelles un message est annulé dépend de la session fournie par le serveur d'application :

- Dans le cas d'une session non transactionnelle avec AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, un message est annulé uniquement après une erreur système ou un arrêt imprévu de l'application.
- Dans le cas d'une session non transactionnelle avec CLIENT_ACKNOWLEDGE, les messages pour lesquels aucun accusé de réception n'a été envoyé peuvent être annulés par le serveur d'application via un appel `Session.recover()`.

  Généralement, l'implémentation client de MessageListener ou le serveur d'application appelle `Message.acknowledge()`. `Message.acknowledge()` accuse réception de tous les messages distribués jusqu'à présent au cours de la session.
- Dans le cas d'une session transactionnelle, les messages pour lesquels aucun accusé de réception n'a été envoyé peuvent être annulés par le serveur d'application via un appel `Session.rollback()`.

Navigateurs de file d'attente

Une application utilise un navigateur de files d'attente pour parcourir les messages d'une file d'attente sans les supprimer.

Pour créer un navigateur de files d'attente, une application appelle la méthode Create Queue Browser de l'objet ISession, en spécifiant en tant que paramètre un objet Destination qui identifie la file d'attente à parcourir. L'application peut créer un navigateur de files d'attente avec ou sans sélecteur de message.

Après avoir créé un navigateur de files d'attente, l'application peut appeler la méthode GetEnumerator de l'objet IQueueBrowser pour obtenir la liste des messages en file d'attente. La méthode renvoie une expression énumérative qui encapsule une liste d'objets Message. L'ordre des objets Message dans la liste est le même que celui dans lequel les messages seront extraits de la file d'attente. L'application peut alors utiliser l'expression énumérative pour parcourir chaque message à son tour.

L'expression énumérative est mise à jour de manière dynamique à mesure que les messages sont insérés dans la file d'attente et supprimés de la file d'attente. Chaque fois que l'application appelle IEnumerator.MoveNext () pour parcourir le message suivant dans la file d'attente, le message reflète le contenu en cours de la file d'attente.

Une application peut appeler la méthode GetEnumerator plusieurs fois pour un navigateur de files d'attente donné. Chaque appel renvoie une nouvelle expression énumérative. L'application peut ainsi utiliser plus d'une expression énumérative pour parcourir les messages dans une file d'attente et gérer plusieurs positions dans la file d'attente.

Une application peut utiliser un navigateur de files d'attente pour rechercher un message à supprimer d'une file d'attente, puis faire appel à un consommateur de message avec sélecteur pour supprimer le message. Le sélecteur peut choisir le message en fonction de la valeur de la zone d'en-tête JMSMessageID. Pour plus d'informations sur les zones d'en-tête de message JMS, voir «Zones d'en-tête dans un message XMS», à la page 684.

Demandeurs

Une application utilise un demandeur pour envoyer un message de demande et ensuite, attendre et recevoir la réponse.

De nombreuses applications de messagerie sont basés sur des algorithmes qui envoient un message de demande et attendent ensuite une réponse. XMS fournit une classe appelée Requestor pour vous aider au développement de ce type d'application.

Pour créer un demandeur, une application appelle le constructeur Create Requestor de la classe Requestor et spécifie en tant que paramètres un objet Session et un objet Destination qui identifie l'emplacement où les messages de demande devront être envoyés. La session ne doit pas être transactionnelle, ni être associée au mode d'accusé de réception XMSC_CLIENT_ACKNOWLEDGE. Le constructeur crée automatiquement une file d'attente ou une rubrique temporaire où les messages de réponse devront être envoyés.

Après avoir créé un demandeur, l'application peut appeler la méthode Request de l'objet Requestor pour envoyer un message de demande, puis attendre et recevoir une réponse de l'application à laquelle le message de demande a été envoyé. L'appel attend la réponse jusqu'à sa réception ou jusqu'à ce que la session prenne fin. Une seule réponse est requise par le demandeur pour chaque message de demande.

Lorsque l'application ferme le demandeur, la file d'attente ou la rubrique temporaire est supprimée. Toutefois, la session associée n'est pas fermée.

Suppression d'objet

Lorsqu'une application supprime un objet XMS qu'elle a créé, XMS libère les ressources internes qui ont été allouées à cet objet.

Lorsqu'une application crée un objet XMS, XMS alloue de la mémoire et d'autres ressources internes à cet objet. XMS réserve ces ressource internes jusqu'à ce que l'application supprime l'objet par l'intermédiaire d'une méthode de fermeture et de suppression ; XMS libère alors les ressources internes. Si une application tente de supprimer un objet qui l'a déjà été, l'appel est ignoré.

Lorsqu'une application supprime un objet Connection ou Session, XMS supprime automatiquement certains objets associés et libère leurs ressources internes. Sont concernés les objets créés par l'objet Connection ou Session et n'ayant aucune fonction indépendante. Ces objets sont présentés dans Tableau 83, à la page 666.

Remarque : Si une application ferme une connexion avec des sessions dépendantes, tous les objets dépendant de ces sessions sont également supprimés. Seuls les objets Connection et Session peuvent avoir des objets dépendants.

Tableau 83. Objets supprimés automatiquement

Objet supprimé	Méthode	Objets dépendants supprimés automatiquement
Connexion	Close Connection	Objets ConnectionMetaData et Session
Session	Close Session	Objets MessageConsumer, MessageProducer, QueueBrowser et Requestor

Types de données pour XMS.NET

XMS.NET prend en charge les types System.Boolean, System.Byte, System.SByte, System.Char, System.String, System.Single, System.Double, System.Decimal, System.Int16, System.Int32, System.Int64, System.UInt16, System.UInt32, System.UInt64 et System.Object. Les types de données pour XMS .NET sont différents des types de données pour XMS C/C + +. Vous pouvez utiliser cette rubrique pour identifier les types de données correspondants.

Le tableau suivant présente et décrit brièvement les types de données XMS .NET et XMS C/C++.

Tableau 84. Types de données pour XMS .NET et XMS C/C++

Type XMS .NET	Types XMS C/C++	Description
System.SByte	xmsSBYTE xmsINT8	Valeur 8 bits signée
System.Byte	xmsBYTE xmsUINT8	Valeur 8 bits non signée
System.Int16	xmsINT16 xmsSHORT	Valeur 16 bits signée
System.UInt16	xmsUINT16 xmsUSHORT	Valeur 16 bits non signée
System.Int32	xmsINT32 xmsINT	Valeur 32 bits signée
System.UInt32	xmsUINT32 xmsUINT	Valeur 32 bits non signée
System.Int64	xmsLONG xmsINT64	Valeur 64 bits signée
System.UInt64	xmsULONG xmsUINT64	Valeur 64 bits non signée
System.Char	xmsCHAR16	Caractère 16 bits non signé (Unicode pour .NET)
System.Single	xmsFLOAT	Valeur flottante 32 bits IEEE
System.Double	xmsDOUBLE	Valeur flottante 64 bits IEEE
System.Boolean	xmsBOOL	Valeur True/False
Non applicable	xmsCHAR	Valeur 8 bits signée ou non signée (selon la plateforme)
System.Decimal	Non applicable	Entier 96 bits signé fois 10^0 à 10^{28}
System.Object	Non applicable	Base de tous les types
System.String	Non applicable	Type String

Types primitifs XMS

XMS fournit des équivalents pour les huit types primitifs Java (byte, short, int, long, float, double, char et boolean). Ils permettent l'échange de messages entre XMS et JMS sans perte ni corruption de données.

Le Tableau 85, à la page 668 répertorie l'équivalent Java du type de données, de la taille et de la valeur minimale et maximale pour chaque type de primitif XMS.

Tableau 85. Types de données XMS et leur équivalent Java				
Type de données XMS	**Type de données Java compatible**	**Taille**	**Valeur minimale**	**Valeur maximale**
System.Boolean	boolean	32 bits	false	Oui
System.SBYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.BYTE	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.CHAR	byte	8 bits	-2^7 (-128)	2^7-1 (127)
System.Int16	short	16 bits	-2^{15} (-32768)	2^{15}-1 (32767)
System.Int32	int	32 bits	-2^{31} (-2147483648)	2^{31}-1 (2147483647)
System.Int64	long	64 bits	-2^{63} (-9223372036854775808)	2^{63}-1 (9223372036854775807)
System.Single	float	32 bits	-3.402823E-38 (précision à 7 chiffres après la virgule)	3.402823E+38 (précision à 7 chiffres après la virgule)
System.Double	double	64 bits	-1.79769313486231E-308 (précision à 15 chiffres après la virgule)	1.79769313486231E+308 (précision à 15 chiffres après la virgule)

Conversion implicite d'une valeur de propriété d'un type de données à un autre

Lorsqu'une application extrait la valeur d'une propriété, celle-ci peut être convertie par XMS dans un autre type de données. De nombreuses règles régissent les conversions prises en charge et la manière dont XMS les réalise.

Une propriété d'un objet possède un nom et une valeur ; la valeur est associée à un type de données, où la valeur d'une propriété est également appelée *type de propriété*.

Une application utilise les méthodes de la classe PropertyContext pour extraire et définir les propriétés des objets. Pour extraire la valeur d'une propriété, une application appelle la méthode appropriée au type de propriété. Par exemple, pour extraire la valeur d'une propriété de type entier, une application appelle la méthode GetIntProperty.

Toutefois, lorsqu'une application extrait la valeur d'une propriété, cette valeur peut être convertie par XMS dans un type de données différent. Par exemple, pour extraire la valeur d'une propriété de type entier, une application peut appeler la méthode GetStringProperty, qui renvoie cette valeur sous forme de chaîne. Les conversions prises en charge par XMS sont présentées dans Tableau 86, à la page 668.

Tableau 86. Conversions prises en charge d'un type de propriété vers d'autres types de données	
Type de propriété	**Types de données cible pris en charge**
System.String	System.Boolean, System.Double, System.Float, System.Int32, System.Int64, System.SByte, System.Int16
System.Boolean	System.String, System.SByte, System.Int32, System.Int64, System.Int16
System.Char	System.String
System.Double	System.String
System.Float	System.String, System.Double
System.Int32	System.String, System.Int64

Tableau 86. Conversions prises en charge d'un type de propriété vers d'autres types de données (suite)	
Type de propriété	**Types de données cible pris en charge**
System.Int64	System.String
System.SByte	System.String, System.Int32, System.Int64, System.Int16
System.SByte array	System.String
System.Int16	System.String, System.Int32, System.Int64

Les règles générales suivantes régissent les conversions prises en charge :

- Les valeurs de propriétés numériques peuvent être converties d'un type de données à un autre, du moment qu'aucune donnée n'est perdue au cours de la conversion. Par exemple, la valeur d'une propriété de type System.Int32 peut être convertie en une valeur de type System.Int64, mais pas en une valeur de type System.Int16.
- Une valeur de propriété de tout type de données peut être convertie en chaîne.
- Une valeur de propriété de type chaîne peut être convertie en tout autre type de données, du moment que la chaîne est correctement formatée pour la conversion. Si une application tente de convertir une valeur de propriété de type chaîne qui n'est pas formatée correctement, XMS risque de retourner des erreurs.
- Si une application tente d'effectuer une conversion qui n'est pas prise en charge, XMS risque de retourner une erreur.

Les règles suivantes s'appliquent lorsqu'une valeur de propriété est convertie d'un type de données à un autre :

- Lors de la conversion d'une valeur de propriété de type booléen en type chaîne, la valeur true est convertie en une chaîne "true", et la valeur false est convertie en une chaîne "false".
- Lors de la conversion d'une valeur de propriété de type booléen en type numérique, y compris System.SByte, la valeur true est convertie en 1, et la valeur false est convertie en 0.
- Lors de la conversion d'une valeur de propriété de type chaîne en type booléen, la chaîne "true" (non sensible à la casse) ou "1" et convertie en true, et la chaîne "false" (non sensible à la casse) ou "0" est convertie en false. Toutes les autres chaînes ne peut pas être converties.
- Lors de la conversion d'une valeur de propriété de type chaîne en une valeur de type System.Int32, System.Int64, System.SByte ou System.Int16, la chaîne doit avoir le format suivant :

  [*blancs*][*signe*]*chiffres*

  Les composants de la chaîne sont définis comme suit :

  blancs
  Caractères blancs facultatifs de début.

  signe
  Caractère facultatif de signe plus (+) ou moins (-).

  chiffres
  Séquence contiguë de caractères numériques (0-9). Au moins un chiffre doit être présent.

  Après la séquence de caractères numériques, la chaîne peut contenir d'autres caractères autres que des chiffres, mais la conversion s'arrête au premier caractère de ce type. La chaîne est supposée représenter un entier décimal.

  XMS peut renvoyer une erreur si la chaîne n'est pas formatée correctement.
- Lors de la conversion d'une valeur de propriété de type chaîne en une valeur de type System.Double ou System.Float, la chaîne doit avoir le format suivant :

  [*blancs*][*signe*][*chiffres*][*point*[*d_chiffres*]][*e_car*[*e_signe*]*e_chiffres*]

  Les composants de la chaîne sont définis comme suit :

blancs
Caractères blancs facultatifs de début.

signe
Caractère facultatif de signe plus (+) ou moins (-).

chiffres
Séquence contiguë de caractères numériques (0-9). Au moins un chiffre doit être présent dans *chiffres* ou *d_chiffres*.

point
(Facultatif) Point décimal (.).

d_chiffres
Séquence contiguë de caractères numériques (0-9). Au moins un chiffre doit être présent dans *chiffres* ou *d_chiffres*.

e_car
Caractère exposant, qui peut être *E* ou *e*.

e_signe
Caractère facultatif de signe plus (+) ou moins (-) pour l'exposant.

e_chiffres
Séquence contiguë de caractères numériques (0-9) pour l'exposant. Au moins un chiffre doit être présent si la chaîne contient un caractère exposant.

Après la séquence de chiffres, la chaîne peut contenir d'autres caractères autres que des chiffres, mais la conversion s'arrête au premier caractère de ce type. La chaîne est supposée représenter un nombre décimal en virgule flottante avec un exposant puissance de 10.

XMS peut renvoyer une erreur si la chaîne n'est pas formatée correctement.

- Lors de la conversion d'une valeur de propriété numérique en une chaîne, y compris une valeur de propriété de type System.SByte, la valeur est convertie en une représentation de chaîne sous forme de nombre décimal et non sous forme de chaîne de caractères ASCII pour cette valeur. Par exemple, l'entier 65 est converti en chaîne "65", et non en chaîne "A".
- Lors de la conversion d'une valeur de propriété de type tableau d'octets en une chaîne, chaque octet est converti dans les 2 caractères hexadécimaux représentant cet octet. Par exemple, le tableau d'octets {0xF1, 0x12, 0x00, 0xFF} est converti en chaîne "F11200FF".

Les conversions d'un type de propriété en d'autres types de données sont prises en charge par les méthodes des classes Property et PropertyContext.

Itérateurs

Un itérateur encapsule une liste d'objets et un curseur qui gère la position en cours dans la liste. Le concept d'un itérateur, tel qu'il est disponible dans IBM MQ Message Service Client (XMS) for C/C++, est implémenté à l'aide de l'interface IEnumerator dans IBM MQ Message Service Client (XMS) for .NET.

Lorsqu'un itérateur est créé, le curseur est positionné avant le premier objet. Une application utilise un itérateur pour extraire chaque objet à son tour.

La classe Iterator de IBM MQ Message Service Client (XMS) for C/C++ est équivalente à la classe Enumerator de Java. IBM MQ Message Service Client (XMS) for .NET est similaire à Java et utilise une interface IEnumerator.

Une application peut utiliser une interface IEnumerator pour effectuer les tâches suivantes :

- Extraire les propriétés d'un message
- Extraire les paires nom-valeur du corps d'un message de mappe
- Consulter les messages dans une file d'attente
- Extraire les nom des propriétés de message définies par JMS prises en charge par une connexion

Traitement des erreurs dans XMS .NET

Les exceptions XMS .NET sont toutes dérivées de System.Exception.

XMS .NET exceptions

Les appels de méthode XMS peuvent émettre des exceptions XMS spécifiques comme MessageFormatException, des exceptions XMSExceptions générales ou des exceptions système comme NullReferenceException.

Ecrivez des applications pour intercepter l'une de ces erreurs, soit dans des blocs catch spécifiques, soit dans des blocs catch `System.Exception` en général, en fonction des exigences de votre application.

Codes d'erreur et d'exception XMS

XMS utilise une plage de codes d'erreur pour signaler les pannes. Ces codes d'erreur ne sont pas listés explicitement dans cette documentation, car ils peuvent varier d'une édition à une autre. Seuls les codes d'exception XMS (au format XMS_X_...) sont documentés car ils restent identiques quelle que soit l'édition de XMS.

Information associée

Classe.NET MQException

Codes d'erreur SSL courants émis par les bibliothèques client XMS .NET

Configuration FFDC pour des applications XMS.NET

Traçage des applications XMS .NET

Utilisation des programmes d'écoute des messages et des exceptions dans .NET

Une application .NET utilise un programme d'écoute de message pour recevoir les messages de manière asynchrone, et un programme d'écoute des exceptions pour être averti de manière asynchrone d'un problème avec une connexion.

Pourquoi et quand exécuter cette tâche

La fonctionnalité des programmes d'écoute des messages et des exceptions est la même pour .NET et pour C + +. Toutefois, il existe de petites différences d'implémentation.

Procédure

- Pour configurer un programme d'écoute de messages afin de recevoir des messages de manière asynchrone, procédez comme suit:
  a) Définissez une méthode qui correspond à la signature de la délégation du programme d'écoute de message.

     Vous pouvez définir une méthode statique ou une méthode d'instance dans n'importe quelle classe accessible. La signature de délégation est la suivante :

     ```
     public delegate void MessageListener(IMessage msg);
     ```

     vous pouvez donc définir la méthode de la manière suivante :

     ```
     void SomeMethodName(IMessage msg);
     ```

  b) Instanciez cette méthode en tant que délégué à l'aide d'un élément similaire à l'exemple suivant:

     ```
     MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
     ```

c) Enregistrez la délégation avec un ou plusieurs consommateurs en la définissant sur la propriété MessageListener du consommateur :

```
consumer.MessageListener = OnMsgMethod;
```

Vous pouvez annuler la délégation en redéfinissant la valeur Null pour la propriété MessageListener :

```
consumer.MessageListener = null;
```

- Pour configurer un programme d'écoute des exceptions, procédez comme suit.

  Le programme d'écoute des exceptions fonctionne de la même manière que le programme d'écoute de message, mais sa définition de délégation est différente et il est affecté à la connexion plutôt qu'au consommateur de message. Il en est de même pour C++.

  a) Définissez la méthode.

  La signature de délégation est la suivante :

```
public delegate void ExceptionListener(Exception ex);
```

  vous pouvez donc définir la méthode de la manière suivante :

```
void SomeMethodName(Exception ex);
```

  b) Instanciez cette méthode en tant que délégué à l'aide d'un élément similaire à l'exemple suivant:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) Enregistrez la délégation avec la connexion en définissant sa propriété ExceptionListener :

```
connection.ExceptionListener = OnExMethod ;
```

  Vous pouvez annuler la délégation en redéfinissant la valeur Null pour la propriété ExceptionListener :

```
null:  connection.ExceptionListener = null;
```

Reconnexion automatique du client IBM MQ via XMS

Configurez votre client XMS pour vous reconnecter automatiquement après une panne de réseau, de gestionnaire de files d'attente ou de serveur lors de l'utilisation du client IBM WebSphere MQ 7.1 ou version ultérieure en tant que fournisseur de messagerie.

Utilisez les propriétés `WMQ_CONNECTION_NAME_LIST` et `WMQ_CLIENT_RECONNECT_OPTIONS` de la classe`MQConnectionFactory` pour configurer la reconnexion automatique pour une connexion client. Cette fonction permet de reconnecter automatiquement un client après une panne de connexion, ou en tant qu'option après l'arrêt du gestionnaire de files d'attente. La conception de certaines applications client rend celles-ci incompatibles avec la reconnexion automatique.

La reconnexion automatique devient active une fois que la connexion a été établie.

Remarque : Les propriétés `Client Reconnect Options`, `Client Reconnect Timeout` et `Connection Namelist` peuvent également être définies via la table de définition de canal du client ou l'activation de la reconnexion du client dans le fichier `mqclient.ini`.

Remarque : Si les propriétés de reconnexion sont définies sur l'objet `ConnectionFactory` et dans la table de définition de canal du client, la règle de priorité est la suivante. Si la valeur par défaut de la propriété de liste de noms de connexion est définie dans l'objet `ConnectionFactory`, la table de

définition de canal du client a priorité. Si la liste de noms de connexion n'est pas définie à sa valeur par défaut, les valeurs de propriété définies dans l'objet `ConnectionFactory` ont la priorité. La valeur par défaut de la liste de noms de connexion est `localhost(1414)`.

Utilisation des objets gérés par XMS .NET

Les rubriques de cette section fournissent des informations sur les objets gérés. Les applications XMS peuvent extraire des définitions d'objet depuis un référentiel central d'objets gérés et les utiliser pour créer des fabriques de connexions et des destinations.

Pourquoi et quand exécuter cette tâche

Cette section fournit des informations qui peuvent vous aider à créer et utiliser des objets gérés ; elle décrit les types de référentiel d'objets gérés pris en charge par XMS. Elle explique également comment une application XMS établit une connexion à un référentiel d'objets gérés afin d'extraire les objets gérés requis.

Cette section contient les rubriques suivantes :

XMS .NET Types de référentiel d'objets gérés pris en charge

Les objets gérés de type système de fichiers et protocole LDAP peuvent être utilisés pour se connecter à IBM MQ et à WebSphere Application Server, tandis que le répertoire d'affectation de classe de service permet de se connecter uniquement à WebSphere Application Server.

Les répertoires objet de système de fichiers prennent la forme d'objets sérialisés Java Naming Directory Interface (JNDI). Les répertoires objet LDAP sont des répertoires qui contiennent des objets JNDI. Les répertoires d'objets du système de fichiers et LDAP peuvent être administrés par IBM MQ Explorer, qui est fourni avec IBM MQ et versions ultérieures. Les répertoires objet de système de fichiers et LDAP permettent d'administrer les connexions client en centralisant les fabriques de connexions et les destinations IBM MQ. L'administrateur réseau peut déployer plusieurs applications se référant au même référentiel central, qui sont automatiquement mises à jour pour refléter les modifications de paramètres de connexion effectuées dans le référentiel central.

Un répertoire d'affectation de classe de service contient les fabriques de connexions et les destinations WebSphere Application Server service integration bus ; il peut être administré via la console d'administration WebSphere Application Server. Pour qu'une application XMS extraie des objets depuis un annuaire de dénomination COS, un service Web de recherche JNDI doit être déployé. Ce service Web n'est pas disponible sur toutes les WebSphere Application Server service integration technologies. Pour plus de détails, reportez-vous à la documentation relative au produit.

Remarque : Redémarrez les connexions d'applications pour que les modifications apportées au répertoire d'objet soient prises en compte.

XMS .NET Mappage de propriété pour les objets gérés

Pour permettre aux applications XMS .NET d'utiliser les définitions d'objet de destination et de fabrique de connexions IBM MQ JMS et WebSphere Application Server , les propriétés qui sont extraites de ces définitions doivent être mappées sur les propriétés XMS correspondantes qui peuvent être définies sur les fabriques de connexions et les destinations XMS .

Pour créer, par exemple, une fabrique de connexions XMS avec des propriétés extraites d'une fabrique de connexions JMS IBM MQ , les propriétés doivent être mappées entre les deux.

Les mappages de propriétés s'effectuent automatiquement.

Le Tableau 87, à la page 674 illustre les mappages entre certaines des propriétés les plus courantes des fabriques de connexions et des destinations. Les propriétés affichées dans ce tableau ne sont qu'un petit ensemble d'exemples, et toutes les propriétés affichées ne sont pas pertinentes pour tous les types de connexion et tous les serveurs.

Tableau 87. Exemples de mappage de noms pour des propriétés de fabrique de connexions et de destination

Nom de la propriété IBM MQ JMS	**Nom de la propriété XMS**	**Nom de la propriété WebSphere Application Server service integration bus**
PERSISTENCE (PER)	XMSC_DELIVERY_MODE	
EXPIRY (EXP)	XMSC_TIME_TO_LIVE	
PRIORITY (PRI)	XMSC_PRIORITY	
	XMSC_WPM_HOST_NAME	serverName
	XMSC_WPM_BUS_NAME	busName
	XMSC_WPM_TOPIC_SPACE	topicName

Remarque : Les propriétés affichées dans le Tableau 88, à la page 674 s'appliquent à JMS ainsi qu'à XMS .NET.

Tableau 88. XMS .NETpropriétés

Propriété	**Type d'objet**				
	CF	**QCF**	**TCF**	**File d'attente**	**Topic**
APPLICATIONNAME	Y	Y	Y	Non disponible	Non disponible
ASYNCEXCEPTION	Y	Y	Y	Non disponible	Non disponible
CCDTURL	Y	Y	Y	Non disponible	Non disponible
Canal	Y	Y	Y	Non disponible	Non disponible
ConnectionNameList	Y	Y	Y	Non disponible	Non disponible
CLIENTRECONNECTOPTIONS	Y	Y	Y	Non disponible	Non disponible
CLIENTRECONNECTTIMEOUT	Y	Y	Y	Non disponible	Non disponible
clientId	Non disponible	Y	Non disponible	Non disponible	Non disponible

Tableau 88. XMS .NETpropriétés (suite)					
Propriété	**Type d'objet**				
	CF	**QCF**	**TCF**	**File d'attente**	**Topic**
COMPHDR «1», à la page 675	Y	Non disponible	Y	Non disponible	Non disponible
COMPMSG «1», à la page 675	Y	Y	Y	Non disponible	Non disponible
CONNOPT «1», à la page 675	Y	Y	Y	Non disponible	Non disponible
CONNTAG «1», à la page 675	Y	Y	Y	Non disponible	Non disponible
DESCRIPTION «1», à la page 675	Non disponible	Y	Non disponible	Y	Y
Expiration «1», à la page 675	Non disponible	Non disponible	Non disponible	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
nom_hôte	Non disponible	Y	Non disponible	Non disponible	Non disponible
LOCALADDRESS	Non disponible	Y	Non disponible	Non disponible	Non disponible
Persistance	Non disponible	Non disponible	Non disponible	Y	Y
Port	Non disponible	Y	Non disponible	Non disponible	Non disponible
Priorité«1», à la page 675	Non disponible	Non disponible	Non disponible	Y	Y
PROVIDERVERSION «1», à la page 675	Non disponible	Y	Non disponible	Non disponible	Non disponible
QMANAGER	Y	Y	Y	Y	Non disponible
File d'attente«1», à la page 675	Non disponible	Non disponible	Non disponible	Y	Non disponible
SHARECONVALLOWED	Y	Y	Y	Non disponible	Non disponible
TOPIC «1», à la page 675	Non disponible	Non disponible	Non disponible	Non disponible	Y
Transport «1», à la page 675	Non disponible	Y	Non disponible	Non disponible	Non disponible

Remarque :

1. Ces propriétés ne possèdent pas de propriétés au niveau de l'application, mais elles peuvent éventuellement être définies à l'aide de propriétés administrées.

OutboundSNI Propriété

Depuis la IBM MQ 9.3.0, vous pouvez définir la propriété XMSC_WMQ_OUTBOUND_SNI, qui définit la propriété **OutboundSNI** dans une application.

La propriété XMSC_WMQ_OUTBOUND_SNI_PROPERTY prend les valeurs suivantes:

- XMSC_WMQ_OUTBOUND_SNI_CHANNEL, qui est mappé à "CHANNEL"
- XMSC_WMQ_OUTBOUND_SNI_HOSTNAME, qui est mappé à "HOSTNAME"
- XMSC_WMQ_OUTBOUND_SNI_ASTERISK, qui correspond à "*"

En outre, vous pouvez définir la propriété **OutboundSNI** à l'aide de la variable d'environnement MQOUTBOUND_SNI, qui prend les valeurs suivantes:

- Canal
- HOSTNAME
- *

Remarque : La valeur par défaut de la propriété est XMSC_WMQ_OUTBOUND_SNI_CHANNEL si aucune valeur spécifique n'est définie.

L'ordre de priorité pour la définition de la propriété **OutboundSNI** dans le noeud géré est le suivant:

1. Propriété de niveau application
2. Variable d'environnement

Pour la propriété **OutboundSNI** dans un noeud non géré, `mqclient.ini` uniquement est pris en charge.

XMS .NET Propriétés requises pour les objets ConnectionFactory gérés

Lorsqu'une application crée une fabrique de connexions, un certain nombre de propriétés doivent être définies pour la création d'une connexion à un serveur de messagerie.

Les propriétés répertoriées dans les tableaux suivants constituent le minimum requis pour la création d'une connexion à un serveur de messagerie. Si vous voulez personnaliser la manière dont une connexion est créée, votre application peut définir des propriétés supplémentaires pour l'objet ConnectionFactory. Pour plus d'informations, voir Propriétés de ConnectionFactory. Une liste complète des propriétés disponibles est incluse.

Connexion à un gestionnaire de files d'attente IBM MQ

Tableau 89. Paramètres de propriété des objets ConnectionFactory administrés pour les connexions à un gestionnaire de files d'attente IBM MQ

Propriété XMS requise	Propriété IBM MQ JMS équivalente requise
XMSC_CONNECTION_TYPE	XMS utilise le nom de classe de la fabrique de connexions et la propriété TRANSPORT (TRAN).
XMSC_WMQ_HOST_NAME	HOSTNAME (HOST)
XMSC_WMQ_PORT	PORT
XMSC_WMQ_QUEUE_MANAGER	Nom du gestionnaire de file d'attente

Connexion en temps réel à un courtier

Tableau 90. Paramètres de propriété pour objets ConnectionFactory gérés pour des connexions en temps réel à un courtier

Propriété XMS requise	Propriété IBM MQ JMS équivalente requise
XMSC_CONNECTION_TYPE	XMS utilise le nom de classe de la fabrique de connexions et la propriété TRANSPORT (TRAN).
XMSC_RTT_HOST_NAME	HOSTNAME (HOST)
XMSC_RTT_PORT	PORT

Connexion à un WebSphere Application Server service integration bus

Tableau 91. Paramètres de propriété pour des objets ConnectionFactory gérés pour des connexions à un WebSphere Application Server service integration bus	
XMS Propriété	**Description**
XMSC_CONNECTION_TYPE	Type de serveur de messagerie auquel une application se connecte.: Cette valeur est déterminée à partir du nom de classe de la fabrique de connexions.
XMSC_WPM_BUS_NAME	Pour une fabrique de connexions, nom du bus d'intégration de services auquel l'application se connecte ; pour une destination, nom du bus d'intégration de services dans lequel la destination existe.

XMS .NET Propriétés requises pour les objets Destination gérés

Une application qui crée une destination doit définir plusieurs propriétés pour un objet Destination géré.

Tableau 92. Paramètres de propriété pour les objets Destination gérés		
Type de connexion	**Propriété**	**Description**
Gestionnaire de files d'attente IBM MQ	QUEUE (QU) TOPIC (TOP)	File d'attente à laquelle vous voulez vous connecter Rubrique utilisée par l'application en tant que destination
Connexion en temps réel à un courtier	TOPIC (TOP)	Rubrique utilisée par l'application en tant que destination
WebSphere Application Server service integration bus	topicName queueName	Si votre application se connecte à une rubrique Si votre application se connecte à une file d'attente

XMS .NET Création d'objets gérés

La définition des objets ConnectionFactory et Destination, dont les applications XMS ont besoin pour établir une connexion à un serveur de messagerie, doit être effectuée à l'aide des outils d'administration appropriés.

Avant de commencer

Pour plus de détails sur les différents types de référentiel d'objets gérés pris en charge par XMS, voir «XMS .NET Types de référentiel d'objets gérés pris en charge», à la page 673.

Pourquoi et quand exécuter cette tâche

Afin de créer les objets gérés pour IBM MQ, utilisez IBM MQ Explorer ou l'outil d'administration JMS d'IBM MQ (JMSAdmin).

Pour créer les objets gérés pour IBM MQ ouIBM Integration Bus, utilisez l'outil IBM MQ JMS administration (JMSAdmin).

Pour créer les objets gérés pour WebSphere Application Server service integration bus, utilisez la console d'administrationWebSphere Application Server.

Dans les outils d'administration, la propriété est appelée **APPLICATIONNAME** ou **APPNAME** en abrégé.

Remarque : Vous ne pouvez pas utiliser JMSAdmin pour définir TRANSPORT (UNMANAGED). Par conséquent, pour obtenir un client XMS non géré à l'aide d'un nom d'application choisi par l'administrateur, vous devez entrer la commande suivante:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

Les étapes suivantes récapitulent la procédure à suivre pour créer des objets gérés.

Procédure

1. Créez une fabrique de connexions et définissez les propriétés nécessaires pour créer une connexion à partir de votre application vers le serveur de votre choix.

   Les propriétés minimales requises par XMS pour créer une connexion sont définies dans «XMS .NET Propriétés requises pour les objets ConnectionFactory gérés», à la page 676.
2. Créez la destination requise sur le serveur de messagerie auquel votre application se connecte :
   - Pour une connexion à un gestionnaire de files d'attente IBM MQ , créez une file d'attente ou une rubrique.
   - Pour une connexion en temps réel à un courtier, créez une rubrique.
   - Pour une connexion à un WebSphere Application Server service integration bus, créez une file d'attente ou une rubrique.

   Les propriétés minimales requises par XMS pour créer une connexion sont définies dans «XMS .NET Propriétés requises pour les objets Destination gérés», à la page 677.

XMS .NET création d'objets InitialContext

Une application doit créer un contexte initial afin de créer une connexion au référentiel d'objets gérés et en extraire les objets requis.

Pourquoi et quand exécuter cette tâche

Un objet InitialContext encapsule une connexion au référentiel. L'API XMS fournit les méthodes pour effectuer les tâches suivantes :

- Créer un objet InitialContext
- Rechercher un objet géré dans le référentiel d'objets gérés.

Procédure

- Pour plus de détails sur la création d'un objet InitialContext , voir InitialContext pour .NET et Propriétés de InitialContext.

XMS .NET Propriétés InitialContext

Les paramètres du constructeur InitialContext incluent l'emplacement du référentiel d'objets gérés, indiqué sous la forme d'un identificateur URI. Pour qu'une application établisse une connexion au référentiel, il peut être nécessaire de fournir des informations complémentaires à celles de l'identificateur URI.

Dans JNDI et dans l'implémentation .NET de XMS, les informations complémentaires sont fournies au constructeur dans une table de hachage d'environnement.

L'emplacement d'un référentiel d'objet géré est défini dans la propriété XMSC_IC_URL. Généralement, cette propriété est transmise sur l'appel Create, mais peut être modifiée de manière à se connecter à un répertoire de désignation avant la recherche. Pour les contextes de système de fichiers et de protocole LDAP, cette propriété définit l'adresse de ce répertoire. Dans le contexte de la dénomination COS, il s'agit de l'adresse du service Web qui utilise ces propriétés pour la connexion au répertoire JNDI.

Les propriétés suivantes sont transmises au service Web, qui les utilise pour établir la connexion au répertoire JNDI :

- XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS

- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

Format d'identificateur URI pour contexte initial XMS

L'emplacement du référentiel d'objets gérés est fourni sous la forme d'un identificateur URI. Son format dépend du type de contexte.

Contexte FileSystem

Pour le contexte FileSystem, l'URL indique l'emplacement du répertoire de base du système de fichiers. La structure de l'URL est définie par la norme RFC 1738, *Uniform Resource Locators (URL)* : l'URL est composé du préfixe `file://`, la syntaxe placée après ce préfixe représentant une définition valide d'un fichier pouvant être ouvert sur le système sur lequel XMS est exécuté.

Cette syntaxe peut être spécifique à la plateforme, et utiliser les séparateurs / ou \. Si vous utilisez \, chaque séparateur doit être échappé avec un caractère \ supplémentaire. Cela empêche l'infrastructure .NET de tenter d'interpréter le séparateur comme un caractère d'échappement pour ce qui suit.

Ces exemples illustrent cette syntaxe :

```
file://myBindings
file:///admin/.bindings
file://\\admin\\.bindings
file://c:/admin/.bindings
file://c:\\admin\\.bindings
file://\\\\madison\\shared\\admin\\.bindings
file:///usr/admin/.bindings
```

Contexte LDAP

Pour le contexte LDAP, la structure de base de l'URL est conforme à la norme RFC 2255, *The LDAP URL Format*, avec le préfixe non sensible à la casse `ldap://`

La syntaxe exacte est illustrée dans l'exemple suivant :

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Cette syntaxe est conforme à la norme RFC, mais sans prise en charge d'attributs, de portée, de filtres ou d'extensions.

Exemples de syntaxe :

```
ldap://madison:389/cn=JMSData,dc=IBM,dc=UK
ldap://madison/cn=JMSData,dc=IBM,dc=UK
LDAP:///cn=JMSData,dc=IBM,dc=UK
```

Contexte WSS

Pour le contexte WSS, l'URL se présente sous la forme d'un noeud final de services Web, avec le préfixe `http://`.

Vous pouvez également utiliser le préfixe `cosnaming://` ou `wsvc://`.

Ces deux préfixes signifient que vous utilisez un contexte WSS avec l'URL en accès via http, ce qui permet au type de contexte initial d'être facilement déduit directement à partir de l'URL.

Exemples de syntaxe :

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Service Web de recherche JNDI pour XMS .NET

Pour que vous puissiez accéder à un annuaire de dénomination COS depuis XMS, un service Web de recherche JNDI doit être déployé sur un serveur de WebSphere Application Server service integration bus. Ce service Web transforme les informations Java du service de dénomination COS dans un format lisible par les applications t XMS.

Le service Web est fourni dans le fichier d'archive d'entreprise SIBXJndiLookupEAR.ear, qui se trouve dans le répertoire d'installation. Pour la version actuelle de IBM MQ Message Service Client (XMS) for .NET, SIBXJndiLookupEAR.ear se trouve dans le répertoire *install_dir*\java\lib . Il peut être installé sur un serveur de WebSphere Application Server service integration bus via la console d'administration ou l'outil de création de script wsaadmin. Reportez-vous à la documentation du produit pour plus d'informations sur le déploiement d'applications de service Web.

Pour définir le service Web dans des applications XMS, il suffit de définir la propriété XMSC_IC_URL de l'objet InitialContext sur l'URL du point d'extrémité de service Web. Par exemple, si le service Web est déployé sur le serveur hôte MyServer, vous pouvez définir l'adresse URL de noeud final de service Web suivante :

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

La définition de la propriété XMSC_IC_URL permet à InitialContext Lookup d'appeler le service Web sur le noeud final défini, qui à son tour recherche l'objet géré requis à partir du service de dénomination COS.

Les applications .NET peuvent utiliser le service Web. Le déploiement côté serveur est identique pour XMS C ou /C++, et pour XMS .NET.XMS .NET appelle le service Web directement via Microsoft .NET Framework.

XMS .NET Extraction d'objets gérés

XMS extrait un objet géré du référentiel à l'aide de l'adresse fournie lors de la création de l'objet InitialContext ou dans ses propriétés.

Les objets pouvant être extraits peuvent avoir les types de nom suivants :

- Un nom simple décrivant l'objet Destination, par exemple, une destination de file d'attente appelée SalesOrders
- Un nom composé qui peut être constitué d'éléments SubContexts, séparés par le caractère /, et doit se terminer par le nom d'objet. Exemple de nom composé : "Warehouse/PickLists/DispatchQueue2", où Warehouse et Picklists sont des éléments SubContexts du répertoire de désignation et DispatchQueue2, le nom d'un objet Destination.

Empêcher les applications d'utiliser une version plus récente de XMS

Par défaut, lorsqu'une version XMS plus récente est installée, les applications utilisant la version précédente basculent automatiquement vers la version plus récente sans avoir à recompile.However, vous pouvez empêcher les applications d'utiliser la version plus récente en définissant un attribut dans le fichier de configuration de l'application.

Pourquoi et quand exécuter cette tâche

La fonction de coexistence de versions multiples permet de s'assurer que l'installation d'une version plus récente de XMS ne se substitue pas à la version précédente deXMS. En effet, plusieurs instances d'assemblagesXMS .NET similaires coexistent dans le GAC (Global Assembly Cache) avec des numéros de

version différents. En interne, le GAC utilise un fichier de règles pour acheminer l'appel de l'application vers la version la plus récente de XMS. Les applications s'exécutent sans avoir besoin d'être recompilées et peuvent utiliser les nouvelles fonctionnalités de la version la plus récente de XMS .NET.

Procédure

- Si une application est requise pour utiliser l'ancienne version de XMS .NET , définissez l'attribut `publisherpolicy` sur no dans le fichier de configuration de l'application.

  Remarque : Un fichier de configuration d'application est un fichier dont le nom est composé du nom du programme exécutable auquel il est associé, suivi du suffixe .config. Par exemple, le fichier de configuration d'application de text.exe se nommerait text.exe.config.

  Toutefois, à tout moment, toutes les applications d'un système utilisent la même version de XMS .NET.

Sécurisation des communications pour les applications XMS

Cette section fournit des informations sur la définition de communications sécurisées afin de permettre aux applications XMS de se connecter via SSL (Secure Sockets Layer) à un moteur de messagerie de WebSphere Application Server service integration bus ou à un gestionnaire de files d'attente IBM MQ.

Pourquoi et quand exécuter cette tâche

Cette section contient les rubriques suivantes :

- «Connexions sécurisées à un gestionnaire de files d'attente IBM MQ», à la page 681
- «CipherSuite et CipherSpec mappages de noms pour les connexions XMS à un gestionnaire de files d'attente IBM MQ», à la page 682
- «Connexions sécurisées à un moteur de messagerie WebSphere Application Server service integration bus», à la page 682
- «Mappages de nom CipherSuite et CipherSpec pour connexions à un WebSphere Application Server service integration bus», à la page 683

Connexions sécurisées à un gestionnaire de files d'attente IBM MQ

Pour permettre à une application XMS .NET de créer des connexions sécurisées à un gestionnaire de files d'attente IBM MQ, les propriétés pertinentes doivent être définies dans l'objet ConnectionFactory.

Le protocole utilisé dans la négociation de chiffrement peut être SSL (Secure Sockets Layer) ou TLS (Transport Layer Security), selon la suite de chiffrement CipherSuite spécifiée dans l'objet ConnectionFactory.

Les propriétés ConnectionFactory pour les connexions utilisant SSL à un gestionnaire de files d'attente IBM MQ , avec une brève description, sont répertoriées dans le tableau ci-dessous:

Tableau 93. Propriétés de ConnectionFactory pour des connexions à un gestionnaire de files d'attente IBM MQ via SSL

Nom de la propriété	Description
XMSC_WMQ_SSL_CERT_STORES	Emplacements des serveurs contenant les listes de révocation de certificat à utiliser sur une connexion SSL à un gestionnaire de files d'attente.
XMSC_WMQ_SSL_CIPHER_SPEC	Nom de la spécification CipherSpec à utiliser sur une connexion sécurisée vers un gestionnaire de files d'attente.
XMSC_WMQ_SSL_CIPHER_SUITE	Nom de la suite de chiffrement à utiliser sur une connexion TLS à un gestionnaire de files d'attente. Le protocole utilisé lors de la négociation de la connexion sécurisée dépend de la suite de chiffrement spécifiée.

Tableau 93. Propriétés de ConnectionFactory pour des connexions à un gestionnaire de files d'attente IBM MQ via SSL (suite)	
Nom de la propriété	**Description**
XMSC_WMQ_SSL_CRYPTO_HW	Détails de configuration pour le matériel de cryptographie connecté au système client.
XMSC_WMQ_SSL_FIPS_REQUIRED	La valeur de cette propriété détermine si une application peut ou non utiliser des suites de chiffrement conformes non FIPS. Si cette propriété est pour valeur true, seuls les algorithmes FIPS sont utilisés pour la connexion client/serveur.
XMSC_WMQ_SSL_KEY_REPOSITORY	Emplacement du fichier de clés dans lequel les clés et les certificats sont stockés.
XMSC_WMQ_SSL_KEY_RESETCOUNT	Nombre total d'octets non chiffrés envoyés et reçus dans une conversation SSL avant renégociation de la clé confidentielle.
XMSC_WMQ_SSL_PEER_NAME	Nom d'homologue à utiliser sur une connexion SSL vers un gestionnaire de files d'attente.

CipherSuite et CipherSpec mappages de noms pour les connexions XMS à un gestionnaire de files d'attente IBM MQ

InitialContext effectue la conversion entre la propriété de fabrique de connexions JMSAdmin SSLCIPHERSUITE et la propriété XMS XMSC_WMQ_SSL_CIPHER_SPEC la plus proche. Une conversion similaire est nécessaire si vous spécifiez une valeur pour XMSC_WMQ_SSL_CIPHER_SUITE et aucune valeur pour XMSC_WMQ_SSL_CIPHER_SPEC.

Le Tableau 94, à la page 682 liste les spécifications CipherSpec disponibles et les algorithmes JSSE CipherSuite équivalents.

Tableau 94. Spécifications CipherSpec disponibles et algorithmes JSSE CipherSuite équivalents	
CipherSpec	**Algorithme JSSE CipherSuite équivalent**
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA

Remarque : Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA a été déprécié. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.

Connexions sécurisées à un moteur de messagerie WebSphere Application Server service integration bus

Pour permettre à une application XMS .NET de créer des connexions sécurisées à un moteur de messagerie de WebSphere Application Server service integration bus, les propriétés pertinentes doivent être définies dans l'objet ConnectionFactory.

XMS fournit la prise en charge de SSL et HTTPS pour les connexions à un WebSphere Application Server service integration bus. SSL et HTTPS assurent des connexions sécurisées pour l'authentification et la confidentialité.

Tout comme la sécurité WebSphere, la sécurité XMS est configurée en conformité avec les normes et les conventions de dénomination de sécurité JSSE, qui incluent l'utilisation de suites de chiffrement CipherSuites pour la spécification des algorithmes utilisés lors de la négociation d'une connexion sécurisée. Le protocole utilisé lors de la négociation de chiffrement peut être SSL ou TLS, selon la suite de chiffrement CipherSuite spécifiée dans l'objet ConnectionFactory.

Le Tableau 95, à la page 683 répertorie les propriétés qui doivent être définies dans l'objet ConnectionFactory.

Tableau 95. Propriétés de ConnectionFactory pour des connexions sécurisées à un moteur de messagerie de WebSphere Application Server service integration bus

Nom de la propriété	Description
XMSC_WPM_SSL_CIPHER_SUITE	Nom de la CipherSuite à utiliser sur une connexion TLS à un moteur de messagerie WebSphere Application Server service integration bus . Le protocole utilisé lors de la négociation de la connexion sécurisée dépend de la suite de chiffrement spécifiée.
XMSC_WPM_SSL_KEYRING_LABEL	Certificat à utiliser lors de l'authentification avec le serveur.

Voici un exemple de propriétés ConnectionFactory pour des connexions sécurisées à un moteur de messagerie de WebSphere Application Server service integration bus :

```
cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS, host_name:port_number:chain_name);
cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY, key_repository_pathname);
cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN, transport_chain);
cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE, cipher_suite);
cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE, stash_file_pathname);
```

Où nom_chaîne doit être BootstrapTunneledSecureMessaging ou BootstrapSecureMessaging, et numéro_port représente le numéro de port sur lequel le serveur d'amorçage est à l'écoute des demandes entrantes.

Voici un exemple de propriétés ConnectionFactory avec des valeurs pour des connexions sécurisées à un moteur de messagerie de WebSphere Application Server service integration bus :

```
  /* CF properties needed for an SSL connection */
    cf.setStringProperty(XMSC_WPM_PROVIDER_ENDPOINTS,"localhost:7286:BootstrapSecureMessaging");
    cf.setStringProperty(XMSC_WPM_TARGET_TRANSPORT_CHAIN,"InboundSecureMessaging");
    cf.setStringProperty(XMSC_WPM_SSL_KEY_REPOSITORY,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.kdb");
    cf.setStringProperty(XMSC_WPM_SSL_KEYRING_STASH_FILE,"C:\\Program Files\\IBM\\gsk7\\bin\
\XMSkey.sth");
    cf.setStringProperty(XMSC_WPM_SSL_CIPHER_SUITE,"SSL_RSA_EXPORT_WITH_RC4_40_MD5");
```

Mappages de nom CipherSuite et CipherSpec pour connexions à un WebSphere Application Server service integration bus

Etant donné que IBM Global Security Kit (GSKit) utilise CipherSpecs au lieu de CipherSuites, les noms CipherSuite de style JSSE spécifiés dans la propriété XMSC_WPM_SSL_CIPHER_SUITE doivent être mappés aux noms CipherSpec de style GSKit.

Le Tableau 96, à la page 683 liste la spécification CipherSpec équivalente pour chaque algorithme CipherSuite reconnu.

Tableau 96. Algorithmes CipherSuite disponibles et spécifications CipherSpec équivalentes

CipherSuite	Spécification CipherSpec équivalente
TLS_RSA_WITH_DES_CBC_SHA	TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA

Tableau 96. Algorithmes CipherSuite disponibles et spécifications CipherSpec équivalentes (suite)	
CipherSuite	**Spécification CipherSpec équivalente**
TLS_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA

Remarque : Deprecated TLS_RSA_WITH_3DES_EDE_CBC_SHA a été déprécié. Toutefois, vous pouvez tout de même l'utiliser pour transférer jusqu'à 32 Go de données avant que la connexion ne soit arrêtée avec l'erreur AMQ9288. Pour éviter cette erreur, n'utilisez pas la norme DES triple ou activez la réinitialisation de clé confidentielle lors de l'utilisation de ce CipherSpec.

Messages XMS

Cette section décrit la structure et le contenu des messages XMS et explique comment les applications traitent les messages XMS.

Cette section contient les rubriques suivantes :

- «Parties d'un message XMS», à la page 684
- «Zones d'en-tête dans un message XMS», à la page 684
- «Propriétés d'un message XMS», à la page 685
- «Corps d'un message XMS», à la page 688
- «Sélecteurs de message», à la page 691
- «Mappage de messages XMS en messages IBM MQ», à la page 692

Parties d'un message XMS

Un message XMS est composé d'un en-tête, d'un ensemble de propriétés et d'un corps.

En-tête
L'en-tête d'un message contient des zones ; tous les messages contiennent le même ensemble de zones d'en-tête. XMS et les applications utilisent les valeurs des zones d'en-tête pour identifier et acheminer les messages. Pour plus d'informations sur les zones d'en-tête, voir «Zones d'en-tête dans un message XMS», à la page 684.

Ensemble de propriétés
Les propriétés d'un message spécifient des informations supplémentaires sur celui-ci. Bien que tous les messages disposent du même ensemble de zones d'en-tête, chaque message peut avoir un ensemble de propriétés différent. Pour plus d'informations, voir «Propriétés d'un message XMS», à la page 685.

Corps
Le corps d'un message contient des données d'application. Pour plus d'informations, voir «Corps d'un message XMS», à la page 688.

Une application peut sélectionner les messages qu'elle souhaite recevoir. En utilisant des sélecteurs de message, qui spécifient les critères de sélection. Les critères peuvent être basés sur les valeurs de certaines zones d'en-tête et sur les valeurs de l'une des propriétés du message. Pour plus d'informations sur les sélecteurs de message, voir «Sélecteurs de message», à la page 691.

Zones d'en-tête dans un message XMS

Pour permettre à une application XMS d'échanger des messages avec une application WebSphere JMS, l'en-tête d'un message XMS contient les zones d'en-tête du message JMS.

Les noms de ces zones d'en-tête commencent par le préfixe JMS. Pour une description des zones d'en-tête de message JMS , voir la *spécificationJava Message Service*.

XMS implémente les zones d'en-tête de message JMS en tant qu'attributs d'un objet Message. Chaque zone d'en-tête a ses propres méthodes de définition et d'extraction de sa valeur. Pour une description de ces méthodes, voir IMessage. Une zone d'en-tête est toujours accessible en lecture et en écriture.

Le Tableau 97, à la page 685 répertorie les zones d'en-tête de message JMS et indique comment la valeur de chaque zone est définie pour un message transmis. Certaines zones sont définies automatiquement par XMS lorsqu'une application envoie un message ou, dans le cas de JMSRedelivered, lorsqu'une application reçoit un message.

Tableau 97. Zones d'en-tête de message JMS.]

Nom de la zone d'en-tête de message JMS	**Mode de définition de la valeur pour un message transmis (au format *méthode* [*classe*])**
JMSCorrelationID	Set JMSCorrelationID [Message]
JMSDeliveryMode	Send [MessageProducer]
JMSDestination	Send [MessageProducer]
JMSExpiration	Send [MessageProducer]
JMSMessageID	Send [MessageProducer]
JMSPriority	Send [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Set JMSReplyTo [Message]
JMSTimestamp	Send [MessageProducer]
JMSType	Set JMSType [Message]

Propriétés d'un message XMS

XMS prend en charge trois sortes de propriétés de message : les propriétés définies par JMS, les propriétés définies par IBM et les propriétés définies par l'application.

Une application XMS peut échanger des messages avec une applicationWebSphere JMS car XMS prend en charge les propriétés prédéfinies d'un objet Message suivantes :

- Les mêmes propriétés définies par JMS que celles prises en charge par WebSphere JMS. Les noms de ces propriétés commencent par le préfixe JMSX.
- Les mêmes propriétés définies par IBM que celles prises en charge par WebSphere JMS. Les noms de ces propriétés commencent par le préfixe JMS_IBM_.

Chaque propriété prédéfinie a deux noms :

- Un nom JMS pour une propriété définie par JMS, ou un nom WebSphere JMS pour une propriété définie par IBM.

  Il s'agit du nom sous lequel la propriété est connue dans JMS ou WebSphere JMS, et également du nom transmis avec un message comportant cette propriété. Une application XMS utilise ce nom pour identifier la propriété dans une expression de sélecteur de message.
- Un nom XMS pour identifier la propriété dans toutes les situations, sauf dans une expression de sélecteur de message. Chaque nom XMS est défini en tant que constante nommée dans la classe `IBM.XMS.XMSC`. La valeur de la constante nommée est le nom JMS ou WebSphere JMS correspondant.

En complément des propriétés prédéfinies, une application XMS peut créer et utiliser son propre ensemble de propriétés de message. Ces propriétés sont appelées *propriétés définies par l'application.*

Les propriétés d'un message créé par une application sont accessibles en lecture et en écriture. Elles le reste après l'envoi du message par l'application. Lorsqu'une application reçoit un message, les propriétés de ce message sont en lecture seule. Si une application appelle la méthode `Clear Properties` de

la classe Message lorsque les propriétés d'un message sont en lecture seule, les propriétés deviennent accessibles en lecture et en écriture. La méthode efface également les propriétés.

Le message reçu, puis réacheminé après suppression de ses propriétés, se comporte de manière cohérente avec le comportement de réacheminement d'un BytesMessage MQ XMS for .NET standard dont les propriétés sont effacées.

Ceci n'est toutefois pas recommandé, car les propriétés suivante seront perdues :

- La valeur de la propriété JMS_IBM_Encoding, ce qui implique que les données de message ne peuvent pas être décodées correctement.
- La valeur de la propriété JMS_IBM_Format, ce qui implique que le chaînage d'en-tête entre l'en-tête de message (MQMD ou le nouveau MQRFH2) et les en-têtes existants sera interrompu.

Pour déterminer les valeurs de toutes les propriétés d'un message, une application peut appeler la méthode Get Properties de la classe Message. La méthode crée un itérateur qui encapsule une liste d'objets Property, où chaque objet représente une propriété du message. L'application peut ensuite utiliser les méthodes de la classe Iterator pour extraire tout à tour chaque objet Property, mais aussi les méthodes de la classe Property pour extraire le nom, le type de données et la valeur de chaque propriété.

Propriétés de message définies par JMS

Plusieurs propriétés de message définies par JMS sont prises en charge à la fois par XMS et parWebSphere JMS.

Tableau 98, à la page 686 liste les propriétés de message définies par JMS prises en charge à la fois par XMS et par WebSphere JMS. Pour une description des propriétés définies par JMS, voir la *spécification Java Message Service*. Les propriétés définies par JMS ne sont pas valides pour une connexion en temps réel à un courtier.

Le tableau spécifie le type de données de chaque propriété et indique la manière dont la valeur de la propriété est définie pour un message transmis. Certaines propriétés sont définies automatiquement par XMS lors de l'envoi d'un message par une application ou, dans le cas de JMSXDeliveryCount, lors de la réception d'un message par une application.

Tableau 98. Propriétés de message définies par JMS

Nom XMS de la propriété définie par JMS	**Nom JMS**	**Type de données**	**Mode de définition de la valeur pour un message transmis (au format *méthode* [*classe*])**
JMSX_APPID	JMSXAppID	System.String	Send [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
JMSX_GROUPID	JMSXGroupID	System.String	Set String Property [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Set Integer Property [PropertyContext]
JMSX_USERID	JMSXUserID	System.String	Send [MessageProducer]

Propriétés de message définies par IBM

Plusieurs propriétés de message définies par IBM sont prises en charge par XMS et WebSphere JMS.

Tableau 99, à la page 687 répertorie les propriétés de message définies par IBM qui sont prises en charge par XMS et par WebSphere JMS. Pour plus d'informations sur les propriétés définies par IBM, voir la documentation du produit IBM MQ ou WebSphere Application Server.

Le tableau spécifie le type de données de chaque propriété et indique la manière dont la valeur de la propriété est définie pour un message transmis. Certaines propriétés sont définies automatiquement par XMS lors de l'envoi d'un message par une application.

Tableau 99. Propriétés de message définies par IBM

Nom XMS de la propriété définie IBM	**Nom WebSphere JMS**	**Type de données**	**Mode de définition de la valeur pour un message transmis (au format *méthode* [*classe*])**
JMS_IBM_CHARACTER_SET	JMS_IBM_Character_Set	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_ENCODING	JMS_IBM_Encoding	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_EXCEPTIONMESSAGE	JMS_IBM_ExceptionMessage	System.String	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONREASON	JMS_IBM_ExceptionReason	System.Int32	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONTIMESTAMP	JMS_IBM_ExceptionTimestamp	System.Int64	Receive [MessageConsumer]
JMS_IBM_EXCEPTIONPROBLEM DESTINATION	JMS_IBM_ExceptionProblemDestination	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	JMS_IBM_Feedback	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_FORMAT	JMS_IBM_Format	System.String	Set String Property [PropertyContext]
JMS_IBM_LAST_MSG_IN_GROUP	JMS_IBM_Last_Msg_In_Group	System.Boolean	Set Integer Property [PropertyContext]
JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_PUTAPPLTYPE	JMS_IBM_PutApplType	System.Int32	Send [MessageProducer]
JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Send [MessageProducer]
JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Send [MessageProducer]
JMS_IBM_REPORT_COA	JMS_IBM_Report_COA	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	JMS_IBM_Report_Exception	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	JMS_IBM_Report_Expiration	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_NAN	JMS_IBM_Report_NAN	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PAN	JMS_IBM_Report_PAN	System.Int32	Set Integer Property [PropertyContext]

Tableau 99. Propriétés de message définies par IBM (suite)			
Nom XMS de la propriété définie IBM	**Nom WebSphere JMS**	**Type de données**	**Mode de définition de la valeur pour un message transmis (au format *méthode* [*classe*])**
JMS_IBM_REPORT_PASS_CORREL_ division d"identification	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Set Integer Property [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Send [MessageProducer]

Propriétés définies par l'application d'un message

Une application XMS peut créer et utiliser son propre ensemble de propriétés de message. Lorsqu'une application envoie un message, ces propriétés sont également transmises. Une application de réception peut, à l'aide de sélecteurs de message, choisir les messages qu'elle veut recevoir en fonction de la valeur de ces propriétés.

Pour permettre à une application WebSphere JMS de sélectionner et de traiter les messages envoyés par une application XMS , le nom d'une propriété définie par l'application doit être conforme aux règles de formation des identificateurs dans les expressions de sélecteur de message. Pour plus d'informations, voir «Sélecteurs de message dans JMS», à la page 149. Le type de données d'une valeur de propriété définie par l'application doit être l'un des suivants : System.Boolean, System.SByte, System.Int16, System.Int32, System.Int64, System.Float, System.Double ou System.String.

Corps d'un message XMS

Le corps d'un message contient des données d'application. Un message peut toutefois ne pas comporter de corps, mais uniquement des zones d'en-tête et des propriétés.

XMS prend en charge cinq types de corps de message :

Octets
Le corps contient un flux d'octets. Un message avec ce type de corps est appelé *message d'octets*. L'interface IBytesMessage contient les méthodes permettant de traiter le corps d'un message d'octets.

Mappe
Le corps contient un ensemble de paires nom-valeur, où chaque valeur est associée à un type de données. Un message avec ce type de corps est appelé *message de mappe*. L'interface IMapMessage contient les méthodes permettant de traiter le corps d'un message de mappe.

Objet
Le corps contient un objet Java ou .NET sérialisé. Un message avec ce type de corps est appelé *message d'objet*. L'interface IObjectMessage contient les méthodes permettant de traiter le corps d'un message d'objet.

Flux
Le corps contient un flux de valeurs, où chaque valeur est associée à un type de données. Un message avec ce type de corps est appelé *message de flux*. L'interface IStreamMessage contient les méthodes permettant de traiter le corps d'un message de flux.

Texte
Le corps contient une chaîne. Un message avec ce type de corps est appelé *message texte*. L'interface ITextMessage contient les méthodes permettant de traiter le corps d'un message texte.

L'interface IMessage est le parent de tous les objets message et peut être utilisée dans les fonctions de messagerie pour représenter tous les types de message XMS.

Pour plus d'informations sur la taille et les valeurs maximales et minimales de chacun de ces types de données, voir Tableau 85, à la page 668.

Messages d'octets

Le corps d'un message d'octets contient un flux d'octets. Le corps contient uniquement les données réelles ; il est de la responsabilité de l'application émettrice et de l'application réceptrice d'interpréter ces données.

Les messages d'octets sont utiles si l'application XMS a besoin d'échanger des messages avec des applications qui n'utilisent pas l'interface de programme d'application XMS ou JMS.

Lorsqu'une application a créé un message d'octets, le corps de ce message est accessible en écriture seule. L'application assemble les données d'application dans le corps en appelant les méthodes d'écriture appropriées de l'interface IBytesMessage pour .NET. Chaque fois que l'application écrit une valeur dans le flux de message d'octets, celle-ci est assemblée immédiatement après la valeur précédemment écrite. XMS gère un curseur interne de manière à mémoriser la position du dernier octet assemblé.

Lorsque l'application envoie un message, le corps de ce message passe en lecture seule. Dans ce mode, l'application peut envoyer le message de manière répétée.

Lorsqu'une application reçoit un message d'octets, le corps de ce message est en lecture seule. L'application peut utiliser les méthodes de lecture appropriées de l'interface IBytesMessage pour lire le contenu du flux de message d'octets. L'application lit les octets en séquence, et XMS gère un curseur interne pour mémoriser la position du dernier octet lu.

Si une application appelle la méthode Reset de l'interface IBytesMessage lorsque le corps d'un message d'octets est accessible en écriture, celui-ci passe en lecture seule. La méthode repositionne également le curseur au début du flux de message d'octets.

Si une application appelle la méthode `Clear Body` de l'interface IMessage pour .NET lorsque le corps d'un message d'octets est en lecture seule, celui-ci devient accessible en écriture. La méthode efface également le corps.

Messages de mappe

Le corps d'un message de mappe contient un ensemble de paires nom-valeur, où chaque valeur est associée à un type de données.

Dans chaque paire nom-valeur, le nom est une chaîne qui identifie la valeur, et la valeur est un élément de données d'application qui a l'un des types de données XMS listés dans Tableau 100, à la page 691. L'ordre des paires nom-valeur n'est pas défini. La classe MapMessage contient les méthodes qui permettent de définir et d'extraire les paires nom-valeur.

Une application peut accéder de manière aléatoire à une paire nom-valeur en spécifiant son nom.

Une application .NET peut utiliser la propriété MapNames pour extraire une énumération des noms du corps d'un message de mappe.

Lorsqu'une application extrait la valeur d'une paire nom-valeur, celle-ci peut être convertie par XMS dans un autre type de données. Par exemple, pour obtenir un entier à partir du corps d'un message de mappe, une application peut appeler la méthode GetString de la classe MapMessage, qui retourne l'entier sous forme de chaîne. Les conversions prises en charge sont identiques à celles admises lorsque XMS convertit une valeur de propriété d'un type de données à un autre. Pour plus d'informations sur les conversions prises en charge, voir «Conversion implicite d'une valeur de propriété d'un type de données à un autre», à la page 668.

Lorsqu'un message de mappe a été créé par une application, le corps du message est accessible en lecture et en écriture. Il le reste après l'envoi du message par l'application. Lorsqu'une application reçoit un message de mappe, le corps de ce message est en lecture seule. Si une application appelle la méthode Clear Body de la classe Message alors que le corps du message texte est en lecture seule, celui-ci devient accessible en lecture et en écriture. La méthode efface également le corps.

Messages d'objet

Le corps d'un message d'objet contient un objet sérialisé Java ou .NET.

Une application XMS peut recevoir un message d'objet, modifier ses zones d'en-tête et ses propriétés, puis l'envoyer à une autre destination. Une application peut également copier le corps d'un message d'objet et l'utiliser pour former un autre message d'objet. XMS traite le corps d'un message d'objet en tant que tableau d'octets.

Lorsqu'un message d'objet a été créé par une application, le corps du message est accessible en lecture et en écriture. Il le reste après l'envoi du message par l'application. Lorsqu'une application reçoit un message d'objet, le corps du message est accessible en lecture seulement. Si une application appelle la méthode `Clear Body` de l'interface IMessage pour .NET lorsque le corps d'un message d'objet est en lecture seule, le corps devient accessible en lecture et en écriture. La méthode efface également le corps.

Messages de flux

Le corps d'un message de flux contient un flux de valeurs, chacune d'entre elles ayant un type de données associé.

Les types de données XMS qui peuvent être associés à une valeur sont listés dans Tableau 100, à la page 691.

Lorsqu'une application a créé un message de flux, le corps de ce message est accessible en écriture. L'application assemble les données d'application dans le corps en appelant les méthodes d'écriture appropriées de l'interface IStreamMessage pour .NET. Chaque fois que l'application écrit une valeur dans le flux de message, cette valeur et son type de données sont immédiatement assemblés à la suite de la valeur écrite précédemment par l'application. XMS gère un curseur interne de manière à mémoriser la position de la dernière valeur assemblée.

Lorsque l'application envoie un message, le corps de ce message passe en lecture seule. Dans ce mode, l'application peut envoyer le message plusieurs fois.

Lorsqu'une application reçoit un message de flux, le corps de ce message est en lecture seule. L'application peut utiliser les méthodes de lecture appropriées de l'interface IStreamMessage pour .NET afin de lire le contenu du flux de messages. L'application lit les valeurs en séquence, et XMS gère un curseur interne pour mémoriser la position de la dernière valeur lue.

Lorsqu'une application lit une valeur à partir du flux de message, celle-ci est convertie par XMS dans un autre type de données. Par exemple, pour lire un entier à partir du flux de messages, une application peut faire appel à la méthode ReadString, qui renvoie l'entier sous forme de chaîne. Les conversions prises en charge sont identiques à celles admises lorsque XMS convertit une valeur de propriété d'un type de données à un autre. Pour plus d'informations sur les conversions prises en charge, voir «Conversion implicite d'une valeur de propriété d'un type de données à un autre», à la page 668.

Si une erreur se produit alors qu'une application est en train de lire une valeur du flux de messages, la position du curseur n'est pas modifiée. L'application peut tenter de lire la valeur sous la forme d'un autre type de données.

Si une application appelle la méthode Reset de l'interface IStreamMessage pour XMS lorsque le corps d'un message de flux est en écriture seule, celui-ci passe en lecture seule. La méthode repositionne également le curseur au début du flux de messages.

Si une application appelle la méthode `Clear Body` de l'interface IMessage pour XMS lorsque le corps d'un message de flux est en lecture seule, celui-ci passe en écriture seule. La méthode efface également le corps.

Messages texte

Le corps d'un message texte contient une chaîne.

Lorsqu'un message texte a été créé par une application, le corps du message est accessible en lecture et en écriture. Il le reste après l'envoi du message par l'application. Lorsqu'une application reçoit un

message texte, le corps du message est accessible en lecture seulement. Si une application appelle la méthode Clear Body de l'interface IMessage pour .NET alors que le corps du message texte est en lecture seule, celui-ci devient accessible en lecture et en écriture. La méthode efface également le corps.

Types de données d'éléments de données d'application

Pour qu'une application XMS puisse échanger des messages avec une application IBM MQ classes for JMS , les deux applications doivent pouvoir interpréter les données d'application dans le corps d'un message de la même manière.

Pour cette raison, le type de données de chaque élément de données d'application écrit dans le corps d'un message par une application XMS doit être l'un des types de données répertoriés dans le Tableau 100, à la page 691. Pour chaque type de données, le tableau présente le type de données Java compatible. XMS fournit les méthodes qui permettent d'écrire des éléments de données d'application seulement avec ces types de données.

Tableau 100. Types de données XMS compatibles avec les types de données Java

Type de données XMS	**Représentation**	**Type de données Java compatible**
System.Boolean	Valeur booléenne true ou false	boolean
System.Char16	Caractère codé sur deux octets	char
System.SByte	Entier 8 bits signé	byte
System.Int16	Entier 16 bits signé	short
System.Int32	Entier 32 bits signé	int
System.Int64	Entier 64 bits signé	long
System.Float	Nombre en virgule flottante signé	float
System.Double	Nombre en virgule flottante à double précision signé	double
System.String	Chaîne de caractères	String

Pour plus d'informations sur la taille et les valeurs maximales et minimales de chacun de ces types de données, voir «Types primitifs XMS», à la page 667.

Sélecteurs de message

Une application XMS utilise des sélecteurs de message pour sélectionner les messages qu'elle souhaite recevoir.

Lorsqu'une application crée un consommateur de message, elle peut lui associer une expression de sélecteur de message. Cette expression spécifie les critères de sélection.

Lorsqu'une application se connecte à gestionnaire de files d'attente IBM WebSphere MQ 7.0, la sélection de message s'effectue côté gestionnaire de files d'attente. XMS ne procède à aucune sélection et distribue simplement les messages qu'il reçoit du gestionnaire de files d'attente, fournissant ainsi de meilleures performances.

Une application peut créer plusieurs consommateurs de message avec, pour chacun d'entre eux, sa propre expression de sélecteur de message. Si un message entrant répond aux critères de sélection de plusieurs consommateurs de message, XMS distribue ce message à chacun de ces consommateurs.

Une expression de sélecteur de message peut faire référence aux propriétés d'un message suivantes :

- Propriétés définies par JMS
- Propriétés définie par IBM
- Propriétés définies par l'application

Elle peut également faire référence aux zones d'en-tête de message suivantes :

- JMSCorrelationID
- JMSDeliveryMode
- JMSMessageID
- JMSPriority
- JMSTimestamp
- JMSType

Toutefois, une expression de sélecteur de message ne peut pas faire référence aux données du corps d'un message.

Exemple d'expression de sélecteur de message :

```
JMSPriority > 3 AND manufacturer = 'Jaguar' AND model in ('xj6','xj12')
```

XMS livre un message à un consommateur de message avec cette expression uniquement si le message a une priorité supérieure à 3, une propriété définie par l'application, manufacturer, avec la valeur `Jaguar;` et une autre propriété définie par l'application, model, avec la valeur `xj6` ou `xj12.`

Les règles de syntaxe qui régissent la formation d'une expression de sélecteur de message de XMS sont identiques à celles de IBM MQ classes for JMS. Pour des informations sur la construction d'une expression de sélecteur de message, voir la documentation du produit IBM MQ. Notez que, dans une expression de sélecteur de message, les noms des propriétés définies par JMS doivent être des noms JMS, et les noms des propriétés définies par IBM doivent être des noms IBM MQ classes for JMS. Vous ne pouvez pas utiliser les noms XMS dans une expression de sélecteur de message.

Mappage de messages XMS en messages IBM MQ

Les zones d'en-tête et les propriétés JMS d'un message XMS sont mappées vers des zones dans les structures d'en-tête d'un message IBM MQ.

Lorsqu'une application XMS est connectée à un gestionnaire de files d'attente IBM MQ, les messages envoyés au gestionnaire de files d'attente sont mappés vers des messages IBM MQ de la même manière que les messages IBM MQ classes for JMS sont mappés vers des messages IBM MQ dans des circonstances similaires.

Si la propriété XMSC_WMQ_TARGET_CLIENT d'un objet Destination est définie sur XMSC_WMQ_TARGET_DEST_JMS, les zones d'en-tête et les propriétés JMS d'un message envoyées à la destination sont mappées vers les zones dans la structure d'en-tête MQMD et MQRFH2 du message IBM MQ. Cette définition de la propriété XMSC_WMQ_TARGET_CLIENT suppose que l'application qui réceptionne le message puisse gérer un en-tête MQRFH2. L'application de réception peut donc être une autre application XMS, une application IBM MQ classes for JMS ou une application IBM MQ native, conçue pour gérer un en-tête MQRFH2.

Si la propriété XMSC_WMQ_TARGET_CLIENT d'un objet Destination est définie sur XMSC_WMQ_TARGET_DEST_MQ, les zones d'en-tête et les propriétés JMS d'un message envoyé à la destination sont mappées vers des zones dans la structure d'en-tête MQMD du message IBM MQ. Le message ne contient pas d'en-tête MQRFH2 ; les zones d'en-tête JMS et les propriétés qui ne peuvent pas être mappées en zones dans la structure d'en-tête MQMD sont ignorées. L'application qui reçoit le message peut donc être une application IBM MQ native qui n'est pas conçue pour gérer un en-tête MQRFH2.

Les messages IBM MQ reçus en provenance d'un gestionnaire de files d'attente sont mappés en messages XMS de la même manière que des messages IBM MQ sont mappés en messages IBM MQ classes for JMS dans des circonstances similaires.

Si un message IBM MQ entrant contient un en-tête MQRFH2, le message XMS résultant contient un corps dont le type est déterminé par la valeur de la propriété **Msd** contenue dans le dossier `mcd` de l'en-tête MQRFH2. Si la propriété **Msd** ne figure pas dans l'en-tête MQRFH2, ou si le message IBM MQ n'a pas d'en-tête MQRFH2, le message XMS résultant a un corps dont le type est déterminé par la valeur de la zone *`Format`* de l'en-tête MQMD. Si la zone *`Format`* est définie à MQFMT_STRING, le message XMS est un

message texte. Sinon, le message XMS est un message d'octets. Si le message IBM MQ n'a pas d'en-tête MQRFH2, seules les zones d'en-tête et les propriétés JMS pouvant dériver des zones de l'en-tête MQMD sont définies.

Pour plus d'informations sur le mappage des messages IBM MQ classes for JMS vers les messages IBM MQ , voir «Mappage de messages JMS en messages IBM MQ», à la page 152.

Lecture et écriture du descripteur de message à partir d'une application IBM MQ Message Service Client (XMS) for .NET

Vous pouvez accéder à toutes les zones du descripteur de message (MQMD) d'un message IBM MQ, à l'exception de StrucId et de Version ; BackoutCount et accessible en lecture, mais pas en écriture.

Les attributs de message fournis par IBM MQ Message Service Client (XMS) for .NET facilite la définition des zones MQMD par les applications XMS, ainsi que le pilotage des applications IBM WebSphere MQ.

Certaines restrictions s'appliquent lors de l'utilisation de la messagerie de publication/abonnement. Par exemple, les zones MQMD telles que MsgID et CorrelId sont ignorées si elles sont définies.

La fonction est également indisponible lorsque la propriété **PROVIDERVERSION** est définie sur 6.

Accès aux données de message IBM MQ depuis une application IBM MQ Message Service Client (XMS) for .NET

Vous pouvez accéder aux données de message IBM MQ, y compris à l'en-tête MQRFH2 le cas échéant, et à tout autre en-tête IBM MQ existant dans une application IBM MQ Message Service Client (XMS) for .NET en tant que corps d'un objet JMSBytesMessage.

La fonction décrite dans cette rubrique est disponible uniquement lors de la connexion à un gestionnaire de files d'attente IBM WebSphere MQ 7.0 ou version ultérieure et le fournisseur de messagerie IBM MQ est en mode normal.

Les propriétés de l'objet destination déterminent la manière dont l'application XMS accède à l'ensemble d'un message IBM MQ (y compris l'en-tête MQRFH2, le cas échéant) en tant que corps d'un objet JMSBytesMessage.

ALW Développement d'applications client AMQP

La prise en charge par IBM MQ des API AMQP permet à un administrateur IBM MQ de créer un canal AMQP. Lorsqu'il est démarré, ce canal définit un numéro de port qui accepte les connexions des applications client AMQP.

Vous pouvez installer un canal AMQP sur les systèmes AIX, Linux, and Windows ; il n'est pas disponible sous IBM i ou z/OS.

Une application client AMQP 1.0 peut se connecter au gestionnaire de files d'attente avec un canal AMQP.

Développement d'applications à l'aide de la bibliothèque Apache Qpid JMS

Introduction

La bibliothèque Apache Qpid JMS utilise le protocole AMQP 1.0 pour fournir une implémentation de la spécification JMS 2.

Apache Qpid JMS utilise certains aspects du protocole AMQP 1.0 d'une manière différente des API de messagerie MQ Light . IBM MQ 9.2 a ajouté une prise en charge des canaux IBM MQ AMQP, afin que les applications Apache Qpid JMS puissent se connecter à IBM MQ et effectuer des messages de publication / abonnement, y compris l'utilisation d'abonnements partagés.

IBM MQ 9.3 a ajouté une prise en charge supplémentaire des canaux IBM MQ AMQP, afin que les applications Apache Qpid JMS puissent se connecter à IBM MQ et effectuer une messagerie point-à-point. Pour plus d'informations, voir «Prise en charge point à point sur les canaux AMQP», à la page 698.

IBM MQ 9.3.0 a ajouté une prise en charge supplémentaire de l'exploration des files d'attente pour les canaux AMQP IBM MQ , afin que les applications JMS Apache Qpid puissent se connecter à IBM MQ et

parcourir les messages à partir d'une file d'attente. Pour plus d'informations, voir «Prise en charge point à point sur les canaux AMQP», à la page 698.

IBM MQ 9.3.0 a ajouté deux attributs de canal supplémentaires pour les canaux AMQP, TMPMODEL et TMPQPRFX. Ces attributs sont destinés à la file d'attente modèle et au préfixe de file d'attente temporaire à utiliser lors de la création d'une file d'attente temporaire.

Intercommunication avec d'autres applications IBM MQ

Il est possible d'envoyer des messages entre les applications Apache Qpid JMS et d'autres applications IBM MQ . Par exemple, une application Apache Qpid peut publier des messages sur une rubrique et les applications MQ Light peuvent les recevoir en créant un abonnement.

Une application Apache Qpid JMS peut également publier des messages qui sont consommés par les applications IBM MQ traditionnelles, par exemple à l'aide de l'appel API MQSUB pour s'abonner à la même rubrique.

De même, les applications JMS Qpid d' Apache peuvent s'abonner à des rubriques IBM MQ sur lesquelles les applications IBM MQ traditionnelles publient des messages.

Il est également possible pour une application JMS Qpid Apache de partager un abonnement avec une application MQ Light , à condition que les deux clients spécifient le même nom de partage et le même modèle de rubrique.

Pour ce faire, l'application Apache Qpid JMS ne doit pas se connecter avec un ID client. Cela garantit que le nom d'abonnement IBM MQ utilisé par les deux applications est identique.

Avertissement : Il n'est pas possible pour une application Apache Qpid JMS de partager un abonnement avec une application IBM MQ JMS .

RestrictionsApache Qpid JMS

Les fonctions JMS suivantes sont prises en charge:

- Accusé de réception client, accusé de réception automatique et dups en mode d'accusé de réception ok (DUPS_OK_ACCUSE réception)
  - Connexion avec ou sans données d'identification
  - Création d'un destinataire sur une destination de sujet
  - Création d'un consommateur durable sur une destination de rubrique
  - Création d'un destinataire partagé sur une destination de rubrique
  - Création d'un consommateur durable partagé sur une destination de rubrique
  - Modes d'accusé de réception et d'accusé de réception automatique du client
  - Accusé de réception de message et accusé de réception de session
  - Désabonnement d'un abonnement durable
  - Création d'une file d'attente temporaire
  - Création d'un destinataire dans une file d'attente ou une destination de file d'attente temporaire
  - JMS MessageListeners
  - Consommateur JMS à recevoir le corps ; la méthode JMS 2.0 appelée `Consumer.receiveBody()`
  - Les types de message JMS suivants sont pris en charge:
    - BytesMessage
    - MapMessage
    - ObjectMessage
    - StreamMessage
    - TextMessage
  - Exploration des messages à partir d'une file d'attente

Les fonctions JMS suivantes ne sont pas prises en charge par les clients AMQP:

- L'utilisation de sessions et de JMSContexts faisant l'objet de transactions
  - Utilisation des sélecteurs de message
  - Utilisation de l'attribut **nolocal**
  - Utilisation des sessions de transaction
  - Utilisation du délai de livraison
  - Dans IBM MQ 9.3.0, exploration des messages à partir d'une file d'attente.
  - Création de plusieurs abonnements durables ou consommateurs avec le même ID client et la même rubrique
  - Rubriques temporaires JMS
  - Les filtres AMQP ne sont pas pris en charge.

Téléchargement des exemples de clients AMQP

IBM MQ ne fournit pas de clients AMQP, mais vous pouvez télécharger des clients MQ Light ou des clients AMQP open source basés sur des bibliothèques Qpid Apache . Pour plus d'informations, voir IBM MQ Light et Apache Qpid.

Vous pouvez également télécharger d'autres clients AMQP open source basés sur des bibliothèques Qpid Apache . Pour plus d'informations, voir https://qpid.apache.org/index.html.

Avertissement : Le support IBM ne peut pas fournir de support de configuration ou d'incident pour ces packages client et toute question d'utilisation ou tout rapport d'incident de code doit être dirigé vers les projets respectifs.

Déploiement de clients AMQP dans IBM MQ

Lorsqu'une application est prête à être déployée, elle requiert les mêmes capacités de surveillance, de fiabilité et de sécurité que d'autres applications d'entreprise. Elle peut également échanger des données avec d'autres applications d'entreprise.

Une fois que vous avez déployé un client AMQP, vous pouvez échanger des messages avec des applications IBM MQ . Par exemple, si vous utilisez le client AMQP pour envoyer un message de chaîne JavaScript , l'application IBM MQ reçoit un message MQ , où la zone de format du MQMD est définie sur MQSTR.

Gestion du canal AMQP

Le canal AMQP peut être géré de la même façon que les autres canaux MQ. Vous pouvez utiliser des commandes MQSC, des messages de commande PCF ou IBM MQ Explorer pour définir, démarrer, arrêter et gérer les canaux. Dans Création et utilisation de canaux AMQP, des exemples de commande sont fournis pour définir et démarrer la connexion de clients à un gestionnaire de files d'attente.

Lorsqu'un canal AMQP est démarré, vous pouvez le tester en connectant un client AMQP 1.0 . Par exemple, MQ Light, Apache Qpid Proton ou Apache Qpid JMS.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW MQ Light, Apache Qpid JMSet AMQP (Advanced Message Queuing Protocol)

Le client MQ Light , les clients Apache Qpid comme Apache Proton et les API Apache Qpid JMS sont basés sur le protocole de connexion OASIS Standard AMQP 1.0 . AMQP spécifie la façon dont les messages sont envoyés entre les émetteurs et les récepteurs. Une application agit comme un émetteur lorsqu'elle envoie

un message à un courtier de messages, par exemple IBM MQ. IBM MQ agit comme un émetteur lorsqu'il envoie un message à une application AMQP.

Voici quelques-uns des avantages d'AMQP :

- Protocole normalisé ouvert
- Compatibilité avec d'autres clients AMQP 1.0 open source
- Disponibilité de nombreuses implémentations client open source

Bien qu'un client AMQP 1.0 puisse se connecter à un canal AMQP, certaines fonctions AMQP ne sont pas prises en charge, par exemple les transactions ou les sessions multiples.

Pour plus d'informations, voir le site WebAMQP.org et OASIS Standard AMQP 1.0 PDF.

Les API MQ Light et Apache Qpid JMS possèdent les fonctions de messagerie suivantes:

- Distribution du message au plus une fois
- Distribution du message au moins une fois
- Adressage de destination de chaîne de rubrique
- Durabilité du message et de la destination
- Destinations partagées pour permettre à plusieurs abonnés de partager la charge de travail
- Reprise de client pour une résolution facile des clients bloqués
- Lecture anticipée des messages configurable
- Accusé de réception des messages configurable

Pour obtenir la documentation complète de l'API Apache Qpid JMS , voir Qpid JMS.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Prise en charge d'AMQP 1.0

Les canaux AMQP offrent un niveau de prise en charge pour les applications compatibles avec AMQP 1.0.

Les canaux AMQP prennent en charge un sous-ensemble du protocole AMQP 1.0. Vous pouvez connecter des clients compatibles AMQP 1.0 à un canal IBM MQ AMQP. Pour utiliser toutes les fonctions de messagerie prises en charge par les canaux AMQP, vous devez définir correctement la valeur de certaines zones AMQP 1.0.

Ces informations décrivent la manière selon laquelle les zones AMQP doivent être formatées et répertorient les fonctions de la spécification AMQP 1.0 qui ne sont pas prises en charge par les canaux AMQP.

Les fonctions suivantes de la spécification AMQP 1.0 ne sont pas prises en charge ou sont limitées dans leur utilisation :

Cadre ATTACH

Les canaux AMQP s'attendent à ce que les fonctions du cadre ATTACH contiennent l'un des éléments suivants:

```
topic
temporary queue
queue
shared
```

Les capacités impliquent le type d'objet, et dans le cas de capacités multiples, l'ordre de priorité de la sélection de la capacité est sujet, file d'attente temporaire, file d'attente.

Si une fonction ne contient pas de valeur attendue, la fonction par défaut est la rubrique. Toutes les autres fonctions sont ignorées.

Remarque : Certains clients AMQP ne définissent pas ces fonctions et obtiendront le comportement par défaut IBM MQ de la publication / abonnement. Par exemple, le connecteur Quarkus Reactive Messaging AMQP 1.0 ne définit que les fonctions à partir de la version 2.8.0CR1 .

Les canaux AMQP s'attendent à ce que le `distribution-Mode` sur le cadre ATTACH contienne l'un des éléments suivants, pour une source ou une cible:

- `Déplacer`
- `copy`

où `move` implique une extraction destructive, et `copy` implique un navigateur.

Remarque : Si `distribution-Mode` n'est pas défini ou défini sur une valeur autre que `copy`, `move` est supposé.

Nom des liens

Les canaux AMQP s'attendent à ce que le nom d'un lien AMQP respecte l'un des formats suivants:

- Une rubrique en clair (pour la publication et l'abonnement)
  - La publication des messages : une chaîne de rubrique en clair (nom d'un lien "`/sports/football`", par exemple) entraîne la publication d'un message dans la rubrique `/sports/football`.
  - L'abonnement à une rubrique pour la réception de messages : une chaîne de rubrique en clair (nom d'un lien "`/sports/football`", par exemple) entraîne la définition d'un abonnement dans la rubrique `/sports/football`.
- Une rubrique prolixe privée (pour l'abonnement)
  - Chaîne de rubrique prolixe qui décrit un abonnement privé au format suivant: "`private:`*`topic string`*" (par exemple: "`private:/sports/football`"). Le comportement est identique à une chaîne de rubrique simple. La déclaration `private` différencie un abonnement spécifique d'un client AMQP donné d'un abonnement partagé par plusieurs clients.
- Une rubrique prolixe partagée (pour l'abonnement)
  - Chaîne de rubrique prolixe qui décrit un abonnement partagé au format suivant: "`share:`*`share name`*`:`*`topic string`*" (par exemple: "`share:bbc:/sports/football`").
- Une file d'attente (pour la messagerie point à point pour le fournisseur et le consommateur)
  - Expéditeur pour l'envoi de messages ; une chaîne de nom de file d'attente entraîne l'envoi d'un message par un expéditeur dans une file d'attente.
  - Destinataire devant recevoir des messages ; une chaîne de nom de file d'attente entraîne un destinataire à recevoir des messages d'une file d'attente.
- Vide (pour la messagerie point à point sur une file d'attente temporaire)
  - L'expéditeur envoie des messages dans une file d'attente temporaire ; la valeur Blank permet à un expéditeur d'envoyer un message dans une file d'attente temporaire.
  - Destinataire devant recevoir des messages dans une file d'attente temporaire. Une valeur Blank permet à un destinataire de recevoir des messages d'une file d'attente temporaire.

Pour plus d'informations sur la façon dont les messages AMQP sont mappés vers et depuis les messages IBM MQ , voir «Mappage de zones AMQP à des zones IBM MQ (messages entrants)», à la page 702.

Longueurs maximales des chaînes de rubrique, des noms de partage et des ID client

La taille de la chaîne de rubrique, du nom de partage et de l'ID client ne doit pas être supérieure à 10237 octets. De plus, un ID client ne peut pas comporter plus de 256 caractères.

Ces longueurs maximales signifient que vous pouvez choisir entre :

- une chaîne de rubrique très longue, à condition que le nom de partage soit court,
- un nom de partage long, associé à une chaîne de rubrique courte.

ID conteneur

Les canaux AMQP s'attendent à ce que l'ID conteneur d'un agent AMQP Open contienne un ID client AMQP unique. La longueur maximale d'un ID client AMQP est de 256 caractères et l'ID peut contenir des caractères alphanumériques, le signe de pourcentage (%), la barre oblique (/), le point (.) et le trait de soulignement (_).

Sessions

Les canaux AMQP ne prennent en charge qu'une seule session AMQP. Un client AMQP qui tente de créer plusieurs sessions AMQP reçoit un message d'erreur et est déconnecté du canal.

Transactions

Les canaux AMQP ne prennent pas en charge les transactions AMQP. Une trame de liaison AMQP qui tente de coordonner une nouvelle transaction ou une trame de transfert AMQP qui tente de déclarer une nouvelle transaction est refusée avec un message d'erreur.

Etat de distribution

Les canaux AMQP prennent uniquement en charge un état de distribution pour les trames d'élimination Accepté, Publié ou Modifié. Notez que, lorsqu'un état Modifié est utilisé, les canaux AMQP ne prennent pas en charge l'option non distribuable-ici.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Prise en charge point à point sur les canaux AMQP

Le canal IBM MQ AMQP fournit une prise en charge pour l'envoi de messages aux files d'attente et la réception de messages à partir de ces dernières.

Les clients AMQP tels qu'une bibliothèque Apache Qpid™ JMS demandent une fonction `queue` ou `temporary-queue` lors de l'envoi du cadre de connexion AMQP. Les fonctions permettent au canal AMQP d'identifier l'objet en tant que file d'attente, file d'attente temporaire ou rubrique. En l'absence d'une fonction de file d'attente ou de file d'attente temporaire, ou même de l'une de ces fonctions, la demande est supposée être destinée à une rubrique.

Les canaux AMQP IBM MQ prennent en charge les types de file d'attente pour les éléments suivants:

File d'attente de réception et d'envoi
: Les messages peuvent être envoyés à une file d'attente et consommés à partir d'une file d'attente. Pour la consommation de messages, les modes synchrone et asynchrone sont pris en charge.

Message d'exploration de file d'attente
: Outre l'insertion de messages dans une file d'attente et l'extraction de messages à partir d'une file d'attente, les messages peuvent également être consultés à partir d'une file d'attente.

Support de file d'attente temporaire
: Les messages peuvent être envoyés à une file d'attente temporaire et consommés à partir d'une file d'attente temporaire. Notez que la suppression de file d'attente temporaire est prise en charge si le même objet de file d'attente temporaire que celui utilisé pour créer la file d'attente temporaire est également utilisé pour supprimer la file d'attente temporaire.

  SYSTEM.DEFAULT.MODEL.QUEUE est utilisé lors de la création d'une file d'attente temporaire et le préfixe de la file d'attente temporaire est `AMQP.*`.

SYSTEM.DEFAULT.MODEL.QUEUE est par défaut une file d'attente dynamique temporaire, mais vous pouvez utiliser la propriété **Definition type** sur SYSTEM.DEFAULT.MODEL.QUEUE pour modifier la file d'attente en file d'attente dynamique permanente.

file d'attente dynamique permanente
Une file d'attente dynamique permanente est supprimée lorsqu'un client AMQP, tel qu'une bibliothèque Apache Qpid JMS , envoie une demande avec un cadre `detach` avec l'attribut **closed** défini sur *true*.

Important :

Comportement de Qpid JMS :
Vous devez appeler une commande d'API Qpid JMS , par exemple, la méthode `javax.jms.TemporaryQueue.delete()` pour détruire la file d'attente après utilisation et ce processus efface également les messages présents dans la file d'attente.

Si vous n'émettez pas de commande de ce type, la file d'attente reste avec tous les messages encore présents, lorsque la connexion est fermée.

-

file d'attente dynamique temporaire
Une file d'attente dynamique temporaire est supprimée lorsque le client AMQP ferme la connexion.

Important :

Comportement de Qpid JMS :
Si vous appelez une commande d'API JMS Qpid, par exemple, la méthode `javax.jms.TemporaryQueue.delete()` , la fermeture de la connexion JMS ou les interruptions de connexion, la file d'attente est supprimée et les messages sont perdus.

La fermeture d'une session JMS en elle-même n'entraîne pas la suppression de la file d'attente temporaire, même si la file d'attente temporaire aurait pu être créée à l'aide de la méthode `javax.jms.Session.createTemporaryQueue()` .

-

Tâches associées
Création et utilisation de canaux AMQP
Sécurisation des clients AMQP

ALW Mappage des zones de message AMQP et IBM MQ

Les messages AMQP sont composés d'un en-tête, d'annotations de distribution, d'annotations de message, de propriétés, de propriétés d'application, de corps et de pied de page.

Les messages AMQP sont composés des parties suivantes:

En-tête
L'en-tête facultatif contient cinq attributs fixes du message:

- **durable** -spécifie les exigences de durabilité
- **priority** -priorité relative des messages
- **ttl** -durée de vie en millisecondes
- **first-acquirer** -si c'est le cas, le message n'a pas été acquis par un autre lien
- **delivery-count** -nombre de tentatives de distribution précédentes ayant échoué.

Annotations de distribution
Facultatif. Indique les attributs d'en-tête non standard du message pour les différentes audiences prévues. Les annotations de distribution transmettent des informations de l'homologue d'envoi à l'homologue de réception.

Annotations de message
Facultatif. Indique les attributs d'en-tête non standard du message pour les différentes audiences prévues. La section des annotations de message est utilisée pour les propriétés du message qui sont destinées à l'infrastructure et qui doivent être propagées à chaque étape de distribution.

Propriétés
Facultatif. Cette partie est équivalente au descripteur de message MQ . Il contient les zones fixes suivantes:

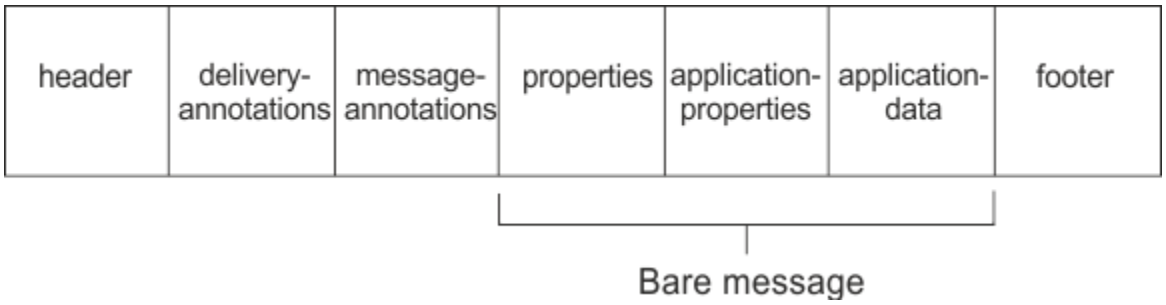
- **message-id** -identificateur de message d'application
- **user-id** -ID de création de l'utilisateur
- **to** -adresse du noeud auquel le message est destiné
- **subject** -objet du message
- **reply-to** -noeud auquel les réponses sont envoyées
- **correlation-id** -identificateur de corrélation d'application
- **content-type** -Type de contenu MIME
- **content-encoding** -Type de contenu MIME. Utilisé comme modificateur du type de contenu.
- **absolute-expired-time** -heure à laquelle ce message est considéré comme arrivé à expiration
- **creation-time** -heure à laquelle ce message a été créé
- **group-id** -groupe auquel appartient ce message
- **group-sequence** -numéro de séquence de ce message dans son groupe
- **reply-to-group-id** -groupe auquel appartient le message de réponse

Applications-propriétés
Equivalent aux propriétés de message MQ .

Corps
Equivalent à la charge utilisateur MQ .

Pied de page
Facultatif. Le pied de page est utilisé pour obtenir des détails sur le message ou la distribution qui ne peuvent être calculés ou évalués qu'après la construction ou l'affichage de l'ensemble du message (par exemple, hachages de message, HMAC, signatures et détails de chiffrement).

Le format de message AMQP est illustré dans la figure suivante:



Les propriétés, les propriétés d'application et la partie de données d'application sont appelées "message nu". Il s'agit du message tel qu'il est envoyé par l'expéditeur et il est non modifiable. Le destinataire voit l'intégralité du message, y compris l'en-tête, le pied de page, les annotations de distribution et les annotations de message.

Pour une description complète du format de message AMQP 1.0 , voir la norme OASIS à l'adresse https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

Tâches associées
Création et utilisation de canaux AMQP
Sécurisation des clients AMQP

ALW Mappage de zones IBM MQ à des zones AMQP (messages sortants)

Lorsqu'un message IBM MQ est publié et que IBM MQ l'envoie à un consommateur AMQP, il propage certains des attributs du message IBM MQ dans des attributs de message AMQP équivalents.

en-tête

Un en-tête n'est inclus que si l'une des cinq zones de l'en-tête contient une valeur autre que la valeur par défaut. Seules les zones dont la valeur n'est pas la valeur par défaut sont incluses dans l'en-tête. Les cinq zones d'en-tête sont initialement dérivées de la propriété `mq_amqp.Hdr` équivalente, si elle est définie, puis modifiées comme indiqué dans le tableau suivant:

Tableau 101. Mappages de zones d'en-tête

Zone	Valeur par défaut	Valeur
durable	false	True si `MQMD.Persistence` est défini sur `MQPER_PERSISTENT`, false dans le cas contraire.
priority	4	A partir de `mq_amqp.Hdr.Pri`, s'il est défini, ou à partir de `MQMD.Priority`, s'il est défini. Si aucun de ces deux ensembles n'est défini, définissez la valeur sur 4.
durée de vie	Non applicable	`MQMD.Expiry` en millisecondes. Si la valeur de `MQMD.Expiry` est MQEI_UNLIMITED, définissez la valeur maximale de la zone AMQP ttl
premier acquéreur	false	A partir de `mq_amqp.Hdr.Fac`, s'il est défini, ou à la valeur false dans le cas contraire.
comptage-livraison	0	A partir de `mq_amqp.Hdr.Dct`, si défini, ou 0 dans le cas contraire.

annotation de distribution

Défini selon les besoins par le canal AMQP.

annotation de message

Non inclus.

propriétés

Les **propriétés** ne seront pas modifiées à partir des propriétés `mq_amqp.Prp` équivalentes si elles sont définies. Si le message n'était pas à l'origine un message AMQP (c'est-à-dire que le type PutAppln'est pas MQAT_AMQP), une section de propriétés est générée comme décrit dans le tableau suivant:

Tableau 102. Mappages de zones de propriétés

Nom	Valeur
ID-message	`MQMD.MsgId` est défini comme binaire.
ID utilisateur	La forme UTF-8 de `MQMD.UserIdentifier` est définie comme binaire dans l'ordre des octets du réseau.
à	File d'attente à partir de laquelle le message a été reçu ou, pour une publication, chaîne de rubrique.
sujet	Non défini.

Tableau 102. Mappages de zones de propriétés (suite)	
Nom	**Valeur**
répondre à	Le paramètre `MQMD.ReplyToQ` s'il n'est pas vide, sinon il n'est pas défini.
ID-corrélation	`MQMD.CorrelId` est défini comme binaire s'il n'est pas vide, sinon il n'est pas défini.
type de contenu	Non défini.
Content-Encoding	Non défini.
heure-d'expiration-absolue	Non défini.
heure de création	Les zones `MQMD.PutDate` et `MQMD.PutTime` sont utilisées pour générer un horodatage.
id-groupe	Non défini.
séquence de groupes	Non défini.
ID-groupe-réponse	Non défini.

propriétés d'application

Toutes les propriétés IBM MQ du groupe "usr" sont ajoutées en tant que **application-properties**.

corps

Le canal AMQP effectue une opération get with convert pour convertir le contenu IBM MQ en UTF-8.

Si le contenu IBM MQ ne contient pas de message AMQP, le contenu IBM MQ est défini dans le corps comme une section de données de chaîne unique pour le format MQFMT_STRING (la conversion en UTF-8 a abouti), ou comme une section de données binaires unique dans le cas contraire.

Si un message de format AMQP est inclus, il est défini comme corps. Tous les en-têtes IBM MQ (à l'exception des propriétés de message qui sont renvoyées dans un descripteur de message) qui précèdent le message AMQP sont ajoutés en tant que valeur binaire si le corps est une séquence AMQP. Sinon, les en-têtes IBM MQ sont supprimés.

bas de page

Aucun pied de page n'est inclus.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

Référence associée

MQMD-Descripteur de message

ALW Mappage de zones AMQP à des zones IBM MQ (messages entrants)

Lorsque le canal AMQP reçoit un message et le place dans IBM MQ, il propage certains des attributs du message AMQP dans des attributs de message IBM MQ équivalents.

Les restrictions suivantes s'appliquent lors du mappage d'un message AMQP entrant:

- Si la zone `message-id` ou `correlation-id` de la partie propriétés est un identificateur unique universel ou un identificateur unique universel, le message est rejeté.
- Tout `message-annotations` entraîne le rejet du message.

- Les sections `delivery-annotations` et `footer` sont autorisées, mais ne sont pas propagées dans le message IBM MQ .

Les sous-sections suivantes montrent l'expression IBM MQ d'un message AMQP.

Descripteur de message

Tableau 103. Descripteur de message pour le message AMQP

Zone	**Valeur**
StrucId	ID_STRUCD_MQM
Version	MQMD_VERSION_1
Rapport	MQRO_AUCUN
MsgType	MQMT_DATAGRAM
Expiration	Valeur extraite de la zone `ttl` de l'en-tête de message AMQP
Commentaires	MQFB_AUCUN
Codage	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Format	Voir le contenu
Priorité	Valeur extraite de la zone `priority` dans l'en-tête de message AMQP. Si cette option est définie, limitée à un maximum de 9. S'il n'est pas défini, la valeur par défaut est 4.
Persistance	Si la zone `durable` de l'en-tête de message AMQP est définie sur true, définissez sur MQPER_PERSISTENT. Sinon, définissez la valeur sur MQPER_NOT_PERSISTENT.
MagId	Le gestionnaire de files d'attente alloue un MsgIdunique de 24 octets.
Corrélation	Valeur extraite de la zone `correlation-id` dans les propriétés AMQP, si elles sont définies. Défini sur une valeur binaire de 24 octets. Sinon, défini sur MQCI_NONE/.
BackoutCount	0
ReplyToQ	Valeur extraite de la zone `reply-to` dans les propriétés AMQP, si elles sont définies. Sinon, il est défini sur "".
ReplyToQMgr	""
Rapport	Valeur dérivée des propriétés de rapport JMS IBM définies dans les propriétés d'application AMQP.
UserIdentifier	Défini sur l'identificateur de l'utilisateur authentifié qui s'est connecté au canal AMQP
AccountingToken	MQACT_AUCUN
ApplIdentityData	Chaîne hexadécimale. Défini sur les 8 derniers octets de l'identificateur de connexion MQ du canal AMQP.
PutApplType	MQAT_AMQP
PutApplName	
PutDate	Valeur extraite de la zone `creation-time` des propriétés AMQP, si elles sont définies. Sinon, définissez la date en cours.

Tableau 103. Descripteur de message pour le message AMQP (suite)	
Zone	**Valeur**
PutTime	Valeur extraite de la zone `creation-time` des propriétés AMQP, si elles sont définies. Sinon, il est défini sur l'heure en cours.
ApplOriginData	""

Propriétés des messages

Il existe deux raisons pour lesquelles vous pouvez définir des propriétés de message:

- Permet aux parties du message AMQP de circuler dans le gestionnaire de files d'attente sans affecter le contenu du message.
- Pour autoriser la sélection du `application-properties`.

Le tableau suivant présente les propriétés définies à partir du message AMQP:

Tableau 104. Propriétés des messages AMQP			
Nom de la propriété	**Nom MQRFH2**	**Tapez**	**Description**
AMQPListener (programme d'écoute MQ)	mq_amqp.Lis	MQTYPE_CHAINE	Chaîne d'identification du canal AMQP. Il est utilisé pour générer le message, afin que les parties intéressées puissent savoir quelle version a inséré le message (par exemple, l'équipe de maintenance lors du diagnostic des problèmes). La valeur n'est pas validée par le gestionnaire de files d'attente et ne doit pas être documentée en externe.
Version AMQPVersion	mq_amqp.Ver	MQTYPE_CHAINE	Version du message AMQP. S'il n'est pas présent, "1.0" est supposé. La valeur n'est pas validée par le gestionnaire de files d'attente.
AMQPClient	mq_amqp.Cli	MQTYPE_CHAINE	Chaîne d'identification de l'API. Il est utilisé pour envoyer le message AMQP au canal, afin que les parties intéressées puissent savoir quelle version a inséré le message (par exemple, l'équipe de service lors du diagnostic des problèmes). La valeur n'est pas validée par le gestionnaire de files d'attente et ne doit pas être documentée en externe.
AMQPDable	mq_amqp.Hdr.Dur	MQTYPE_BOOLÉEN	Valeur de la zone `durable` dans l'en-tête de message AMQP, si elle est définie.
Priorité AMQP	mq_amqp.Hdr.Pri	MQTYPE_INT32	Valeur de la zone `priority` dans l'en-tête de message AMQP, si elle est définie.

Tableau 104. Propriétés des messages AMQP (suite)

Nom de la propriété	Nom MQRFH2	Tapez	Description
AMQPTtl	mq_amqp.Hdr.Ttl	MQTYPE_INT64	Valeur de la zone `ttl` dans l'en-tête de message AMQP, si elle est définie.
AMQPFirstAcquirer	mq_amqp.Hdr.Fac	MQTYPE_BOOLÉEN	Valeur de la zone `first-acquirer` dans l'en-tête de message AMQP, si elle est définie.
AMQPDeliveryCount	mq_amqp.Hdr.Dct	MQTYPE_INT64	Valeur de la zone `delivery-count` dans l'en-tête de message AMQP, si elle est définie.
AMQPMsgId	mq_amqp.Prp.Mid	MQTYPE_CHAINE	Valeur de la zone `message-id` dans les propriétés AMQP, si elle est définie sous la forme d'une chaîne.
		MQTYPE_BYTE_STRING	Valeur de la zone `message-id` dans les propriétés AMQP, si elle est définie en tant que chaîne d'octets.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	Valeur de la zone `user-id` dans les propriétés AMQP, si définie.
AMQPTo	mq_amqp.Prp.To	MQTYPE_CHAINE	Valeur de la zone `to` dans les propriétés AMQP, si définie.
Sujet AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_CHAINE	Valeur de la zone `subject` dans les propriétés AMQP, si définie.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_CHAINE	Valeur de la zone `reply-to` dans les propriétés AMQP, si définie.
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_CHAINE	Valeur de la zone `correlation-id` dans les propriétés AMQP, si elle est définie sous la forme d'une chaîne.
		MQTYPE_BYTE_STRING	Valeur de la zone `correlation-id` dans les propriétés AMQP, si elle est définie en tant que chaîne d'octets.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_CHAINE	Valeur de la zone `content-type` dans les propriétés AMQP, si définie.
AMQPContentEncoding	mq_amqp.Prp.Cneing	MQTYPE_CHAINE	Valeur de la zone `content-encoding` dans les propriétés AMQP, si définie.
Heure AMQPAbsoluteExpiry	mq_amqp.Prp.Aet	MQTYPE_CHAINE	Valeur de la zone `absolute-expiry-time` dans les propriétés AMQP, si définie.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_CHAINE	Valeur de la zone `creation-time` dans les propriétés AMQP, si définie.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_CHAINE	Valeur de la zone `group-id` dans les propriétés AMQP, si définie.
AMQPGroupSequence	mq_amqp.Prp.Gsq	MQTYPE_INT64	Valeur de la zone `group-sequence` dans les propriétés AMQP, si définie.

Tableau 104. Propriétés des messages AMQP (suite)			
Nom de la propriété	**Nom MQRFH2**	**Tapez**	**Description**
AMQPReplyToGroupId	mq_amqp.Prp.Rtg	MQTYPE_CHAINE	Valeur de la zone `reply-to-group-id` dans les propriétés AMQP, si définie.

Chacune des propriétés d'application du message AMQP est définie en tant que propriété de message IBM MQ . La section `application-properties` doit être reconstituée de la même manière octet par octet et les restrictions suivantes s'appliquent:

- Si une propriété d'application est rejetée par le code de validation MQSETMP, le message à rejeter. Exemple :
  - La longueur du nom de propriété est limitée à MQ_MAX_PROPERTY_NAME_LENGTH.
  - Le nom de la propriété doit respecter les règles définies par la spécification de langage Java pour les identificateurs Java .
  - Le nom de la propriété ne doit pas commencer par `JMS` ou `usr.JMS` , à l'exception des propriétés JMS documentées qui peuvent être définies.
  - Le nom de la propriété ne doit pas être un mot clé SQL.
- Une propriété d'application contenant le caractère Unicode U+002E (".") provoque le rejet du message. La propriété doit être expressible dans le groupe de propriétés "usr" utilisé par JMS.
- Seules les propriétés null, boolean, byte, short, int, long, float, double, binary et string sont prises en charge. Une propriété d'application de tout autre type entraîne le rejet du message.

Vous pouvez définir les propriétés JMS suivantes à l'aide de `application-properties`:

- JMS_IBM_REPORT_EXCEPTION
- JMS_IBM_REPORT_EXPIRATION
- JMS_IBM_REPORT_COA
- JMS_IBM_REPORT_COD
- JMS_IBM_REPORT_PAN
- JMS_IBM_REPORT_NAN
- JMS_IBM_REPORT_PASS_MSG_ID
- JMS_IBM_REPORT_PASS_CORREL_ID
- JMS_IBM_REPORT_DISCARD_MSG

Notez que les noms et les valeurs de propriété sont cohérents avec les détails «Mappage des zones spécifiques au fournisseur JMS», à la page 164 équivalents et que les valeurs non valides sont ignorées.

contenu

- Pour un body AMQP avec une section de données binaires unique, les données binaires (à l'exclusion des bits AMQP) sont placées en tant que charge IBM MQ , avec un format MQFMT_NONE.
- Pour un body AMQP avec une seule section de données de chaîne, les données de chaîne (à l'exclusion des bits AMQP) sont placées comme charge utile IBM MQ , avec un format de MQFMT_STRING.
- Sinon, le body AMQP forme le contenu en l'état, avec un format MQFMT_AMQP.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Fiabilité de la distribution des messages

Cette section compare les fonctions de fiabilité de l'API MQ Light et d' Apache Qpid JMS.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Fiabilité des messages MQ Light

Il existe quatre fonctions de l'API MQ Light qui vous permettent de contrôler la fiabilité de la distribution des messages vers et depuis les applications AMQP.

Il s'agit des fonctions suivantes :

- «Qualité de service des messages», à la page 707
- «Confirmation automatique d'un abonné», à la page 708
- «Durée de vie de l'abonnement», à la page 708
- «Persistance des messages», à la page 708

Qualité de service des messages

L'MQ Light API propose deux qualités de service :

- Au plus une fois
- Au moins une fois

Vous pouvez choisir la qualité de service que les diffuseurs de publications et les abonnés doivent utiliser.

Si vous utilisez un client MQ Light , définissez l'option client ou subscribe **qos** sur *QOS_AT_MOST_ONCE* ou *QOS_AT_LEAST_ONCE*.

Si vous utilisez un client AMQP différent, associez l'attribut **`settled`** de la trame de transfert (pour les diffuseurs de publications) ou de la trame de disposition (pour les abonnés) à la valeur *true* ou *false*, selon la qualité de service à appliquer.

La qualité de service détermine quand un message est supprimé du côté `sending` d'une conversation:

Publication

- Si un diffuseur de publications choisit **QOS** *0* (au plus une fois), il n'attend pas l'accusé de réception du gestionnaire de files d'attente avant de supprimer sa copie du message. Si la connexion au gestionnaire de files d'attente échoue alors que l'envoi n'a pas abouti, il se peut que les abonnés ne reçoivent pas le message.
- Si un diffuseur de publications choisit **QOS** *1* (au moins une fois), il attend que le gestionnaire de files d'attente accuse réception du message pour confirmer que le message a été écrit dans les files d'attente des abonnés avant de supprimer sa copie du message. Si la connexion au gestionnaire de files d'attente échoue au cours de l'envoi, le diffuseur de publications renvoie le message une fois qu'il est reconnecté au gestionnaire de files d'attente.

Abonnement

- Si un abonné choisit **QOS** *0*, le gestionnaire de files d'attente n'attend pas d'accusé de réception de la part de l'abonné avant de supprimer sa copie du message. Si la connexion à l'abonné échoue avant que l'abonné n'ait reçu le message, il se peut que le message soit perdu.
- Si un abonné choisit **QOS** *1*, le gestionnaire de files d'attente attend l'accusé de réception de la part de l'abonné avant de supprimer sa copie du message. V 9.4.0 A partir de IBM MQ 9.3.3, les messages d'accusé de réception sont supprimés par lots afin d'améliorer les performances. Pour plus d'informations, voir «Suppression des messages AMQP reconnus de la file d'attente par lots», à la page 710.

Si la connexion à l'abonné échoue avant que l'abonné n'ait reçu le message, le message est conservé par le gestionnaire de files d'attente. Le gestionnaire de files d'attente, une fois reconnecté, renvoie le message à l'abonné ou l'envoie à un autre abonné si l'abonnement est partagé.

Confirmation automatique d'un abonné

Si un abonné choisit **QOS** *1* (au moins une fois), il doit accuser réception de chaque message pour que le gestionnaire de files d'attente puisse supprimer sa copie. L'abonné peut décider du moment auquel il accuse réception des messages.

Si l'option **auto-confirm** a pour valeur *true*, le client MQ Light accuse automatiquement réception de la distribution de chaque message, une fois qu'il a reçu le message sur le réseau.

Ainsi, en cas de défaillance du réseau, le message est redistribué à l'application. Toutefois, il est toujours possible que l'application perde le message en cas d'incident survenant entre l'accusé de réception du message par le client MQ Light et le traitement du message par l'application.

Si l'option **auto-confirm** a pour valeur *false*, le client MQ Light n'accuse pas automatiquement réception de la distribution du message, mais laisse l'application décider du moment auquel en accuser réception.

Ainsi, l'application peut effectuer une mise à jour d'une ressource externe, comme une base de données ou un fichier, avant d'envoyer un accusé de réception au gestionnaire de files d'attente pour indiquer que le message a été traité et qu'il peut être supprimé.

Durée de vie de l'abonnement

Lorsqu'une application s'abonne, elle choisit si l'abonnement et la destination dans laquelle les messages sont stockés pour cet abonnement continuent d'exister après la déconnexion de l'application.

L' MQ Light option d'abonnement **ttl** est utilisée pour spécifier la durée (en millisecondes) pendant laquelle un abonnement continue d'exister après la déconnexion de l'application. Si l'application se reconnecte avant cette heure, l'abonnement reprend et l'application peut continuer à consommer les messages de cet abonnement.

Si la période de durée de vie se passe sans que l'application ne se reconnecte, l'abonnement est supprimé et tous les messages stockés sur sa destination sont perdus, même s'il s'agit de messages persistants.

S'il est important de ne pas perdre de messages, vous devez indiquer à l'application une valeur de durée de vie suffisamment élevée pour que les messages ne soient pas perdus en cas d'indisponibilité.

Persistance des messages

La persistance des messages est contrôlée par les applications de publication et d'abonnement, ainsi que la configuration des objets de rubrique IBM MQ.

Si l'abonné AMQP utilise **QOS** *0* (au plus une fois) et crée un abonnement non durable, le canal AMQP place toujours des messages non persistants dans la file d'attente de l'abonné, quelles que soient les autres options décrites ci-après.

Notez que si le gestionnaire de files d'attente est arrêté, l'abonnement ainsi que les messages sont perdus.

Si un diffuseur de publications associe l'en-tête AMQP **durable** à la valeur *true*, le canal AMQP place des messages persistants dans les files d'attente des abonnés.

Si le gestionnaire de files d'attente est arrêté pour une raison quelconque, les abonnés peuvent tout de même accéder aux messages une fois le gestionnaire de files d'attente redémarré.

Si l'en-tête **durable** n'est pas défini, le canal AMQP choisit la persistance des messages publiés en fonction de l'attribut **DEFPSIST** de l'objet de rubrique IBM MQ pertinent.

Par défaut, il s'agit de SYSTEM.BASE.TOPIC, qui utilise un attribut **DEFPSIST** associé à la valeur *NO* (non persistant).

Avertissement : Les versions ultérieures du client MQ Light ne prennent pas en charge la définition de l'en-tête durable AMQP.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Fiabilité des messages Apache Qpid JMS

Il existe quatre fonctions de la bibliothèque Apache Qpid™ JMS qui vous permettent de contrôler la fiabilité de la distribution des messages vers et depuis les applications AMQP.

Il s'agit des éléments suivants:

- «Publication», à la page 709/Producteur pour la messagerie point à point
  - Expiration du message
  - Persistance des messages
- «Abonnement», à la page 709
  - Durabilité des abonnements
  - Mode d'accusé de réception de session (applicable également à la messagerie point-à-point du consommateur)

Publication

Expiration du message

La définition de la valeur de durée de vie du fournisseur JMS affecte le délai d'expiration donné aux messages publiés par ce fournisseur de messages.

Assurez-vous que la valeur de durée de vie d'un fournisseur JMS est suffisamment élevée pour que les messages soient consommés avant leur expiration.

Vous pouvez également laisser la valeur de durée de vie non définie pour empêcher l'expiration du message à partir de la file d'attente d'abonnement.

Persistance des messages

La définition du mode de distribution de l'expéditeur de message JMS définit la persistance du message IBM MQ publié dans la rubrique spécifiée.

Veillez à utiliser **`DeliveryMode`**.PERSISTENT pour les messages qui doivent être conservés lorsqu'un gestionnaire de files d'attente est arrêté ou en cas d'indisponibilité.

Abonnement

Durabilité des abonnements

Les canaux AMQP prennent en charge la création d'abonnements durables à l'aide des versions durables des méthodes de création de consommateur JMS :

- **`createDurableConsumer()`**
- **`createSharedDurableConsumer()`**

Mode d'accusé de réception de session

Pour garantir qu'un message consommé a été entièrement traité avant d'être supprimé de la file d'attente d'abonnement IBM MQ , créez une session JMS à l'aide de **Session**.Utilisez le mode CLIENT_ACCUSE réception et la méthode **message.acknowledge()** pour accuser réception de ce message et de tout autre message précédemment reçu dans cette session.

Concepts associés

Développement d'applications client AMQP

La prise en charge par IBM MQ des API AMQP permet à un administrateur IBM MQ de créer un canal AMQP. Lorsqu'il est démarré, ce canal définit un numéro de port qui accepte les connexions des applications client AMQP.

V 9.4.0 Suppression des messages AMQP reconnus de la file d'attente par lots

Si une application AMQP utilise la distribution de message QOS_AT_LEAST_ONCE (1), le service AMQP attend un accusé de réception de l'application avant de supprimer la copie d'un message qu'il conserve après avoir envoyé ce message à l'application. Depuis la IBM MQ 9.3.3, les messages dont l'accusé de réception a été émis sont supprimés de la file d'attente par lots, et non individuellement, ce qui améliore les performances.

Pourquoi et quand exécuter cette tâche

Avant IBM MQ 9.4.0, chaque message est supprimé individuellement de la file d'attente.

Depuis la IBM MQ 9.4.0, vous pouvez utiliser les deux propriétés système **com.ibm.mq.AMQP.BATCHSZ** et **com.ibm.mq.AMQP.BATCHINT** pour optimiser le traitement des accusés de réception par lots afin d'améliorer les performances:

com.ibm.mq.AMQP.BATCHSZ
Cet attribut définit le nombre maximal d'accusés de réception à recevoir avant que le service AMQP supprime des messages. Ce nombre peut être compris entre 1 et 9999. Si un nombre non valide est défini ou si le nombre spécifié est hors plage, la valeur par défaut 50 est utilisée.

La taille de lot n'affecte pas la manière dont les messages sont transférés. Les messages sont toujours transférés individuellement, mais sont ensuite supprimés dans un lot une fois que le service AMQP a reçu les accusés de réception. La taille réelle d'un lot peut être inférieure à la valeur spécifiée par **com.ibm.mq.AMQP.BATCHINT**. Par exemple, un lot se termine si la période définie par l'attribut **com.ibm.mq.AMQP.BATCHINT** expire.

com.ibm.mq.AMQP.BATCHINT
Cet attribut définit la durée, en millisecondes, pendant laquelle le service AMQP conserve les messages d'accusé de réception dans la file d'attente. Si le lot n'est pas saturé, il est effacé après cette durée. Vous pouvez indiquer n'importe quel nombre de millisecondes, compris entre 1 et 999 999 999. La valeur par défaut est 50. Si vous n'indiquez pas de valeur pour cet attribut, la valeur par défaut 50 est utilisée.

Remarques :

1. Le fait que le service AMQP attende un accusé de réception avant de supprimer un message dépend de l'une des deux qualités de service suivantes utilisées par une application pour la distribution des messages:
   - QOS pour QOS_AT_MOST_ONCE (0)

     Si une application AMQP utilise cette qualité de service, elle n'accuse pas réception des messages, de sorte que le service AMQP supprime les messages après les avoir envoyés à l'application sans attendre un accusé de réception.
   - QOS_AT_LEAST_ONCE (1)

     Si une application AMQP utilise cette qualité de service, elle accuse réception des messages, de sorte que le service AMQP conserve une copie de chaque message après l'avoir envoyé à l'application jusqu'à ce qu'elle reçoive un accusé de réception de l'application. Si l'application se

déconnecte ou perd la connexion avant d'accuser réception du message, le message est mis à la disposition des autres applications. Le service AMQP ne supprime pas un message de la file d'attente tant qu'il n'a pas été accusé de réception.

2. MQ Appliance Les propriétés système **com.ibm.mq.AMQP.BATCHSZ** et **com.ibm.mq.AMQP.BATCHINT** ne sont pas applicables sous IBM MQ Appliance. La valeur par défaut 50 est utilisée sous IBM MQ Appliance.

.

Procédure

Utilisez les propriétés système **com.ibm.mq.AMQP.BATCHSZ** et **com.ibm.mq.AMQP.BATCHINT** pour optimiser le traitement des accusés de réception par lots.

Depuis la IBM MQ 9.3.3, lorsque le gestionnaire de files d'attente est créé, le fichier `amqp_java.properties` contient les valeurs par défaut suivantes pour les propriétés système:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

En fonction du débit de messages consommé, vous pouvez optimiser le traitement des accusés de réception par lots afin d'améliorer les performances. Un gestionnaire de files d'attente migré ne possède pas ces propriétés dans le fichier `amqp_java.properties` . Par conséquent, pour un gestionnaire de files d'attente migré ou si les propriétés ne sont pas définies, les valeurs par défaut sont utilisées. Vous pouvez ajouter ces propriétés pour affiner les valeurs afin d'optimiser les performances.

Les messages avec accusé de réception sont supprimés par lots lorsque l'une des conditions suivantes est remplie:

- Le nombre de messages d'accusé de réception atteint **com.ibm.mq.AMQP.BATCHSZ**.
- **com.ibm.mq.AMQP.BATCHINT** est dépassé après le démarrage du lot.
- L'application se déconnecte ou ferme la file d'attente ou la rubrique avant que les deux conditions précédentes ne soient satisfaites.

ALW Topologies pour les clients AMQP avec IBM MQ

Exemples de topologies pour vous aider à développer vos clients AMQP en vue de leur utilisation avec IBM MQ.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Clients AMQP communiquant via IBM MQ

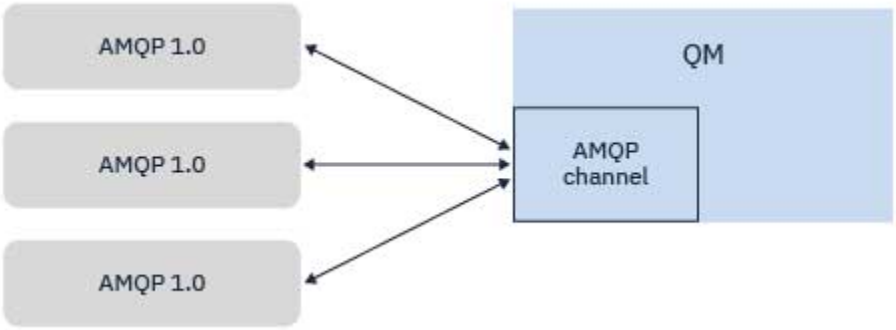
Vous pouvez utiliser IBM MQ comme fournisseur de messagerie pour toute application compatible avec AMQP 1.0. Bien qu'un client AMQP 1.0 puisse se connecter à un canal AMQP, certaines fonctions AMQP ne sont pas prises en charge, par exemple les transactions ou les sessions multiples.

En définissant un ou plusieurs canaux AMQP, les clients AMQP 1.0 peuvent se connecter au gestionnaire de files d'attente et envoyer des messages à une chaîne de rubrique. Ils peuvent également s'abonner à un modèle de rubrique afin de recevoir les messages correspondant à ce modèle.

Dans le scénario suivant, les seules applications qui envoient et reçoivent des messages sont les applications AMQP 1.0 .

Les applications peuvent choisir de rendre persistantes les destinations créées via l'abonnement à une chaîne de rubrique pour que les messages ne soient pas perdus si l'application perd provisoirement la connexion au gestionnaire de files d'attente.

Elles peuvent également choisir la durée pendant laquelle les messages sont conservés avant d'être purgés sur la destination.



Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Clients AMQP échangeant des messages avec des applications IBM MQ

En définissant et en démarrant un canal AMQP, les applications AMQP 1.0 peuvent publier des messages reçus par des applications MQ existantes. Les messages publiés via un canal AMQP sont tous envoyés dans des rubriques MQ et non dans des files d'attente MQ. Une application MQ qui a créé un abonnement avec l'appel API MQSUB reçoit les messages publiés par les applications AMQP 1.0, à condition que la chaîne de rubrique ou l'objet de rubrique utilisé par l'application MQ corresponde à la chaîne de rubrique publiée par le client AMQP.

Les données de message, les attributs et les propriétés AMQP sont définis dans le message MQ reçu par l'application MQ. Pour plus d'informations sur les mappages de messages AMQP vers MQ , voir «Mappage de zones AMQP à des zones IBM MQ (messages entrants)», à la page 702.

Si l'application MQ a créé un abonnement durable, les messages publiés par l'application AMQP sont stockés dans la file d'attente associée à l'abonnement. Les messages sont ensuite reçus par l'application MQ lorsque l'application reprend son abonnement. Si l'application AMQP spécifie une durée de vie pour les messages et que l'application MQ ne se reconnecte pas dans cet intervalle de temps, le message arrive à expiration dans la file d'attente.

Les applications AMQP 1.0 peuvent également consommer des messages publiés par des applications MQ existantes. Les messages publiés par des applications MQ dans une chaîne de rubrique ou une rubrique MQ sont reçus par une application AMQP 1.0, à condition que l'application soit abonnée à un modèle de rubrique qui correspond à la chaîne de rubrique publiée.

Si l'application AMQP 1.0 spécifie une valeur de durée de vie pour l'abonnement et que l'application AMQP se déconnecte pendant une période supérieure à la durée de vie, l'abonnement arrive à expiration dans le gestionnaire de files d'attente et les messages stockés dans la file d'attente d'abonnement sont perdus.

Les zones, les propriétés de message et les données d'application MQMD sont définies dans le message AMQP reçu par l'application AMQP. Pour plus d'informations sur les mappages de messages MQ vers AMQP, voir «Mappage de zones IBM MQ à des zones AMQP (messages sortants)», à la page 701.

Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

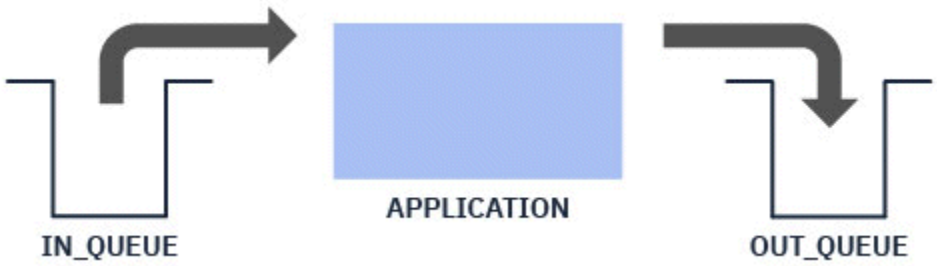
ALW Configuration des clients AMQP pour qu'ils interagissent directement avec les applications dans les files d'attente IBM MQ

L'implémentation IBM MQ AMQP prend en charge la publication / l'abonnement et le point à point. Pour tout client AMQP qui ne prend pas en charge le point à point, procédez comme suit pour envoyer des messages à une file d'attente ou pour recevoir des messages d'une file d'attente.

Présentation

Par exemple, supposons qu'une application extrait des messages d'une file d'attente d'entrée IN_QUEUE et les place dans une file d'attente de sortie OUT_QUEUE. Il est possible pour les clients AMQP d'insérer des messages dans IN_QUEUEet d'obtenir des messages à partir de OUT_QUEUE

Remarque : Aucune modification n'est requise pour l'application elle-même.



Pour qu'un diffuseur de publications AMQP puisse placer un message dans une file d'attente, vous devez créer un abonnement d'administration pour la chaîne de rubrique dans laquelle le client AMQP publie, avec une destination de la file d'attente prévue ; voir «Envoi de messages à l'application:», à la page 713.

Pour qu'un abonné AMQP puisse extraire des messages d'une file d'attente, vous devez remplacer la file d'attente par une file d'attente alias du même nom, avec une cible d'un objet de rubrique représentant la chaîne de rubrique à laquelle le client AMQP est abonné ; voir «Obtention des messages de l'application:», à la page 713

Envoi de messages à l'application:

L'application est déjà en train de prélever des messages dans IN_QUEUE et vous souhaitez qu'un client AMQP puisse publier des messages afin qu'ils soient placés dans cette file d'attente pour être traités par l'application.

Pour ce faire, vous créez un nouvel abonnement d'administration, dans lequel la chaîne de rubrique dont cet abonnement reçoit les messages, correspond à la chaîne de rubrique dans laquelle le client AMQP publie. La file d'attente de destination de cet abonnement est la file d'entrée de l'application, IN_QUEUE.

Tous les messages publiés dans la chaîne de rubrique définie pour cet abonnement d'administration sont acheminés vers la destination définie, en l'occurrence IN_QUEUE.

En supposant que le client AMQP publie dans une chaîne de rubrique /application/in, vous pouvez créer un abonnement d'administration APP_IN, à l'aide de la commande MQSC suivante:

```
DEF SUB(APP_IN) TOPICSTR('/application/in') DEST('IN_QUEUE')
```

Une fois cet objet défini, tous les messages publiés dans /application/in sont acheminés vers la destination IN_QUEUE, où ils sont récupérés par l'application de la même manière que les autres messages placés dans cette file d'attente par d'autres applications.

Obtention des messages de l'application:

L'application place des messages dans OUT_QUEUE, où ils peuvent être récupérés et traités par d'autres clients.

Toutefois, dans ce cas, vous souhaitez que les messages soient distribués à un client AMQP à la place, mais les clients AMQP utilisent uniquement la publication / l'abonnement et ne peuvent pas extraire les messages directement d'une file d'attente.

Pour remplacer les clients ayant précédemment reçu un message par le client AMQP abonné, vous devez créer un objet de rubrique, pour la chaîne de rubrique à laquelle le client AMQP est abonné et une file d'attente alias.

Avertissement : Si vous définissez la file d'attente alias, puis que vous démarrez l'application émettrice avant que des clients AMQP aient eu la possibilité de s'abonner, les messages envoyés par l'application émettrice à la "file d'attente" (désormais une rubrique) seront perdus car il n'y a pas d'abonnés.

Les modifications décrites dans ce texte remplacent les clients qui recevaient précédemment des messages par le client AMQP abonné uniquement. Pour utiliser une combinaison d'AMQP et d'autres clients afin d'obtenir des messages, des modifications plus étendues sont nécessaires.

En supposant que le client AMQP s'abonne à une chaîne de rubrique `/application/out`, vous pouvez définir l'objet de rubrique APP_OUT à l'aide de la commande MQSC suivante:

```
DEF TOPIC(APP_OUT) TOPICSTR('/application/out')
```

Tous les messages distribués à cet objet de rubrique sont distribués au client AMQP s'abonnant à la même chaîne de rubrique.

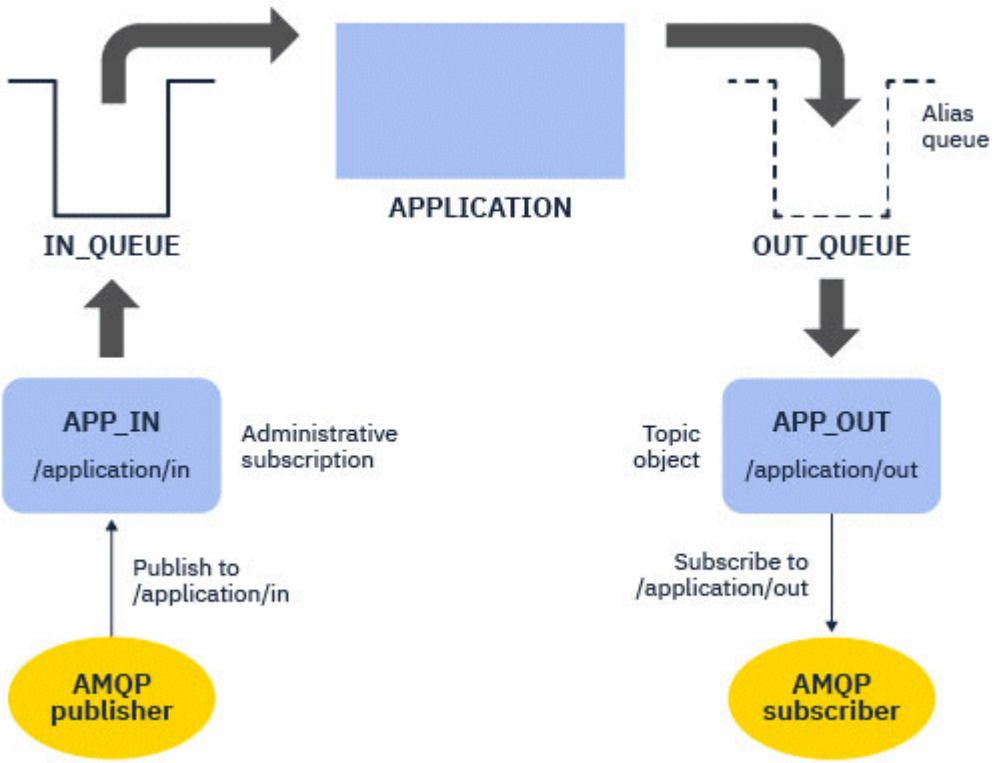
Vous devez ensuite vous assurer que les messages placés dans OUT_QUEUE par l'application sont distribués à ce nouvel objet de rubrique, de sorte qu'ils soient envoyés au client abonné.

Pour ce faire, remplacez la file d'attente existante OUT_QUEUE par une file d'attente alias du même nom, avec un type cible de l'objet de rubrique que vous venez de créer, à l'aide de la commande MQSC suivante:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Désormais, les messages insérés par l'application dans OUT_QUEUE n'attendent pas dans la file d'attente à récupérer ; ils sont distribués à la cible de cette file d'attente alias, c'est-à-dire le nouvel objet de rubrique APP_OUT.

Le client AMQP, qui est abonné à la chaîne de rubrique représentée par cet objet de rubrique `/application/out`, reçoit ensuite tous les messages envoyés à cet objet de rubrique à partir de la file d'attente alias.



Tâches associées

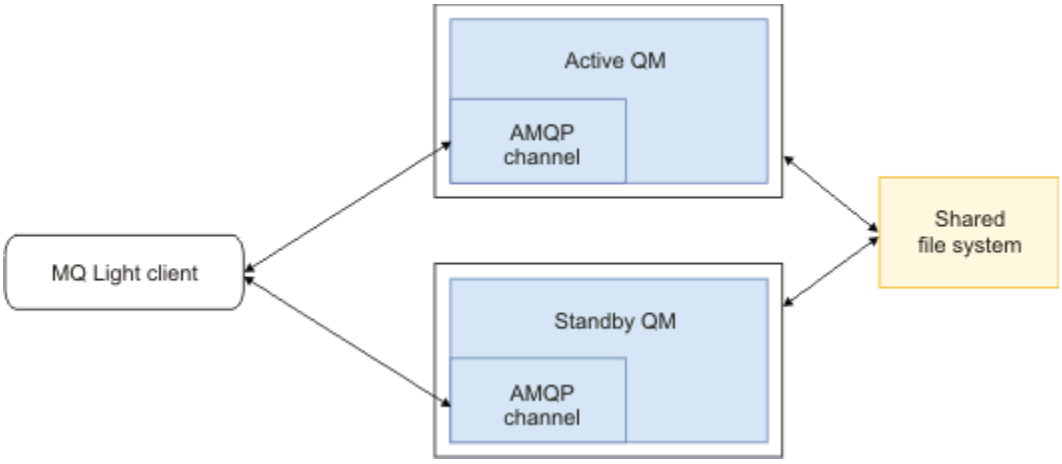
Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Configuration d'un client AMQP pour la haute disponibilité

Vous pouvez configurer des applications AMQP 1.0 pour qu'elles se connectent à l'instance active d'un gestionnaire de files d'attente multi-instance IBM MQ et basculent vers l'instance de secours du gestionnaire de files d'attente multi-instance dans une paire à haute disponibilité. Pour ce faire, vous configurez l'application AMQP avec deux adresses IP et deux paires de ports.

Vous pouvez configurer l'API client AMQP avec une fonction personnalisée, qui est appelée si le client perd sa connexion au serveur. La fonction peut établir la connexion à une adresse IP alternative, par exemple un gestionnaire de files d'attente IBM MQ de secours ou l'adresse IP d'origine. Pour les autres clients AMQP, si le client prend en charge la configuration de plusieurs noeuds finaux de connexion, configurez l'application avec deux paires hôte-port et utilisez les fonctions de reconnexion fournies par la bibliothèque AMQP pour passer au gestionnaire de files d'attente de secours.



Tâches associées

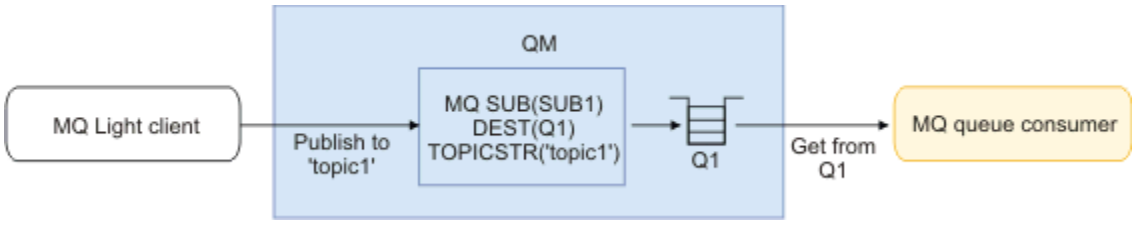
Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Configuration de la publication / l'abonnement pour les clients AMQP

Les clients AMQP peuvent publier dans une rubrique avec un abonnement IBM MQ qui achemine les messages vers une file d'attente IBM MQ lue par une application existante. Si vous souhaitez qu'une application AMQP 1.0 envoie des messages à une application IBM MQ existante configurée pour lire à partir d'une file d'attente, vous devez définir un abonnement IBM MQ administré sur le gestionnaire de files d'attente.

Configurez l'abonnement pour qu'il utilise un modèle de rubrique qui correspond à la chaîne de rubrique utilisée par l'application AMQP. Comme destination d'abonnement, définissez le nom de la file d'attente depuis laquelle l'application IBM MQ obtient ou parcourt les messages.



Tâches associées

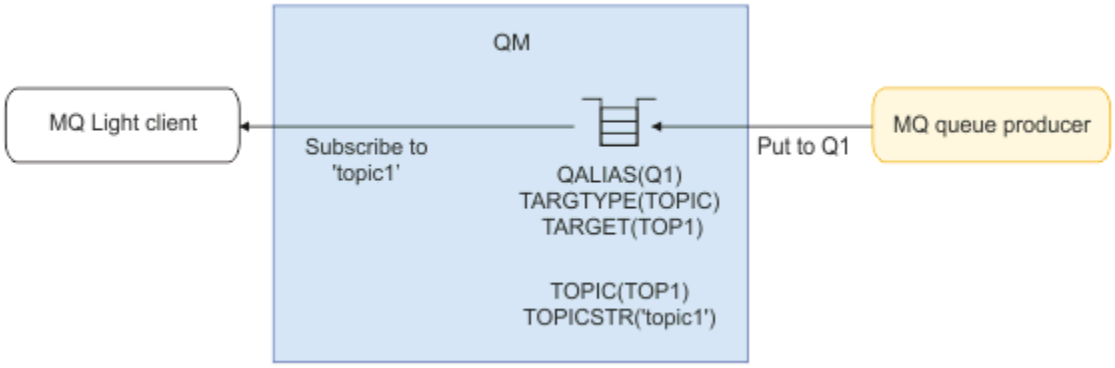
Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Client AMQP utilisant un alias de file d'attente pour recevoir des messages d'une application IBM MQ

Un client AMQP peut s'abonner à une rubrique et recevoir des messages insérés dans une file d'attente alias par une application IBM MQ . Si vous souhaitez qu'une application AMQP 1.0 reçoive des messages d'une application IBM MQ existante configurée pour placer des messages dans une file d'attente, vous devez définir un alias de file d'attente (QALIAS) sur le gestionnaire de files d'attente.

L'alias de file d'attente doit avoir le même nom que la file d'attente que l'application IBM MQ ouvre pour le placement. Il doit spécifier un type de base TOPIC et un objet de base d'un objet de rubrique IBM MQ dont la chaîne de rubrique correspond au modèle de rubrique auquel l'application AMQP est abonnée.



Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Client AMQP soumettant des demandes à un serveur d'applications et consommant des réponses provenant d'un serveur d'applications

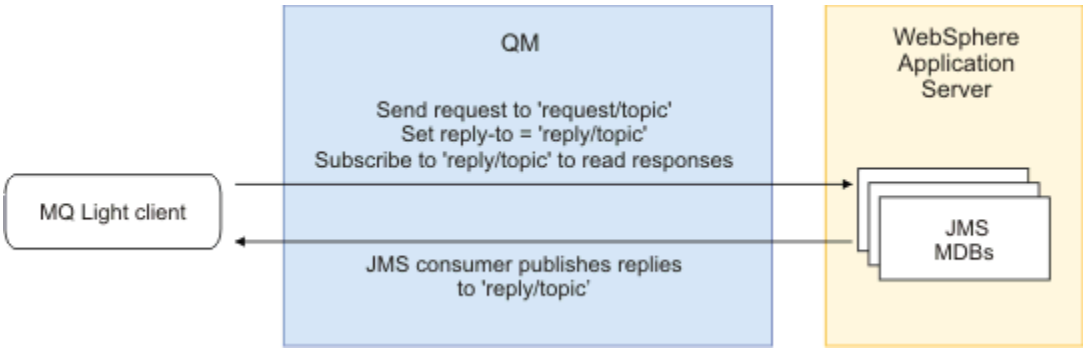
Un client AMQP peut soumettre des demandes à un bean géré par message s'exécutant sur un serveur d'applications et consommer des réponses à partir d'une rubrique de réponse. IBM MQ prend en charge les applications AMQP 1.0 définissant une rubrique de réponse dans les messages publiés par IBM MQ . Lorsqu'un message AMQP est publié avec l'attribut de réponse défini, la valeur de la zone de réponse est définie en tant que propriété JMS pour les consommateurs JMS. Ce paramètre permet aux consommateurs JMS de lire la rubrique de réponse depuis le message et d'envoyer un message de réponse au client AMQP.

La propriété JMS est **JMSReplyTo**. Le type de la chaîne de réponse AMQP doit être l'un des suivants :

- Une chaîne de rubrique, Exemple: `'reply/topic'`
- URL d'adresse AMQP au format `amqp://host:port/[topic-string]`. Exemple : `amqp://localhost:5672/reply/topic`

Si vous spécifiez une URL d'adresse AMQP comme zone de réponse, tout sauf la chaîne de rubrique à la fin de l'URL est supprimée avant la définition de la propriété **JMSReplyTo** .

Pour plus d'informations sur les mappages d'une adresse de réponse AMQP à une propriété **JMSReplyTo** , voir «Mappage de zones AMQP à des zones IBM MQ (messages entrants)», à la page 702



Tâches associées

Création et utilisation de canaux AMQP

Sécurisation des clients AMQP

ALW Interopérabilité entre les applications MQ Light et Apache Qpid JMS

Les applications MQ Light et Apache Qpid JMS fonctionnent de la même manière et, lors de l'abonnement à une rubrique, créent des abonnements IBM MQ qui suivent la même convention de dénomination.

Abonnement privé, non partagé

Le nom de l'abonnement IBM MQ créé par l'application est `:private:<clientid>:<topicstring>`.

Une application utilisant un ID client différent ne peut pas accéder aux abonnements créés par d'autres applications, car le nom de l'abonnement est automatiquement généré et inclut l'ID client AMQP.

Les applications Apache Qpid JMS et MQ Light utilisent cette convention de dénomination pour les abonnements privés.

Abonnements partagés globalement

Le nom d'un abonnement IBM MQ partagé globalement créé par un client AMQP est `:share:<sharename>:<topicstring>`.

Si plusieurs applications avec des ID client AMQP différents spécifient le même nom de partage et la même chaîne de rubrique, elles partagent un seul abonnement et peuvent travailler ensemble pour traiter les messages de cet abonnement. Vous pouvez utiliser ce modèle si vous souhaitez mettre à l'échelle le nombre d'applications worker qui drainent des messages d'un abonnement.

Les applications Apache Qpid JMS et MQ Light utilisent cette convention de dénomination pour les abonnements partagés globalement. Dans le cas d' Apache Qpid JMS, cela nécessite que la connexion JMS n'ait pas d'ID client spécifié dessus.

La bibliothèque Apache Qpid JMS génère automatiquement un ID client AMQP, mais cet ID client n'est pas utilisé pour la dénomination des abonnements IBM MQ .

Remarque : Les abonnements partagés globalement sont toujours limités à un gestionnaire de files d'attente individuel.

Abonnements partagés privés

Le nom d'un abonnement IBM MQ partagé en privé créé par un client AMQP est `:privateshare:<clientid>:<sharename>:<topicstring>`.

Si plusieurs unités d'exécution d'une même application Apache Qpid JMS utilisent le même nom de partage et la même chaîne de rubrique et qu'un ID client a été configuré sur la connexion JMS , ces unités d'exécution partagent le même objet d'abonnement IBM MQ .

Toutefois, les autres connexions Apache Qpid JMS ne peuvent pas partager l'abonnement car elles doivent utiliser un ID client différent.

Les clients MQ Light ne prennent pas en charge le concept d'abonnements partagés privés et ne peuvent pas consommer de messages à partir d'un abonnement partagé privé créé par une application Apache Qpid JMS .

Abonnements IBM MQ JMS

Les abonnements IBM MQ JMS utilisent un schéma de dénomination différent des canaux AMQP. Il n'est pas possible pour les applications MQ Light ou Apache Qpid JMS de partager des abonnements avec des applications IBM MQ JMS .

Concepts associés

Développement d'applications client AMQP
La prise en charge par IBM MQ des API AMQP permet à un administrateur IBM MQ de créer un canal AMQP. Lorsqu'il est démarré, ce canal définit un numéro de port qui accepte les connexions des applications client AMQP.

ALW Propriétés de contrôle du programme d'écoute AMQP IBM MQ

Pour de meilleures performances dans une application à unités d'exécution multiples, vous pouvez optimiser le nombre d'unités d'exécution de tâche que le service AMQP doit utiliser en configurant une propriété dans le fichier de propriétés AMQP.

Vous pouvez configurer les propriétés du service d'écoute AMQP dans les fichiers de propriétés suivants:

- Windows Sur les systèmes Windows : `amqp_win.properties` .
- Linux AIX Sur les systèmes AIX and Linux : `amqp_unix.properties` .

Les propriétés que vous pouvez configurer sont les suivantes:

Tableau 105. Propriétés du service d'écoute AMQP	
Propriété	**Description**
com.ibm.mq.MQXR.Workers	Nombre d'unités d'exécution de tâche de serveur créées par le service d'écoute AMQP. Si cette valeur n'est pas spécifiée, elle est par défaut égale au nombre de processeurs logiques sur le système.
MQIBindType	Type de liaison du service AMQP: FASTPATH, SHAREDou ISOLÉ. La valeur par défaut est FASTPATH.

Le service d'écoute AMQP équilibre la charge de travail de connexion client entre un certain nombre d'unités d'exécution de tâche. Le nombre d'unités d'exécution de tâche que le service AMQP doit utiliser peut être spécifié à l'aide de la propriété **com.ibm.mq.MQXR.Workers** .

L'administrateur du gestionnaire de files d'attente IBM MQ peut optimiser le nombre d'unités d'exécution de tâche pour de meilleures performances dans une application à unités d'exécution multiples. En règle générale, les meilleures performances sont obtenues lorsque le nombre d'unités d'exécution de tâche correspond au nombre de processeurs logiques sur le système. Toutefois, cela peut ne pas toujours être le cas pour certaines configurations de machine et certaines caractéristiques de charge du client, de sorte qu'un élément d'optimisation peut être nécessaire pour trouver la valeur optimale pour le nombre d'unités d'exécution de tâche.

Avant de procéder à l'optimisation, assurez-vous de bien comprendre la nature de vos applications client et de leurs charges de travail. La mesure des performances de votre application à l'aide de différents nombres d'unités d'exécution et de tests de performances doit vous aider à déterminer la valeur optimale du nombre d'unités d'exécution de tâche.

Remarque : MQ Appliance Ces propriétés ne sont pas applicables sous IBM MQ Appliance. Les valeurs par défaut sont utilisées sous IBM MQ Appliance.

Développement d'applications REST avec IBM MQ

Vous pouvez développer des applications REST pour envoyer et recevoir des messages. IBM MQ prend en charge différentes API REST en fonction de la plateforme et des capacités.

Les options suivantes sont les options prises en charge par IBM MQ que vous pouvez sélectionner pour envoyer et recevoir des messages à partir de IBM MQ:

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

IBM MQ messaging REST API

Vous pouvez utiliser messaging REST API pour envoyer, recevoir et parcourir des messages IBM MQ au format texte en clair. messaging REST API est activé par défaut.

La prise en charge est fournie pour un certain nombre d'en-têtes HTTP différents qui peuvent être utilisés pour définir des propriétés de message communes.

messaging REST API est entièrement intégré à la sécurité IBM MQ . Pour utiliser le messaging REST API, les utilisateurs doivent être authentifiés auprès du serveur mqweb et être membres du rôle MQWebUser .

Pour plus d'informations, reportez-vous à la section «Messagerie à l'aide de REST API», à la page 721. Voir aussi Tutoriel: Initiation à la IBM MQ REST API sur IBM Developer, qui inclut des exemples Go et Node.js pour l'utilisation de l'API REST de messagerie.

IBM z/OS Connect EE

IBM z/OS Connect EE est un produit z/OS qui vous permet de générer des API REST sur des actifs z/OS existants, tels que des transactions CICS ou IMS , ainsi que des files d'attente et des rubriques IBM MQ . L'actif z/OS existant est masqué pour l'utilisateur. Cela vous permet d'activer REST vos actifs sans les modifier ni les applications existantes qui les utilisent.

IBM z/OS Connect EE fournit une transformation automatique des données à traduire entre les données JSON utilisées par les API REST et les structures de langage plus traditionnelles, par exemple COBOL, attendues par de nombreuses applications de grand système.

Le kit d'outils d'API Eclipse basé sur IBM z/OS Connect EE peut être utilisé pour créer une API RESTful complète utilisant des paramètres de requête et des segments de chemin d'URL, en manipulant le format JSON lorsqu'il passe par l'environnement d'exécution IBM z/OS Connect EE.

IBM z/OS Connect EE peut être utilisé pour exposer des files d'attente et des rubriques IBM MQ en tant qu'API RESTful via le fournisseur de services IBM MQ . Deux types de service différents sont pris en charge:

- Services unidirectionnels: ils fournissent une API REST qui permet d'effectuer une seule opération IBM MQ sur une file d'attente ou une rubrique. Selon la configuration exacte, une demande HTTP peut entraîner l'envoi d'un message à une file d'attente ou la publication d'un message dans une rubrique ; ou une demande HTTP peut entraîner la réception d'un message de façon destructive à partir d'une file d'attente
- Services bidirectionnelle: ils fournissent une API REST sur une paire de files d'attente utilisée par une application de type demande-réponse de back end. Les appelants émettent une demande HTTP au service bidirectionnel. Le contenu de la demande HTTP est transformé de JSON en structure de langage traditionnelle et placé dans une file d'attente de demandes où il est traité par l'application dorsale et une réponse est placée dans la file d'attente de réponses. Cette réponse est extraite par le service, convertie de la structure de langage traditionnelle en JSON et renvoyée à l'appelant en tant que corps de réponse POST.

Pour plus d'informations sur IBM z/OS Connect EE, voir z/OS Connect EE.

Pour plus d'informations sur le fournisseur de services IBM MQ, voir Utilisation du fournisseur de services IBM MQ.

IBM Integration Bus

IBM Integration Bus est la technologie d'intégration leader d' IBMqui peut être utilisée pour connecter des applications et des systèmes entre eux, quels que soient les formats de message et les protocoles qu'ils prennent en charge.

IBM Integration Bus a toujours pris en charge IBM MQ et fournit des noeuds *HTTPInput* et *HTTPRequest* qui peuvent être utilisés pour construire une interface RESTful sur IBM MQ, ainsi que de nombreux autres systèmes tels que des bases de données.

IBM Integration Bus peut être utilisé pour faire beaucoup plus que de fournir une interface REST simple sur IBM MQ. Ses capacités peuvent être utilisées pour fournir une manipulation de contenu avancée, un enrichissement de contenu et de nombreuses autres améliorations dans le cadre d'un REST API.

Pour plus d'informations, voir l' exemple de technologie qui expose une interface JSON sur REST par-dessus une application IBM MQ qui attend une charge XML.

DataPower

La passerelle DataPower est une passerelle multicanal unique qui permet d'assurer la sécurité, le contrôle, l'intégration et l'accès optimisé à une gamme de systèmes, y compris IBM MQ. Il est fourni à la fois dans les facteurs de forme matérielle et virtuelle.

L'un des services fournis par DataPower est une passerelle multiprotocole qui peut prendre une entrée dans un protocole et générer une sortie dans un autre protocole. En particulier, DataPower peut être configuré pour accepter les données HTTP (S) et les acheminer vers IBM MQ via une connexion client, qui

peut être utilisée pour générer une interface REST sur IBM MQ. D'autres services DataPower , tels que la transformation, peuvent également être utilisés pour améliorer l'interface REST.

Pour plus d'informations, voir Passerelle multiprotocole.

Messagerie à l'aide de REST API

Vous pouvez utiliser messaging REST API pour effectuer une messagerie point-à-point simple et une messagerie de publication. Vous pouvez publier des messages dans une rubrique, envoyer des messages à une file d'attente, parcourir les messages d'une file d'attente et extraire de façon destructive des messages d'une file d'attente. Les informations sont envoyées à messaging REST API et reçues de ce dernier au format texte en clair.

Avant de commencer

Remarque :

- messaging REST API est activé par défaut. Vous pouvez désactiver le messaging REST API pour empêcher toute messagerie. Pour plus d'informations sur l'activation ou la désactivation du messaging REST API, voir Configuration de messaging REST API.
- messaging REST API est intégré à la sécurité IBM MQ . Pour utiliser le messaging REST API, les utilisateurs doivent être authentifiés auprès du serveur mqweb et être membres du rôle MQWebUser . L'utilisateur doit également être autorisé à accéder à la file d'attente ou à la rubrique indiquée. Pour plus d'informations sur la sécurité pour REST API, voir Sécurité IBM MQ Console et REST API.
- Si vous utilisez Advanced Message Security (AMS) avec messaging REST API, notez que tous les messages sont chiffrés à l'aide du contexte du serveur mqweb et non du contexte de l'utilisateur qui envoie le message.
- Lors de la réception ou de la consultation d'un message, seuls les messages formatés IBM MQ `MQSTR` ou `JMS TextMessage` sont pris en charge. Par la suite, tous les messages sont reçus de façon destructive sous le point de synchronisation et tous les messages non gérés sont laissés dans la file d'attente. La file d'attente IBM MQ peut être configurée pour déplacer ces messages incohérents vers une autre destination. Pour plus d'informations, reportez-vous à la section «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.
- Le messaging REST API ne vous permet pas de distribuer les messages une fois et une seule fois avec le support transactionnel. Si une demande HTTP POST est émise et que la connexion échoue avant qu'une réponse HTTP ne soit reçue par le client, ce dernier ne peut pas déterminer immédiatement si le message a été envoyé à la file d'attente spécifiée ou publié dans la rubrique spécifiée. Si une instruction HTTP DELETE est émise et que la connexion échoue avant la réception d'une réponse HTTP par le client, il se peut qu'un message ait été extrait de la file d'attente de façon destructive et qu'il ait été perdu, car il n'est pas possible de réexécuter la commande destructive.
- Depuis la IBM MQ 9.3.0, les nouvelles lignes des chaînes entrantes ne sont plus supprimées par l'opération HTTP POST. Les applications REST qui utilisent des versions antérieures ne doivent pas utiliser de nouvelles lignes dans les messages envoyés ou publiés à l'aide de l'API REST, car elles seront perdues.

Procédure

- «Guide d'initiation à messaging REST API», à la page 722
- «Utilisation de messaging REST API», à la page 724
- REST API traitement des erreurs
- ReconnaissanceREST API
- Prise en charge des langues nationales dansREST API

Référence associée

Informations de référence sur la messagerie REST API

Information associée

Tutoriel: Initiation à la messagerie IBM MQ REST API

Guide d'initiation à messaging REST API

Initiez-vous rapidement à messaging REST API et essayez quelques exemples de commandes à l'aide de cURL.

Avant de commencer

Pour vous initier à l'utilisation du messaging REST API, les exemples de cette tâche comportent les exigences suivantes:

- Les exemples utilisent cURL pour envoyer des demandes REST afin d'insérer et d'extraire des messages d'une file d'attente. Par conséquent, pour effectuer cette tâche, vous devez installer cURL sur votre système.
- Les exemples utilisent un gestionnaire de files d'attente `QM1`. Créez un gestionnaire de files d'attente portant le même nom ou remplacez un gestionnaire de files d'attente existant sur votre système. Le gestionnaire de files d'attente doit se trouver sur la même machine que le serveur mqweb.
- Pour effectuer cette tâche, vous devez être un utilisateur doté de certains privilèges vous permettant d'employer la commande **dspmqweb** :
  - z/OS Sur z/OS, vous devez disposez des droits permettant d'exécuter la commande **dspmqweb** et de l'accès en écriture sur le fichier `mqwebuser.xml`.
  - Multi Sur tous les autres systèmes d'exploitation, vous devez être un utilisateur privilégié.

IBM i Sous IBM i, les commandes doivent être exécutées dans QSHELL.

Procédure

1. Vérifiez que le serveur mqweb est configuré pour messaging REST API:
   - Vérifiez que vous avez configuré le serveur mqweb pour qu'il soit utilisé par administrative REST API, administrative REST API for MFT, messaging REST APIou IBM MQ Console.Pour plus d'informations sur la configuration du serveur mqweb avec un registre de base, voir Configuration de base pour le serveur mqweb.
   - Si le serveur mqweb est déjà configuré, assurez-vous d'avoir ajouté les utilisateurs appropriés pour activer la messagerie à l'étape 5 de la section Configuration de base du serveur mqweb.
     - Si **mqRestMessagingAdoptWebUserContext** est défini sur `true` dans la configuration du serveur mqweb, les utilisateurs de messaging REST API doivent être membres du rôle `MQWebUser` . Les rôles `MQWebAdmin` et `MQWebAdminRO` ne sont pas applicables pour le messaging REST API. Les utilisateurs doivent également être autorisés à accéder aux files d'attente et aux rubriques utilisées pour la messagerie via OAM ou RACF.
     - Si **mqRestMessagingAdoptWebUserContext** est défini sur `false` dans la configuration du serveur mqweb, l'ID utilisateur utilisé pour démarrer le serveur mqweb doit être autorisé à accéder aux files d'attente utilisées pour la messagerie via OAM ou RACF.
2. z/OS
   Sur z/OS, définissez la variable d'environnement WLP_USER_DIR afin de pouvoir utiliser la commande **dspmqweb** . Définissez la variable de telle sorte qu'elle pointe vers votre configuration de serveur mqweb à l'aide de la commande suivante :

   ```
   export WLP_USER_DIR=WLP_user_directory
   ```

   où *WLP_user_directory* est le nom du répertoire transmis à `crtmqweb`. Par exemple :

   ```
   export WLP_USER_DIR=/var/mqm/web/installation1
   ```

Pour plus d'informations, voir Creating the mqweb server.

3. Déterminez l'URL REST API en entrant la commande suivante:

```
dspmqweb status
```

Les exemples des étapes suivantes supposent que votre URL REST API est l'URL par défaut `https://localhost:9443/ibmmq/rest/v2/`. Si votre URL est différente de l'URL par défaut, remplacez-la dans les étapes ci-après.

4. Créez une file d'attente, MSGQ, sur le gestionnaire de files d'attente QM1. Cette file d'attente est utilisée pour la messagerie. Utilisez l'une des méthodes suivantes :

   - Utilisez une demande POST sur la ressource `mqsc` du administrative REST API, en vous authentifiant en tant qu'utilisateur `mqadmin` :

     ```
     curl -k https://localhost:9443/ibmmq/rest/v2/admin/action/qmgr/QM1/mqsc -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/
     json" --data "{\"type\": \"runCommandJSON\",\"command\": \"define\", \"qualifier\":
     \"qlocal\",\"name\": \"MSGQ\"}"
     ```

   - Utilisez les commandes MQSC:

     z/OS Sous z/OS, utilisez une source 2CR à la place de la commande **runmqsc** . Pour plus d'informations, voir Sources à partir desquelles vous pouvez émettre des commandes MQSC et PCF sous IBM MQ for z/OS.

     a. Démarrez **runmqsc** pour le gestionnaire de files d'attente en entrant la commande suivante:

     ```
     runmqsc QM1
     ```

     b. Utilisez la commande **DEFINE QLOCAL** MQSC pour créer la file d'attente:

     ```
     DEFINE QLOCAL(MSGQ)
     ```

     c. Quittez **runmqsc** en entrant la commande suivante:

     ```
     end
     ```

5. Accordez à l'utilisateur que vous avez ajouté à `mqwebuser.xml` à l'étape 5 de la configuration de base pour que le serveur mqweb puisse accéder à la file d'attente MSGQ. Remplacez votre utilisateur sur lequel `myuser` est utilisé:

   - z/OS Sous z/OS :

     a. Accordez à votre utilisateur l'accès à la file d'attente:

     ```
     RDEFINE MQQUEUE hlq.MSGQ UACC(NONE)
     PERMIT hlq.MSGQ CLASS(MQQUEUE) ID(MYUSER) ACCESS(UPDATE)
     ```

     b. Accordez à l'ID utilisateur de la tâche démarrée mqweb l'accès permettant de définir tous les contextes dans la file d'attente:

     ```
     RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
     PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
     ```

   - Multi Sur tous les autres systèmes d'exploitation, si votre utilisateur appartient au groupe mqm, des droits d'accès sont déjà accordés. Sinon, entrez les commandes suivantes:

     a. Démarrez **runmqsc** pour le gestionnaire de files d'attente en entrant la commande suivante:

     ```
     runmqsc QM1
     ```

b. Utilisez la commande **SET AUTHREC** MQSC pour donner à votre utilisateur les droits de navigation, d'interrogation, d'obtention et d'insertion sur la file d'attente:

```
SET AUTHREC PROFILE(MSGQ) OBJTYPE(QUEUE) +
PRINCIPAL(myuser) AUTHADD(BROWSE, INQ, GET, PUT)
```

c. Quittez **runmqsc** en entrant la commande suivante:

```
end
```

6. Insérez un message avec le contenu `Hello World!` dans la file d'attente `MSGQ` du gestionnaire de files d'attente `QM1`, à l'aide d'une demande POST sur la ressource `message` . Remplacez votre ID utilisateur et votre mot de passe par ceux de `mqwebuser.xml` for `myuser` et `mypassword`:

   L'authentification de base est utilisée et un en-tête HTTP `ibm-mq-rest-csrf-token` avec une valeur arbitraire est défini dans la demande REST cURL . Cet en-tête supplémentaire est requis pour les demandes POST, PATCH et DELETE.

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X
POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: text/
plain;charset=utf-8" --data "Hello World!"
```

7. Extrayez de façon destructive le message de la file d'attente `Hello World!` de la file d'attente `MSGQ` du gestionnaire de files d'attente `QM1`, en utilisant une demande DELETE sur la ressource `message` . Remplacez votre ID utilisateur et votre mot de passe par ceux de `mqwebuser.xml` for `myuser` et `mypassword`:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/MSGQ/message -X DELETE
-u myuser:mypassword -H "ibm-mq-rest-csrf-token: value"
```

   Le message `Hello World!` est renvoyé.

Que faire ensuite

- Les exemples utilisent l'authentification de base pour sécuriser la demande. Vous pouvez utiliser à la place l'authentification basée sur un jeton ou l'authentification basé sur le client. Pour plus d'informations, voir Utilisation de l'authentification par certificat client avec l'REST API et IBM MQ Console et Utilisation de l'authentification basée sur un jeton avec l'REST API.
- En savoir plus sur l'utilisation de messaging REST API et la construction d'URL avec des paramètres de requête: «Utilisation de messaging REST API», à la page 724.
- Lorsque vous utilisez messaging REST API, les connexions au gestionnaire de files d'attente sont regroupées pour optimiser les performances. Vous pouvez configurer la taille maximale du pool et l'action à effectuer lorsque toutes les connexions du pool sont utilisées: Configuration du messaging REST API.
- Parcourez les informations de référence pour les ressources messaging REST API disponibles et tous les paramètres de requête disponibles: messaging REST API reference.
- Découvrez administrative REST API, une interface RESTful pour l'administration de IBM MQ : Administration à l'aide de REST API.
- Découvrez IBM MQ Console, une interface graphique basée sur un navigateur: Administration à l'aide de IBM MQ Console.

Utilisation de messaging REST API

Lorsque vous utilisez messaging REST API, vous appelez des méthodes HTTP sur des URL pour envoyer et recevoir des messages IBM MQ . La méthode HTTP, par exemple POST, représente le type d'action à effectuer sur l'objet représenté par l'URL. Des informations supplémentaires sur l'action peuvent être codées dans les paramètres de requête. Des informations sur le résultat de l'exécution de l'action peuvent être renvoyées en tant que corps de la réponse HTTP.

Avant de commencer

Tenez compte des points suivants avant d'utiliser messaging REST API:

- Vous devez vous authentifier auprès du serveur mqweb pour pouvoir utiliser messaging REST API. Vous pouvez vous authentifier à l'aide de l'authentification de base HTTP, de l'authentification par certificat client ou de l'authentification basée sur un jeton. Pour plus d'informations sur l'utilisation de ces méthodes d'authentification, voir SécuritéIBM MQ Console et REST API.
- REST API est sensible à la casse. Par exemple, une demande HTTP POST sur l'URL suivante génère une erreur si le gestionnaire de files d'attente est appelé `qmgr1`.

  ```
  /ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
  ```

- V 9.4.0 Si vous vous connectez à un gestionnaire de files d'attente éloignées avec messaging REST API, vous devez utiliser le nom unique de la connexion de gestionnaire de files d'attente au lieu du nom du gestionnaire de files d'attente.
- Tous les caractères pouvant être utilisés dans les noms d'objet IBM MQ ne peuvent pas être directement codés dans une URL. Pour coder correctement ces caractères, vous devez utiliser le codage d'URL approprié:
  - Une barre oblique doit être codée en tant que %2F.
  - Un signe de pourcentage doit être codé en tant que %25.
  - Un point doit être codé en tant que %2E.
  - Un point d'interrogation doit être codé en tant que %3F.
- Lors de la réception ou de la consultation d'un message, seuls les messages formatés IBM MQ `MQSTR` et `JMS TextMessage` sont pris en charge. Par la suite, tous les messages sont reçus de façon destructive sous le point de synchronisation et tous les messages non gérés sont laissés dans la file d'attente. La file d'attente IBM MQ peut être configurée pour déplacer ces messages incohérents vers une autre destination. Pour plus d'informations, reportez-vous à la section «Traitement des messages incohérents dans IBM MQ classes for JMS», à la page 240.

Pourquoi et quand exécuter cette tâche

Lorsque vous utilisez REST API pour effectuer une action de messagerie sur un objet de file d'attente IBM MQ , vous devez d'abord construire une URL pour représenter cet objet. Chaque URL commence par un préfixe, qui décrit le nom d'hôte et le port auxquels envoyer la demande. Le reste de l'URL décrit un objet particulier, ou une route vers cet objet, appelé ressource.

L'action de messagerie à effectuer sur la ressource définit si l'URL a besoin de paramètres de requête ou non. Il définit également la méthode HTTP utilisée et indique si des informations supplémentaires sont envoyées à l'URL ou renvoyées à partir de celle-ci. Les informations supplémentaires peuvent faire partie de la demande HTTP ou être renvoyées dans le cadre de la réponse HTTP.

Après avoir construit l'URL, vous pouvez envoyer la demande HTTP à IBM MQ. Vous pouvez envoyer la demande à l'aide de l'implémentation HTTP qui est intégrée au langage de programmation de votre choix. Vous pouvez également envoyer la demande à l'aide d'outils de ligne de commande tels que cURL, d'un navigateur Web ou d'un module complémentaire de navigateur Web.

Important : Vous devez au minimum effectuer les étapes «1.a», à la page 725 et «1.b», à la page 726.

Procédure

1. Construisez l'URL:
   a) Déterminez l'URL de préfixe en entrant la commande suivante:

   ```
   dspmqweb status
   ```

   L'URL que vous souhaitez utiliser inclut la phrase `/ibmmq/rest/` .

b) Ajoutez la file d'attente et les ressources de gestionnaire de files d'attente associées à utiliser pour la messagerie dans le chemin d'URL.

Dans la référence de messagerie, les segments de variable peuvent être identifiés dans l'URL par les accolades qui les entourent *{ }*. Pour plus d'informations, voir /messaging/qmgr/{qmgrName}/queue/{queueName}/message.

Par exemple, pour interagir avec la file d'attente *Q1* associée au gestionnaire de files d'attente *QM1*, ajoutez `/qmgr` et `/queue` à l'URL de préfixe pour créer l'URL suivante:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
```

Conseil : V 9.4.0 Si le gestionnaire de files d'attente est un gestionnaire de files d'attente éloignées, vous devez utiliser le nom unique du gestionnaire de files d'attente à la place du nom du gestionnaire de files d'attente. Le gestionnaire de files d'attente éloignées doit être configuré pour pouvoir être utilisé avec messaging REST API. Pour plus d'informations, voir «Configuration d'un gestionnaire de files d'attente éloignées à utiliser avec messaging REST API», à la page 726.

c) Facultatif : Ajoutez un paramètre de requête facultatif à l'URL.

Ajouter un point d'interrogation,?, paramètre de requête, signe égal = et valeur de l'URL.

Par exemple, pour attendre un maximum de 30 secondes que le message suivant soit disponible, créez l'URL suivante:

```
https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
```

d) Facultatif : Ajoutez d'autres paramètres de requête facultatifs à l'URL.

Ajoutez une perluète, &, à l'URL, puis répétez l' étape 1c.

2. Appelez la méthode HTTP appropriée sur l'URL. Spécifiez une charge de message facultative et fournissez les données d'identification de sécurité appropriées pour l'authentification. Exemple :
   - Utilisez l'implémentation HTTP/REST du langage de programmation de votre choix.
   - Utilisez un outil tel qu'un module complémentaire de navigateur de client REST ou cURL.

V 9.4.0 Configuration d'un gestionnaire de files d'attente éloignées à utiliser avec messaging REST API

Vous pouvez utiliser messaging REST API pour vous connecter à des gestionnaires de files d'attente éloignées pour la messagerie. Avant de pouvoir vous connecter à un gestionnaire de files d'attente éloignées, vous devez configurer le gestionnaire de files d'attente éloignées. Vous pouvez ensuite vous connecter au gestionnaire de files d'attente éloignées à l'aide du nom unique défini dans les informations de configuration.

Avant de commencer

- Vérifiez que vous avez configuré le serveur mqweb pour qu'il soit utilisé par administrative REST API, administrative REST API for MFT, messaging REST APIou IBM MQ Console.Pour plus d'informations sur la configuration du serveur mqweb avec un registre de base, voir Configuration de base pour le serveur mqweb.
- Si le serveur mqweb est déjà configuré, assurez-vous d'avoir ajouté les utilisateurs appropriés pour activer la messagerie à l'étape 5 de la section Configuration de base pour le serveur mqweb. Les utilisateurs du messaging REST API doivent être membres du rôle `MQWebUser` . Les rôles `MQWebAdmin` et `MQWebAdminRO` ne sont pas applicables pour le messaging REST API.
  - Si **`mqRestMessagingAdoptWebUserContext`** est défini sur `true` dans la configuration du serveur mqweb, les utilisateurs du rôle `MQWebUser` doivent être autorisés à accéder aux files d'attente et aux rubriques utilisées pour la messagerie via OAM ou RACF.

  - Si **mqRestMessagingAdoptWebUserContext** est défini sur `false` dans la configuration du serveur mqweb, l'ID utilisateur utilisé pour démarrer le serveur mqweb doit être autorisé à accéder aux files d'attente et aux rubriques utilisées pour la messagerie via OAM ou RACF.
- Vérifiez que messaging REST API est configuré pour se connecter aux gestionnaires de files d'attente éloignées. Pour plus d'informations, voir Configuration du mode de connexion pour messaging REST API.

Pourquoi et quand exécuter cette tâche

Vous pouvez vous connecter à des gestionnaires de files d'attente éloignées à l'aide de l' messaging REST API. Un gestionnaire de files d'attente éloignées peut être un gestionnaire de files d'attente sur un autre système, un gestionnaire de files d'attente dans une autre installation ou un gestionnaire de files d'attente dans la même installation que le serveur mqweb.

Pour vous connecter à un gestionnaire de files d'attente éloignées, vous devez effectuer les étapes de configuration suivantes:

- Configurez un canal de connexion serveur et un programme d'écoute.
- Accordez des droits à un utilisateur approprié pour accéder au gestionnaire de files d'attente.
- Créez un fichier CCDT contenant les informations de connexion pour le gestionnaire de files d'attente.
- Ajoutez les informations de connexion au messaging REST API à l'aide de la commande **setmqweb remote** .

Vous pouvez ensuite utiliser le gestionnaire de files d'attente éloignées en fournissant le nom unique dans l'URL de la ressource à la place du nom du gestionnaire de files d'attente.

Vous pouvez également configurer vos gestionnaires de files d'attente éloignées dans le cadre d'un groupe de gestionnaires de files d'attente. Pour plus d'informations, voir «Configuration d'un groupe de gestionnaires de files d'attente à utiliser avec l'API REST de messagerie», à la page 730.

Procédure

1. Sur le gestionnaire de files d'attente éloignées, créez un canal de connexion serveur pour autoriser les connexions distantes au gestionnaire de files d'attente. Vous pouvez créer des canaux de connexion serveur à l'aide de la commande **DEFINE CHANNEL** MQSC sur la ligne de commande.
   Par exemple, pour créer un canal de connexion serveur `QM1.SVRCONN` pour le gestionnaire de files d'attente QM1, entrez la commande suivante:

   ```
   DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   ```

   Pour plus d'informations sur **DEFINE CHANNEL** et les options disponibles, voir DEFINE CHANNEL.
2. Vérifiez qu'un utilisateur approprié est autorisé à accéder au gestionnaire de files d'attente. Cet utilisateur doit également être autorisé à accéder aux files d'attente ou aux rubriques que vous utilisez pour la messagerie. L'utilisateur a besoin des droits `connect`, `inquire`, `alternate user`et `set context` sur le gestionnaire de files d'attente. Sous UNIX, Linux, and Windows , utilisez la commande de contrôle **setmqaut** sur la ligne de commande. Sous z/OS, définissez des profils RACF pour accorder à l'utilisateur autorisé l'accès au gestionnaire de files d'attente.
   Par exemple, sous UNIX, Linux, and Windows, pour autoriser un utilisateur, `exampleUser`, à accéder au gestionnaire de files d'attente QM1, entrez la commande suivante:

   ```
   setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
   ```

   Pour plus d'informations sur l'utilisateur devant être autorisé, voir «Détermination du principal de sécurité utilisé par messaging REST API», à la page 733.
3. ALW
   S'il n'existe aucun programme d'écoute sur le gestionnaire de files d'attente éloignées, créez un programme d'écoute pour accepter les connexions réseau entrantes à l'aide de la commande **DEFINE LISTENER** MQSC sur la ligne de commande.

Par exemple, pour créer un programme d'écoute REMOTE.LISTENER sur le port 1414 pour le gestionnaire de files d'attente éloignées QM1, entrez la commande suivante:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. Vérifiez que le programme d'écoute est en cours d'exécution à l'aide de la commande **START LISTENER** MQSC sur la ligne de commande:

   ALW Par exemple, sous AIX, Linux, and Windows , pour démarrer le programme d'écoute REMOTE.LISTENER pour le gestionnaire de files d'attente QM1, entrez la commande suivante:

   ```
   runmqsc QM1
   START LISTENER(REMOTE.LISTENER)
   end
   ```

   z/OS Par exemple, sous z/OS, pour démarrer le programme d'écoute, entrez la commande suivante:

   ```
   runmqsc QM1
   START LISTENER TRPTYPE(TCP) PORT(1414)
   end
   ```

   L'espace adresse de l'initiateur de canal doit être démarré pour que vous puissiez démarrer un programme d'écoute sur z/OS.
5. Sur le système où le serveur mqweb qui héberge le messaging REST API est en cours d'exécution, créez ou mettez à jour un fichier CCDT JSON contenant les informations de connexion du gestionnaire de files d'attente.

   Le fichier CCDT doit inclure les informations `name`, `clientConnection`et `type` . Vous pouvez éventuellement inclure des informations supplémentaires, telles que des informations `transmissionSecurity` . Pour plus d'informations sur toutes les définitions d'attribut de canal CCDT, voir Liste complète des définitions d'attribut de canal CCDT.

   L'exemple suivant illustre un fichier CCDT JSON de base pour une connexion de gestionnaire de files d'attente éloignées. Il définit le nom du canal sur le même nom que l'exemple de canal de connexion serveur créé à l'étape «1», à la page 727. Le port de connexion est défini sur la même valeur que le port utilisé par le programme d'écoute. L'hôte de connexion est défini sur le nom d'hôte du système sur lequel le gestionnaire de files d'attente éloignées, QM1, est en cours d'exécution.

   ```
   {
     "channel": [{
         "name": "QM1.SVRCONN",
         "clientConnection": {
           "connection": [{
               "host": "example.com",
               "port": 1414
             }],
           "queueManager": "QM1"
         },
         "type": "clientConnection"
       }]
   }
   ```
6. A partir de l'installation qui exécute le serveur mqweb qui héberge le messaging REST API, utilisez la commande **setmqweb remote** pour ajouter les informations du gestionnaire de files d'attente éloignées à la configuration du serveur mqweb.

   Au minimum, vous devez spécifier les paramètres suivants:

   - **-qmgrName**, où vous spécifiez le nom du gestionnaire de files d'attente.
   - **-ccdtURL**, où vous spécifiez l'URL de la table de définition de canal du client pour le gestionnaire de files d'attente.

- **-uniqueName**, où vous spécifiez un nom unique pour le gestionnaire de files d'attente. Le nom unique est utilisé pour différencier les gestionnaires de files d'attente éloignées qui peuvent avoir le même nom et ne doit donc pas exister pour identifier un autre gestionnaire de files d'attente.

Vous pouvez spécifier plusieurs autres options, telles que le nom d'utilisateur et le mot de passe à utiliser pour la connexion du gestionnaire de files d'attente éloignées, ou les détails du magasin de clés de confiance et du magasin de clés. Pour obtenir la liste complète des paramètres pouvant être spécifiés avec la commande **setmqweb remote** , voir setmqweb remote.

Par exemple, pour ajouter le gestionnaire de files d'attente éloignées QM1, avec l'exemple de fichier CCDT, entrez la commande suivante:

```
setmqweb remote add -uniqueName "RemoteQM1" -qmgrName "QM1" -ccdtURL "c:\myccdts\ccdt.json"
```

Résultats

Le gestionnaire de files d'attente éloignées peut être utilisé avec messaging REST API en utilisant le nom unique dans l'URL de la ressource à la place du nom du gestionnaire de files d'attente.

Exemple

L'exemple suivant configure la connexion de gestionnaire de files d'attente éloignées pour un gestionnaire de files d'attente QM1. IBM MQ Console autorisé à administrer le gestionnaire de files d'attente en fonction de l'autorisation accordée à l'utilisateur exampleUser. Les données d'identification de cet utilisateur sont fournies à IBM MQ Console lorsque **setmqweb remote** est utilisé pour configurer la connexion du gestionnaire de files d'attente.

1. Sur le système où se trouve le gestionnaire de files d'attente éloignées QM1 , un canal de connexion serveur et un programme d'écoute sont créés. Le programme d'écoute est démarré et l'utilisateur `exampleUser` est autorisé à se connecter au gestionnaire de files d'attente et à accéder à une file d'attente utilisée pour la messagerie:

```
runmqsc QM1
#Define the server connection channel that will accept connections from the Console
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
# Define the listener to use for the connection from the Console
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
# Start the listener
START LISTENER(REMOTE.LISTENER)
end

#Set mq authorization for exampleUser to access the queue manager and a queue for messaging
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +setall +dsp
setmqaut -m QM1 -t queue -p exampleUser -n EXAMPLEQ +put +get +browse +inq
```

2. Sur le système où s'exécute le serveur mqweb, un fichier `QM1_ccdt.json` est créé avec les informations de connexion suivantes:

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "example.com",
            "port": 1414
          }],
        "queueManager": "QM1"
      },
      "type": "clientConnection"
    }]
}
```

3. Sur le système où le serveur mqweb est en cours d'exécution, les informations de connexion du gestionnaire de files d'attente QM1 sont ajoutées au serveur mqweb. Les données d'identification de `exampleUser` sont incluses dans les informations de connexion:

```
setmqweb remote add -uniqueName "MACHINEAQM1" -qmgrName "QM1" -ccdtURL
"c:\myccdts\QM1_ccdt.json" -username "exampleUser" -password "password"
```

4. messaging REST API peut se connecter au gestionnaire de files d'attente éloignées QM1 en utilisant le nom unique de la connexion de gestionnaire de files d'attente à la place du nom du gestionnaire de files d'attente dans l'URL de la ressource:

```
curl -k https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/MACHINEAQM1/queue/EXAMPLEQ/
message -X POST -u myuser:mypassword -H "ibm-mq-rest-csrf-token: value" -H "Content-Type:
text/plain;charset=utf-8" --data "Hello World!"
```

V 9.4.0 Configuration d'un groupe de gestionnaires de files d'attente à utiliser avec l'API REST de messagerie

Vous pouvez utiliser messaging REST API pour vous connecter à des groupes de gestionnaires de files d'attente pour la messagerie. Avant de vous connecter à un groupe de gestionnaires de files d'attente, vous devez définir la configuration du gestionnaire de files d'attente éloignées pour le groupe. Vous pouvez ensuite vous connecter au groupe de gestionnaires de files d'attente à l'aide du nom unique défini dans les informations de configuration.

Avant de commencer

- Vérifiez que vous avez configuré le serveur mqweb pour qu'il soit utilisé par administrative REST API, administrative REST API for MFT, messaging REST APIou IBM MQ Console.Pour plus d'informations sur la configuration du serveur mqweb avec un registre de base, voir Configuration de base pour le serveur mqweb.
- Si le serveur mqweb est déjà configuré, assurez-vous d'avoir ajouté les utilisateurs appropriés pour activer la messagerie à l'étape 5 de la section Configuration de base pour le serveur mqweb. Les utilisateurs du messaging REST API doivent être membres du rôle `MQWebUser` . Les rôles `MQWebAdmin` et `MQWebAdminRO` ne sont pas applicables pour le messaging REST API.
  - Si **mqRestMessagingAdoptWebUserContext** est défini sur `true` dans la configuration du serveur mqweb, les utilisateurs ayant le rôle `MQWebUser` doivent être autorisés à accéder aux files d'attente et aux rubriques utilisées pour la messagerie. Vous pouvez autoriser ces utilisateurs via OAM ou RACF.
  - Si **mqRestMessagingAdoptWebUserContext** est défini sur `false` dans la configuration du serveur mqweb, l'ID utilisateur qui démarre le serveur mqweb doit être autorisé à accéder aux files d'attente et aux rubriques utilisées pour la messagerie. Vous pouvez autoriser cet utilisateur via OAM ou RACF.
- Vérifiez que messaging REST API est configuré pour se connecter aux gestionnaires de files d'attente éloignées. Pour plus d'informations, voir Configuration du mode de connexion pour messaging REST API

Pourquoi et quand exécuter cette tâche

Un groupe de gestionnaires de files d'attente vous permet de connecter des applications à n'importe quel gestionnaire de files d'attente au sein du groupe. Le groupe est défini comme un ensemble de connexions dans une table de définition de canal du client (CCDT). Lorsque vous utilisez un appel MQCONN ou MQCONNX, vous référencez le groupe en ajoutant un astérisque au nom du gestionnaire de files d'attente. Avec messaging REST API, vous référencez le groupe à l'aide du nom unique associé au groupe de gestionnaires de files d'attente. Le nom unique est inclus dans l'URL de la ressource à la place du nom du gestionnaire de files d'attente. Pour plus d'informations sur les groupes de gestionnaires de files d'attente, voir «Groupes de gestionnaires de files d'attente dans CCDT», à la page 948.

Vous pouvez également configurer vos gestionnaires de files d'attente éloignées individuellement. Pour plus d'informations, voir «Configuration d'un gestionnaire de files d'attente éloignées à utiliser avec messaging REST API», à la page 726.

Procédure

1. Sur chacun des gestionnaires de files d'attente éloignées du groupe, créez un canal de connexion serveur pour autoriser les connexions distantes au gestionnaire de files d'attente. Vous pouvez utiliser

la commande **DEFINE CHANNEL** MQSC sur la ligne de commande pour créer des canaux de connexion serveur.
Par exemple, pour créer un canal de connexion serveur `QM1.SVRCONN` pour le gestionnaire de files d'attente QM1, entrez la commande suivante:

```
DEFINE CHANNEL(QM1.SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

Pour plus d'informations sur **DEFINE CHANNEL** et les options disponibles, voir DEFINE CHANNEL.

2. Sur chacun des gestionnaires de files d'attente éloignées du groupe, vérifiez qu'un utilisateur approprié est autorisé à accéder au gestionnaire de files d'attente. Cet utilisateur doit également être autorisé à accéder aux files d'attente ou aux rubriques que vous utilisez pour la messagerie. L'utilisateur a besoin des droits `connect`, `inquire`, `alternate user`et `set context` sur le gestionnaire de files d'attente. Sous UNIX, Linux, and Windows , utilisez la commande de contrôle **setmqaut** sur la ligne de commande. Sous z/OS, définissez des profils RACF pour accorder à l'utilisateur autorisé l'accès au gestionnaire de files d'attente.
Par exemple, sous UNIX, Linux, and Windows, entrez la commande suivante pour autoriser un utilisateur, `exampleUser`, à accéder au gestionnaire de files d'attente QM1:

```
setmqaut -m QM1 -t qmgr -p exampleUser +connect +inq +altusr +setall
```

Pour plus d'informations sur l'utilisateur devant être autorisé, voir «Détermination du principal de sécurité utilisé par messaging REST API», à la page 733.

3. ALW
S'il n'existe aucun programme d'écoute sur chacun des gestionnaires de files d'attente éloignées du groupe, créez des programmes d'écoute pour accepter les connexions réseau entrantes. Vous pouvez utiliser la commande **DEFINE LISTENER** MQSC sur la ligne de commande pour créer des programmes d'écoute.
Par exemple, pour créer un programme d'écoute `REMOTE.LISTENER` sur le port 1414 pour le gestionnaire de files d'attente éloignées QM1, entrez la commande suivante:

```
runmqsc QM1
DEFINE LISTENER(REMOTE.LISTENER) TRPTYPE(TCP) PORT(1414)
end
```

4. Sur chacun des gestionnaires de files d'attente éloignées du groupe, vérifiez que le programme d'écoute est en cours d'exécution à l'aide de la commande **START LISTENER** MQSC sur la ligne de commande.

ALW Par exemple, sous AIX, Linux, and Windows , pour démarrer le programme d'écoute `REMOTE.LISTENER` pour le gestionnaire de files d'attente QM1, entrez la commande suivante:

```
runmqsc QM1
START LISTENER(REMOTE.LISTENER)
end
```

z/OS Par exemple, sous z/OS, pour démarrer le programme d'écoute, entrez la commande suivante:

```
runmqsc QM1
START LISTENER TRPTYPE(TCP) PORT(1414)
end
```

L'espace adresse de l'initiateur de canal doit être démarré pour que vous puissiez démarrer un programme d'écoute sur z/OS.

5. Sur le système où le serveur mqweb qui héberge le messaging REST API est en cours d'exécution, créez un fichier CCDT JSON. Ce fichier JSON contient des informations de connexion pour chaque gestionnaire de files d'attente du groupe.

Le fichier CCDT doit inclure les informations `name`, `clientConnection`et `type` pour chaque connexion de gestionnaire de files d'attente. Vous pouvez éventuellement inclure des informations supplémentaires, telles que des informations `transmissionSecurity` . Pour plus d'informations sur

toutes les définitions d'attribut de canal CCDT, voir Liste complète des définitions d'attribut de canal CCDT.

L'exemple suivant illustre un fichier CCDT JSON de base pour deux connexions de gestionnaire de files d'attente. La première connexion concerne le gestionnaire de files d'attente QM1. Il dispose d'un canal de connexion serveur `QM1.SVRCONN`, d'un programme d'écoute sur le port 1414et s'exécute sur l'hôte `QM1.example.com`. La deuxième connexion concerne le gestionnaire de files d'attente QM2. Il dispose d'un canal de connexion serveur `QM2.SVRCONN`, d'un programme d'écoute sur le port 1415et s'exécute sur l'hôte `QM2.example.com`. Toutefois, comme les connexions font partie du groupe de gestionnaires de files d'attente QMGRP, la zone **queueManager** des deux connexions est définie sur le nom du groupe de gestionnaires de files d'attente.

```
{
  "channel": [{
      "name": "QM1.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM1.example.com",
            "port": 1414
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
   }],
   "channel": [{
      "name": "QM2.SVRCONN",
      "clientConnection": {
        "connection": [{
            "host": "QM2.example.com",
            "port": 1415
          }],
        "queueManager": "QMGRP"
      },
      "type": "clientConnection"
    }]
}
```

6. A partir de l'installation qui exécute le serveur mqweb qui héberge le messaging REST API, utilisez la commande **setmqweb remote** pour ajouter le groupe de gestionnaires de files d'attente à la configuration du serveur mqweb.

   Au minimum, vous devez spécifier les paramètres suivants:

   - **-qmgrName**, où vous spécifiez le nom de groupe pour le groupe de gestionnaires de files d'attente.
   - **-ccdtURL**, où vous spécifiez l'URL de la table de définition de canal du client pour les gestionnaires de files d'attente.
   - **-uniqueName**, où vous spécifiez un nom unique pour identifier le groupe de gestionnaires de files d'attente.
   - **-group**, pour définir les informations du gestionnaire de files d'attente comme étant pour un groupe.

   Vous pouvez spécifier plusieurs autres options, telles que le nom d'utilisateur et le mot de passe à utiliser pour la connexion, ou les détails du magasin de clés de confiance et du magasin de clés. Pour obtenir la liste complète des paramètres pouvant être spécifiés avec la commande **setmqweb remote** , voir setmqweb remote.

   Par exemple, pour ajouter le groupe de gestionnaires de files d'attente QMGRP, avec l'exemple de fichier CCDT, entrez la commande suivante:

```
setmqweb remote add  -uniqueName "MyQMGRP" -qmgrName "QMGRP" -ccdtURL
"c:\myccdts\group_ccdt.json" -group
```

Résultats

Le groupe de gestionnaires de files d'attente éloignées peut être utilisé avec messaging REST API en utilisant le nom unique dans l'URL de la ressource. Un gestionnaire de files d'attente du groupe est

sélectionné pour exécuter la demande et des informations sur le gestionnaire de files d'attente qui a exécuté la demande sont renvoyées dans l'en-tête de réponse `ibm-mq-resolved-qmgr`.

V 9.4.0 Détermination du principal de sécurité utilisé par messaging REST API

Lorsque vous utilisez messaging REST API, un utilisateur approprié doit être autorisé à accéder aux gestionnaires de files d'attente, aux files d'attente et aux rubriques auxquelles vous souhaitez vous connecter pour la messagerie. L'utilisateur qui doit être autorisé dépend de la manière dont votre serveur mqweb est configuré et de l'utilisation ou non des gestionnaires de files d'attente éloignées avec messaging REST API.

Par défaut, le principal de sécurité utilisé pour autoriser l'accès au gestionnaire de files d'attente est l'utilisateur qui démarre le serveur mqweb qui exécute messaging REST API. Le principal de sécurité utilisé pour autoriser l'accès aux files d'attente et aux rubriques est l'utilisateur qui est connecté à messaging REST API. Toutefois, votre connexion de serveur mqweb ou de gestionnaire de files d'attente éloignées peut être configurée de sorte qu'un autre principal de sécurité soit utilisé.

Détermination du principal de sécurité utilisé pour la connexion au gestionnaire de files d'attente

Pour les connexions de gestionnaire de files d'attente locales, le principal de sécurité utilisé pour la connexion au gestionnaire de files d'attente est l'utilisateur qui démarre le serveur mqweb qui exécute messaging REST API. Pour les connexions de gestionnaire de files d'attente éloignées, les principaux de sécurité suivants sont utilisés par messaging REST API pour autoriser l'accès au gestionnaire de files d'attente, par ordre de priorité. En d'autres termes, si les utilisateurs sont spécifiés de plusieurs manières dans la configuration du gestionnaire de files d'attente éloignées, le premier de la liste est utilisé pour l'autorisation.

1. Le principal de sécurité est un contexte utilisateur adopté à partir d'un exit de sécurité.
2. Le principal de sécurité est un contexte utilisateur adopté dans une règle CHLAUTH sur le canal de connexion serveur utilisé pour la connexion au gestionnaire de files d'attente éloignées.
3. Le principal de sécurité est l'ID utilisateur inclus dans la configuration du gestionnaire de files d'attente éloignées pour messaging REST API. Cet ID utilisateur est éventuellement inclus dans les informations de connexion du gestionnaire de files d'attente lorsque vous ajoutez le gestionnaire de files d'attente à l'aide de la commande **setmqweb remote** .
4. Le principal de sécurité est l'utilisateur qui démarre le serveur mqweb qui exécute le messaging REST API.

Pour plus d'informations sur la configuration des gestionnaires de files d'attente éloignées à utiliser avec messaging REST API, voir «Configuration d'un gestionnaire de files d'attente éloignées à utiliser avec messaging REST API», à la page 726.

Détermination du principal de sécurité utilisé pour la connexion aux files d'attente et aux rubriques

Vous pouvez définir une propriété dans la configuration du serveur mqweb pour déterminer le principal de sécurité utilisé pour autoriser les connexions aux files d'attente et aux rubriques lorsque vous utilisez messaging REST API. Cette propriété est la propriété **mqRestMessagingAdoptWebUserContext** . Vous pouvez afficher la valeur de cette propriété à l'aide de la commande **dspmqweb properties** .

- Si **mqRestMessagingAdoptWebUserContext** est défini sur true, messaging REST API utilise l'ID utilisateur de l'utilisateur connecté à messaging REST API pour l'autorisation. Par conséquent, l'ID utilisateur ou les ID utilisateur qui existent dans la configuration du serveur mqweb à utiliser avec messaging REST API sont les principaux de sécurité qui doivent être autorisés à accéder aux files d'attente et aux rubriques.
- Si **mqRestMessagingAdoptWebUserContext** est défini sur false, messaging REST API utilise l'ID utilisateur de l'utilisateur qui a démarré le serveur mqweb qui héberge messaging REST API pour

l'autorisation. Par conséquent, un ID utilisateur identique à l'ID utilisateur qui démarre le serveur mqweb qui héberge messaging REST API doit être autorisé à accéder aux files d'attente et aux rubriques.

Si vos files d'attente et rubriques se trouvent sur un gestionnaire de files d'attente éloignées, le principal de sécurité utilisé pour l'autorisation peut être déterminé par les paramètres de la configuration du gestionnaire de files d'attente. Les principaux de sécurité suivants peuvent être utilisés, par ordre de priorité:

1. Le principal de sécurité est un contexte utilisateur adopté à partir d'un exit de sécurité.
2. Le principal de sécurité est un contexte utilisateur adopté dans une règle CHLAUTH sur le canal de connexion serveur utilisé pour la connexion au gestionnaire de files d'attente éloignées. Par exemple, vous pouvez configurer une règle CHLAUTH sur le canal de connexion serveur pour utiliser le paramètre MCAUSER. Ensuite, toutes les connexions sont mappées à un ID utilisateur autorisé à utiliser le gestionnaire de files d'attente.
3. Le principal de sécurité est un contexte utilisateur adopté à partir des éléments AUTHINFO du gestionnaire de files d'attente. Si l'objet AUTHINFO référencé par l'attribut CONNAUTH du gestionnaire de files d'attente est configuré pour utiliser **ADOPTCTX(yes)**, le principal de sécurité utilisé pour autoriser les connexions au gestionnaire de files d'attente est également utilisé pour autoriser les files d'attente et les rubriques. Par exemple, ce principal de sécurité peut être l'ID utilisateur inclus dans les informations de connexion du gestionnaire de files d'attente éloignées dans le cadre de la commande **setmqweb remote** .

Information associée

CHLAUTH

CONNAUTH

propriétés de dspmqweb

Développement d'applications MQI avec IBM MQ

IBM MQ prend en charge C, Visual Basic, COBOL, Assembler, RPG, pTALet PL/I. Ces langages procéduraux utilisent l'interface de file d'attente de messages (MQI) pour accéder aux services de mise en file d'attente de messages.

Pour plus d'informations sur l'écriture de vos applications dans la langue de votre choix, voir les sous-rubriques.

Pour une présentation de l'interface d'appel pour les langages de procédure, voir Descriptions des appels. Cette rubrique contient une liste des appels MQI, et chaque appel vous montre comment coder les appels dans chacun de ces langages.

IBM MQ fournit des fichiers de définition de données pour vous aider à écrire vos applications. Pour une description complète, voir «Fichiers de définition de données IBM MQ», à la page 735.

Pour vous aider à choisir le langage de procédure dans lequel coder vos programmes, tenez compte de la longueur maximale des messages traités par vos programmes. Si vos programmes ne traitent que des messages d'une longueur maximale connue, vous pouvez les coder dans n'importe quelle langue prise en charge. Si vous ne connaissez pas la longueur maximale des messages que les programmes doivent traiter, la langue que vous choisissez varie selon que vous écrivez une application CICS, IMSou par lots:

IMS et traitement par lots
: Codez les programmes en langage C, PL/I ou assembleur afin d'utiliser les fonctions offertes par ces langages pour obtenir et libérer des quantités arbitraires de mémoire. Vous pouvez également coder vos programmes en COBOL, mais utiliser des sous-routines en langage assembleur, PL/I ou C pour obtenir et libérer du stockage.

CICS
: Codez les programmes dans n'importe quelle langue prise en charge par CICS. L'interface EXEC CICS fournit les appels pour la gestion de la mémoire, si nécessaire.

Concepts associés

«Applications orientées objet», à la page 16
IBM MQ prend en charge JMS, Java, C + + et .NET. Ces langages et infrastructures utilisent le modèle d'objet IBM MQ , qui fournit des classes fournissant les mêmes fonctionnalités que les appels et les structures IBM MQ .

Présentation technique

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

Référence associée

Références relatives au développement d'applications

Fichiers de définition de données IBM MQ

IBM MQ fournit des fichiers de définition de données pour vous aider à écrire vos applications.

Les fichiers de définition de données sont également appelés:

Langue	Définitions de données
C	Fichiers d'inclusion ou fichiers d'en-tête
Visual Basic	Fichiers de module (versions 32 bits uniquement)
COBOL	Copier des fichiers
Assembler	Macros
PL/I	Inclure les fichiers

Les fichiers de définition de données qui vous aident à écrire des exits de canal sont décrits dans IBM MQ COPY, header, include et module files.

Les fichiers de définition de données destinés à vous aider à écrire des exits de services installables sont décrits dans «Exits utilisateur, exits API et services optionnels d'IBM MQ», à la page 962.

Pour les fichiers de définition de données pris en charge sous C + +, voir Utilisation de C++.

IBM i

Pour les fichiers de définition de données pris en charge dans RPG, voir IBM i Application Programming Reference (ILE/RPG).

Les noms des fichiers de définition de données possèdent le préfixe CMQ et un suffixe déterminé par le langage de programmation:

Suffixe	Langue
a	Langage d'assemblage
b	Visual Basic
c	C
l	COBOL (sans valeurs initialisées)
p	PL/I
v	COBOL (avec les valeurs par défaut définies)

Bibliothèque d'installation

z/OS Le nom **thlqual** est le qualificatif de haut niveau de la bibliothèque d'installation sous z/OS.

Cette rubrique présente les fichiers de définition de données IBM MQ , sous les en-têtes suivants:

- «Fichiers d'inclusion du langage C», à la page 736
- «Fichiers du module Visual Basic», à la page 736
- «Fichiers de copie COBOL», à la page 736
- z/OS «Macros de langage assembleur System/390», à la page 738
- z/OS «Fichiers d'inclusion PL/I», à la page 738

Fichiers d'inclusion du langage C

Les fichiers d'inclusion IBM MQ C sont répertoriés dans Fichiers d'en-tête C. Ils sont installés dans les répertoires ou les bibliothèques suivants:

Plateforme	Répertoire d'installation ou bibliothèque
IBM i IBM i	QMQM/H
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS z/OS	**thlqual**.SCSQC370

où *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Remarque : Pour AIX and Linux, les fichiers d'inclusion sont liés symboliquement à `/usr/include`.

Pour plus d'informations sur la structure des répertoires, voir Planification de la prise en charge du système de fichiers.

Fichiers du module Visual Basic

IBM MQ for Windows fournit quatre fichiers de module Visual Basic.

Ils sont répertoriés dans les fichiers de module Visual Basic et installés dans

```
MQ_INSTALLATION_PATH\Tools\Samples\VB\Include
```

Fichiers de copie COBOL

Pour COBOL, IBM MQ fournit des fichiers de copie distincts contenant les constantes nommées et deux fichiers de copie pour chacune des structures.

Il existe deux fichiers de copie pour chaque structure car chacun est fourni avec et sans valeurs initiales:

- Dans la SECTION WORKING-STORAGE d'un programme COBOL, utilisez les fichiers qui initialisent les zones de structure avec les valeurs par défaut. Ces structures sont définies dans les fichiers de copie dont les noms sont suffixés avec la lettre V (valeurs).
- Dans la section LINKAGE SECTION d'un programme COBOL, utilisez les structures sans valeurs initiales. Ces structures sont définies dans les fichiers de copie dont les noms sont suffixés avec la lettre L (liaison).

IBM i Des fichiers de copie contenant des données et des définitions d'interface pour IBM i sont fournis pour les programmes ILE COBOL utilisant des appels prototypés à l'interface MQI. Les fichiers

existent dans QMQM/QCBLLESRC avec des noms de membre dont le suffixe est L (pour les structures sans valeurs initiales) ou V (pour les structures avec valeurs initiales).

Les fichiers de copie COBOL IBM MQ sont répertoriés dans Fichiers COBOL COPY. Ils sont installés dans les répertoires suivants:

Plateforme	Répertoire d'installation ou bibliothèque
Linux AIX AIX and Linux	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (pour Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (pour IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

N'incluez dans votre programme que les fichiers dont vous avez besoin. Effectuez cette opération avec une ou plusieurs instructions COPY après une déclaration level-01 . Cela signifie que vous pouvez inclure plusieurs versions des structures dans un programme si nécessaire. Notez que CMQV est un fichier volumineux.

Voici un exemple de code COBOL pour inclure le fichier de copie CMQMDV:

```
01 MQM-MESSAGE-DESCRIPTOR.
COPY CMQMDV.
```

Chaque déclaration de structure commence par un élément level-01 ; vous pouvez déclarer plusieurs instances de la structure en codant la déclaration level-01 suivie d'une instruction COPY à copier dans le reste de la déclaration de structure. Pour faire référence à l'instance appropriée, utilisez le mot clé IN.

Voici un exemple de code COBOL pour inclure deux instances 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.
```

Alignez les structures sur des limites de 4 octets. Si vous utilisez l'instruction COPY pour inclure une structure après un élément qui n'est pas l'élément level-01 , assurez-vous que la structure est un multiple de 4 octets à partir du début de l'élément level-01 . Si vous ne le faites pas, vous risquez de réduire les performances de votre application.

Les structures sont décrites dans Types de données utilisés dans l'interface MQI. Les descriptions des zones dans les structures affichent les noms des zones sans préfixe. Dans les programmes COBOL, préfixez les noms de zone avec le nom de la structure suivi d'un trait d'union, comme indiqué dans les déclarations COBOL. Les zones des fichiers de copie de structure sont ainsi préfixées.

Les noms de zone dans les déclarations des fichiers de copie de structure sont en majuscules. Vous pouvez utiliser une casse mixte ou des minuscules à la place. Par exemple, la zone *StrucId* de la structure MQGMO s'affiche sous la forme MQGMO-STRUCID dans la déclaration COBOL et dans le fichier de copie.

Les structures de suffixe V étant déclarées avec des valeurs initiales pour toutes les zones, vous devez définir uniquement les zones dans lesquelles la valeur requise est différente de la valeur initiale.

Macros de langage assembleur System/390

z/OS

IBM MQ for z/OS fournit deux macros de langage assembleur contenant les constantes nommées et une macro pour générer chaque structure.

Ils sont répertoriés dans z/OS Assembler COPY files et installés dans **thlqual**.SCSQMACS.

Ces macros sont appelées à l'aide d'un code tel que celui-ci:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

Fichiers d'inclusion PL/I

z/OS

IBM MQ for z/OS fournit des fichiers d'inclusion qui contiennent toutes les définitions dont vous avez besoin lorsque vous écrivez des applications IBM MQ dans PL/I.

Les fichiers sont répertoriés dans Fichiers d'inclusion PL/I et installés dans le répertoire **thlqual**.SCSQPLIC:

Incluez ces fichiers dans votre programme si vous souhaitez lier le stub IBM MQ à votre programme (voir «Preparing your program to run», à la page 1050). Incluez uniquement CMQP si vous avez l'intention de lier les appels IBM MQ de manière dynamique (voir «Dynamically calling the IBM MQ stub», à la page 1057). La liaison dynamique ne peut être effectuée que pour les programmes par lots et IMS .

Ecriture d'une application de procédure pour la mise en file d'attente

La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

Utilisez les liens suivants pour en savoir plus sur l'écriture d'applications :

- «Présentation de l'interface de file d'attente de messages», à la page 739
- «Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
- «Ouverture et fermeture d'objets», à la page 760
- «Insertion de messages dans une file d'attente», à la page 772
- «Obtention de messages à partir d'une file d'attente», à la page 787
- «Ecriture d'applications de publication / abonnement», à la page 829
- «Interrogation et définition des attributs d'objet», à la page 873
- «Validation et annulation d'unités de travail», à la page 876
- «Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
- «Utilisation de l'interface MQI et des clusters», à la page 909
- z/OS «Using and writing applications on IBM MQ for z/OS», à la page 914
- z/OS «IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72

Concepts associés

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Développement d'applications pour IBM MQ», à la page 5

Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

«Ecriture d'applications de procédure client», à la page 937
Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

«Création d'une application procédurale», à la page 1027
Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

«Traitement des erreurs de programme de procédure», à la page 1065
Ces informations expliquent les erreurs associées aux appels MQI de vos applications lors de l'exécution d'un appel ou de la distribution de son message à sa destination finale.

Tâches associées

«Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085
Ces exemples de programmes sont écrits dans des langages procéduraux et présentent des utilisations typiques de l'interface MQI (Message Queue Interface). Programmes IBM MQ sur différentes plateformes.

Présentation de l'interface de file d'attente de messages

Découvrez les composants MQI (Message Queue Interface).

L'interface de file d'attente de messages comprend les éléments suivants:

- *Appels* via lesquels les programmes peuvent accéder au gestionnaire de files d'attente et à ses fonctions
- *Structures* utilisées par les programmes pour transmettre des données au gestionnaire de files d'attente et en extraire des données
- *Types de données élémentaires* pour la transmission de données au gestionnaire de files d'attente et l'extraction de données à partir de ce dernier

z/OS IBM MQ for z/OS fournit également:

- Deux appels supplémentaires par lesquels les programmes de traitement par lots z/OS peuvent valider et rétablir les modifications.
- *Fichiers de définition de données* (parfois appelés fichiers de copie, macros, fichiers d'inclusion et fichiers d'en-tête) qui définissent les valeurs des constantes fournies avec IBM MQ for z/OS.
- *Programmes de remplacement* pour l'édition de liens vers vos applications.
- Suite d'exemples de programmes qui montrent comment utiliser l'interface MQI sur la plateforme z/OS . Pour plus d'informations sur ces exemples, voir «Using the sample programs for z/OS», à la page 1195.

IBM i IBM MQ for IBM i fournit également:

- *Fichiers de définition de données* (parfois appelés fichiers de copie, macros, fichiers d'inclusion et fichiers d'en-tête) qui définissent les valeurs des constantes fournies avec IBM MQ for IBM i.
- Trois programmes de remplacement pour l'édition de liens vers vos applications ILE C, ILE COBOL et ILE RPG.
- Suite d'exemples de programmes qui montrent comment utiliser l'interface MQI sur la plateforme IBM i .

Les systèmes AIX, Linux, and Windows fournissent également:

- Appels via lesquels les programmes système IBM MQ for AIX, Linux, and Windows peuvent valider et revenir sur les modifications.

- *Inclure les fichiers* qui définissent les valeurs des constantes fournies sur ces plateformes.
- *Fichiers de bibliothèque* pour lier vos applications.
- Suite d'exemples de programmes qui montrent comment utiliser l'interface MQI sur ces plateformes. Pour plus d'informations sur ces exemples, voir «Utilisation des exemples de programme sur Multiplatforms», à la page 1086.
- Exemple de code source et exécutable pour les liaisons à des gestionnaires de transactions externes.

Utilisez les liens suivants pour en savoir plus sur l'interface MQI:

- «Appels MQI», à la page 740
- «Appels de point de synchronisation», à la page 741
- «Conversion de données, types de données, définitions de données et structures», à la page 742
- «Programmes de remplacement IBM MQ et fichiers de bibliothèque», à la page 743
- «Paramètres communs à tous les appels», à la page 748
- «Spécification de mémoires tampon», à la page 749
- z/OS «z/OS batch considerations», à la page 749
- «Traitement des signaux AIX and Linux», à la page 750

Concepts associés

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Appels MQI

Utilisez ces informations pour en savoir plus sur les appels dans l'interface MQI (Message Queue Interface).

Les appels dans l'interface MQI peuvent être regroupés comme suit:

MQCONN, MQCONNX et MQDISC
Utilisez ces appels pour connecter un programme à (avec ou sans options) et déconnecter un programme d'un gestionnaire de files d'attente.

z/OS Si vous écrivez des programmes CICS pour z/OS, vous n'avez pas besoin d'utiliser ces appels. Toutefois, il est recommandé de les utiliser si vous souhaitez porter votre application sur d'autres plateformes.

MQOPEN et MQCLOSE
Utilisez ces appels pour ouvrir et fermer un objet, tel qu'une file d'attente.

MQPUT et MQPUT1
Utilisez ces appels pour placer un message dans une file d'attente.

MQGET
Utilisez cet appel pour parcourir les messages d'une file d'attente ou pour supprimer des messages d'une file d'attente.

MQSUB, MQSUBRQ
Utilisez ces appels pour enregistrer un abonnement à une rubrique et pour demander des publications correspondant à l'abonnement.

MQINQ
Utilisez cet appel pour vous renseigner sur les attributs d'un objet.

MQSET
Utilisez cet appel pour définir certains des attributs d'une file d'attente. Vous ne pouvez pas définir les attributs d'autres types d'objet.

MQBEGIN, MQCMIT et MQBACK
Utilisez ces appels lorsque IBM MQ est le coordinateur d'une unité de travail. MQBEGIN démarre l'unité de travail. MQCMIT et MQBACK arrêtent l'unité d'oeuvre, en validant ou en annulant les mises à jour effectuées au cours de l'unité d'oeuvre. IBM i Le contrôleur de validation IBM i est utilisé pour coordonner les unités d'oeuvre globales sur IBM MQ for IBM i. Les commandes de contrôle de validation de démarrage natif, de validation et d'invalidation sont utilisées.

MQCRTMH, MQBUFMH, MQMHBUF, MQDLTMH
Utilisez ces appels pour créer un descripteur de message, pour convertir un descripteur de message en mémoire tampon ou en mémoire tampon en descripteur de message et pour supprimer un descripteur de message.

MQSETMP, MQINQMP, MQDLTMP
Utilisez ces appels pour définir une propriété de message sur un descripteur de message, interroger une propriété de message et supprimer une propriété d'un descripteur de message.

MQCB, MQCB_FUNCTION, MQCTL
Utilisez ces appels pour enregistrer et contrôler une fonction de rappel.

MQSTAT
Utilisez cet appel pour extraire des informations de statut sur les opérations d'insertion asynchrone précédentes.

Voir Descriptions des appels pour une description des appels MQI.

Appels de point de synchronisation
Utilisez ces informations pour en savoir plus sur les appels de point de synchronisation sur différentes plateformes.

Les appels de point de synchronisation sont disponibles comme suit:

IBM MQ for z/OSappels
z/OS

IBM MQ for z/OS fournit les appels MQCMIT et MQBACK.

Utilisez ces appels dans les programmes batch z/OS pour indiquer au gestionnaire de files d'attente que toutes les opérations MQGET et MQPUT effectuées depuis le dernier point de synchronisation doivent être rendues permanentes (validées) ou annulées. Pour valider et retirer des modifications dans d'autres environnements:

CICS
Utilisez des commandes telles que EXEC CICS SYNCPOINT et EXEC CICS SYNCPOINT ROLLBACK.

IMS
Utilisez les fonctions de point de synchronisation IMS , telles que GU (get unique) pour les appels IOPCB, CHKP (checkpoint) et ROLB (rollback).

RRS
Utilisez MQCMIT et MQBACK ou SRRCMIT et SRRBACK selon les besoins. (Voir «Transaction management and recoverable resource manager services», à la page 881.)

Remarque : SRRCMIT et SRRBACK sont des commandes RRS natives, ce ne sont pas des appels MQI.

IBM iappels

IBM i

IBM MQ for IBM i fournit les commandes MQCMIT et MQBACK. Vous pouvez également utiliser les commandes IBM i COMMIT et ROLLBACK, ou toute autre commande ou appel qui lance les fonctions de contrôle de validation IBM i (par exemple, EXEC CICS SYNCPOINT).

Appels IBM MQ sur les plateformes AIX, Linux, and Windows

ALW

IBM MQ for AIX, Linux, and Windows fournissent les appels MQCMIT et MQBACK.

Utilisez les appels de point de synchronisation dans les programmes pour indiquer au gestionnaire de files d'attente que toutes les opérations MQGET et MQPUT effectuées depuis le dernier point de synchronisation doivent être rendues permanentes (validées) ou annulées. Pour valider et annuler des modifications dans l'environnement CICS , utilisez des commandes telles que EXEC CICS SYNCPOINT et EXEC CICS SYNCPOINT ROLLBACK.

Conversion de données, types de données, définitions de données et structures

Utilisez ces informations pour en savoir plus sur les conversions de données, les types de données élémentaires, les définitions de données IBM MQ et les structures lors de l'utilisation de l'interface de file d'attente de messages.

Conversion de données

L'appel MQXCNVC (caractères de conversion) convertit les données de caractères de message d'un jeu de caractères à un autre. Cet appel est utilisé uniquement à partir d'un exit de conversion de données, sauf sur IBM MQ for z/OS.

Voir MQXCNVC-Conversion de caractères pour la syntaxe utilisée avec l'appel MQXCNVC et «Ecriture des exits de conversion de données», à la page 1010 pour des instructions sur l'écriture et l'appel des exits de conversion de données.

Types de données élémentaires

Pour les langages de programmation pris en charge, l'interface MQI fournit des types de données élémentaires ou des champs non structurés.

Ces types de données sont décrits en détail dans Types de données élémentaires.

IBM MQ définitions de données

z/OS IBM MQ for z/OS fournit des définitions de données sous la forme de fichiers de copie COBOL, de macros de langage d'assemblage, d'un fichier d'inclusion PL/I unique, d'un fichier d'inclusion de langage C unique et de fichiers d'inclusion de langage C + +.

IBM i IBM MQ for IBM i fournit des définitions de données sous la forme de fichiers de copie COBOL, de fichiers de copie RPG, de fichiers d'inclusion en langage C et de fichiers d'inclusion en langage C + +.

Les fichiers de définition de données fournis avec IBM MQ contiennent:

- Définitions de toutes les constantes et codes retour IBM MQ
- Définitions des structures et types de données IBM MQ
- Définitions de constantes pour l'initialisation des structures
- Prototypes de fonction pour chacun des appels (pour PL/I et le langage C uniquement)

Pour une description complète des fichiers de définition de données IBM MQ , voir «Fichiers de définition de données IBM MQ», à la page 735.

Structures

Les structures, utilisées avec les appels MQI répertoriés dans «Appels MQI», à la page 740, sont fournies dans des fichiers de définition de données pour chacun des langages de programmation pris en charge.

Voir Types de données de structure pour un récapitulatif des structures.

IBM i z/OS IBM MQ for z/OS et IBM MQ for IBM i fournissent des fichiers qui contiennent des constantes que vous pouvez utiliser lorsque vous complétez certaines zones de ces structures. Pour plus d'informations sur ces définitions de données, voir Définitions de donnéesIBM MQ.

Programmes de remplacement IBM MQ et fichiers de bibliothèque

Les programmes de remplacement et les fichiers de bibliothèque fournis sont répertoriés ici, pour chaque plateforme.

Pour plus d'informations sur l'utilisation des programmes de remplacement et des fichiers de bibliothèque lors de la génération d'une application exécutable, voir «Création d'une application procédurale», à la page 1027. Pour plus d'informations sur la liaison à des fichiers de bibliothèque C + +, voir Utilisation de C++ *IBM MQ Utilisation de C++*.

AIX *IBM MQ for AIX fichiers de bibliothèque*

Sous IBM MQ for AIX, vous devez lier votre programme aux fichiers de bibliothèque MQI fournis pour l'environnement dans lequel vous exécutez votre application, en plus de ceux fournis par le système d'exploitation.

Dans une application sans unités d'exécution, créez un lien vers l'une des bibliothèques suivantes:

Tableau 106. Fichiers de bibliothèque pour les applications AIX sans unités d'exécution

Fichier de bibliothèque	**Environnement**
libmqm.a	Serveur pour C
libmqic.a et libmqm.a	Client pour C
libmqmzf.a	Exits de service installables pour C
libmqmxa.a	Interface XA du serveur
libmqmxa64.a	Interface XA de remplacement de serveur
libmqcxa.a	Interface XA client
libmqcxa64.a	Interface XA de remplacement client
libmqmcbrt.o	Bibliothèque d'exécution IBM MQ pour la prise en charge de Micro Focus COBOL
libmqmcb.a	Serveur pour COBOL

Tableau 106. Fichiers de bibliothèque pour les applications AIX sans unités d'exécution (suite)	
Fichier de bibliothèque	**Environnement**
libmqicb.a	Client pour COBOL
libimqc23ia.a	Client for C++ (XLC 16)
libimqs23ia.a	Serveur pour C++ (XLC 16)
V 9.4.0 libimqc23ca.a	Client pour C++ (XLC 17)
V 9.4.0 libimqs23ca.a	Serveur pour C++ (XLC 17)

V 9.4.0 Les bibliothèques contenant "ia" ont été générées avec le compilateur XLC 16, tandis que les bibliothèques dont le nom contient "ca" ont été générées avec le compilateur XLC 17.

Dans une application à unités d'exécution, créez un lien vers l'une des bibliothèques suivantes:

Tableau 107. Fichiers de bibliothèque pour les applications AIX à unités d'exécution. Tableau à deux colonnes répertoriant les fichiers de bibliothèque et l'environnement de chaque fichier de bibliothèque.	
Fichier de bibliothèque	**Environnement**
libmqm_r.a	Serveur pour C
libmqic_r.a et libmqm_r.a	Client pour C
libmqmzf_r.a	Exits de service installables pour C
libmqmxa_r.a	Interface XA du serveur
libmqmxa64_r.a	Interface XA de remplacement de serveur
libmqcxa_r.a	Interface XA client
libmqcxa64_r.a	Interface XA de remplacement client
libimqc23ia_r.a	Client for C++ (XLC 16)
libimqs23ia_r.a	Serveur pour C++ (XLC 16)
V 9.4.0 libimqc23ca_r.a	Client pour C++ (XLC 17)
V 9.4.0 libimqs23ca_r.a	Serveur pour C++ (XLC 17)

V 9.4.0 Les bibliothèques dont les noms incluent ia ont été générées avec le compilateur XLC 16, tandis que les bibliothèques dont les noms incluent ca ont été générées avec le compilateur XLC 17.

Remarque : Vous ne pouvez pas créer de lien vers plusieurs bibliothèques. En d'autres termes, vous ne pouvez pas lier à la fois une bibliothèque à unités d'exécution et une bibliothèque non à unités d'exécution.

IBM i *IBM MQ for IBM i fichiers de bibliothèque*

Dans IBM MQ for IBM i, liez votre programme aux fichiers de bibliothèque MQI fournis pour l'environnement dans lequel vous exécutez votre application, en plus de ceux fournis par le système d'exploitation.

Pour les applications sans unités d'exécution:

Tableau 108. Fichiers de bibliothèque pour les applications IBM i sans unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm	Programme de service serveur et client
libmqic	Programme de service client
IMQB23I4	Programme de service de base C++
IMQS23I4	Programme de service du serveur C++
libmqmzf	Exits installables pour C

Dans une application à unités d'exécution:

Tableau 109. Fichiers de bibliothèque pour les applications IBM i à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm_r	Programme de service serveur et client
IMQB23I4_R	Programme de service de base C++
IMQS23I4_R	Programme de service du serveur C++
libmqmzf_r	Exits installables pour C
libmqic_r	Programme de service client

Sous IBM MQ for IBM i, vous pouvez écrire vos applications en C + +. Pour savoir comment lier vos applications C++ et pour plus de détails sur tous les aspects de l'utilisation de C + +, voir Utilisation de C++.

Linux *Fichiers de bibliothèque IBM MQ for Linux*

Sous IBM MQ for Linux, vous devez lier votre programme aux fichiers de bibliothèque MQI fournis pour l'environnement dans lequel vous exécutez votre application, en plus de ceux fournis par le système d'exploitation.

Dans une application sans unités d'exécution, créez un lien vers l'une des bibliothèques suivantes:

Tableau 110. Fichiers de bibliothèque pour les applications Linux sans unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm.so	Serveur pour C
libmqic.so et libmqm.so	Client pour C
libmqmzf.so	Exits de service installables pour C
libmqmxa.so	Interface XA du serveur
libmqmxa64.so	Interface XA de remplacement de serveur
libmqcxa.so	Interface XA client
libmqcxa64.so	Interface XA de remplacement client
libimqc23gl.so	Client pour C++
libimqs23gl.so	Serveur pour C++

Dans une application à unités d'exécution, créez un lien vers l'une des bibliothèques suivantes:

Tableau 111. Fichiers de bibliothèque pour les applications Linux à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm_r.so	Serveur pour C
libmqic_r.so et libmqm_r.so	Client pour C
libmqmzf_r.so	Exits de service installables pour C
libmqmxa_r.so	Interface XA du serveur
libmqmxa64_r.so	Interface XA de remplacement de serveur
libmqcxa_r.so	Interface XA client
libmqcxa64_r.so	Interface XA de remplacement client
libimqc23gl_r.so	Client pour C++
libimqs23gl_r.so	Serveur pour C++

Remarque : Vous ne pouvez pas créer de lien vers plusieurs bibliothèques. En d'autres termes, vous ne pouvez pas lier à la fois une bibliothèque à unités d'exécution et une bibliothèque non à unités d'exécution.

Windows *IBM MQ for Windows fichiers de bibliothèque*

Sous IBM MQ for Windows, vous devez lier votre programme aux fichiers de bibliothèque MQI fournis pour l'environnement dans lequel vous exécutez votre application, en plus de ceux fournis par le système d'exploitation:

Tableau 112. Fichiers de bibliothèque pour les applications Windows	
Fichier de bibliothèque	**Environnement**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Server for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Client for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Server XA interface for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Interface XA client pour C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Client MTS for C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	Prise en charge du serveur TXSeries CICS pour C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	Prise en charge du client TXSeries CICS pour C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Exits des services installables pour C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Server for IBM COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Server for Micro Focus COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Client pour IBM COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Client for Micro Focus COBOL (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Server for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Client for C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Base pour C++ (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Client MTS for C++ (32 bits)

Tableau 112. Fichiers de bibliothèque pour les applications Windows (suite)	
Fichier de bibliothèque	**Environnement**
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Server for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Client for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmxa.lib	Server XA interface for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqcxa.lib	Interface XA client pour C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Client MTS for C (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Server for IBM COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb.lib	Server for Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb.lib	Client for IBM COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Client for Micro Focus COBOL (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Server for C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Client for C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Base pour C++ (64 bits)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Client MTS for C++ (64 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Utilisez `amqmdnet.dll` pour compiler les programmes .NET . Pour plus d'informations, voir «Compilation de programmes IBM MQ .NET», à la page 626 dans la section «Développement d'applications .NET», à la page 567 .

Ces fichiers sont fournis à des fins de compatibilité avec les versions précédentes:

```
mqic32.lib
mqic32xa.lib
```

z/OS *IBM MQ for z/OS stub programs*

Before you can run a program written with IBM MQ for z/OS, you must link-edit it to the stub program supplied with IBM MQ for z/OS for the environment in which you are running the application.

The stub program provides the first stage of the processing of your calls into requests that IBM MQ for z/OS can process.

IBM MQ for z/OS supplies the following stub programs:

CSQBSTUB
Stub program for z/OS batch programs

CSQBRRSI
Stub program for z/OS batch programs using RRS by way of the MQI

CSQBRSTB
Stub program for z/OS batch programs using RRS directly

CSQCSTUB
Stub program for CICS programs

CSQQSTUB
Stub program for IMS programs

CSQXSTUB
Stub program for distributed queuing non-CICS exits

CSQASTUB
Stub program for data-conversion exits

Attention: If you use a stub program other than one listed for a specific environment, it might have unpredictable results.

Note: If you use the CSQBRSTB stub program, link-edit with ATRSCSS from SYS1.CSSLIB. (SYS1.CSSLIB is also known as the *Callable Services Library*). For more information about RRS see "Transaction management and recoverable resource manager services" on page 881.

Alternatively, you can dynamically call the stub from within your program. This technique is described in "Dynamically calling the IBM MQ stub" on page 1057.

In IMS, you might also need to use a special language interface module that is supplied by IBM MQ.

Do not run applications that are link-edited with CSQBSTUB and CSQQSTUB in the same IMS MPP region. This can cause problems such as DFS3607I or CSQQ005E messages. The first MQCONN call in an address space determines which interface is used, therefore CSQQSTUB and CSQBSTUB transactions must run in different IMS message regions.

Paramètres communs à tous les appels

Il existe deux types de paramètre communs à tous les appels: les descripteurs et les codes retour.

Utilisation de descripteurs

Tous les appels MQI utilisent un ou plusieurs *descripteurs*. Ils identifient le gestionnaire de files d'attente, la file d'attente ou tout autre objet, message ou abonnement, selon les besoins de l'appel.

Pour qu'un programme puisse communiquer avec un gestionnaire de files d'attente, il doit posséder un identificateur unique permettant de le connaître. Cet identificateur est appelé *descripteur de connexion*, parfois appelé *Hconn*. Pour les programmes CICS , le descripteur de connexion est toujours égal à zéro. Pour toutes les autres plateformes ou tous les autres styles de programmes, le descripteur de connexion est renvoyé par l'appel MQCONN ou MQCONNX lorsque le programme se connecte au gestionnaire de files d'attente. Les programmes transmettent le descripteur de connexion en tant que paramètre d'entrée lorsqu'ils utilisent les autres appels.

Pour qu'un programme puisse fonctionner avec un objet IBM MQ , il doit avoir un identificateur unique par lequel il connaît cet objet. Cet identificateur est appelé *descripteur d'objet*, parfois appelé *Hobj*. Le descripteur est renvoyé par l'appel MQOPEN lorsque le programme ouvre l'objet pour l'utiliser. Les programmes transmettent le descripteur d'objet en tant que paramètre d'entrée lorsqu'ils utilisent des appels MQPUT, MQGET, MQINQ, MQSET ou MQCLOSE ultérieurs.

De même, l'appel MQSUB renvoie un *descripteur d'abonnement* ou *Hsub*, qui est utilisé pour identifier l'abonnement dans les appels MQGET, MQCB ou MQSUBRQ suivants, et certains appels qui traitent les propriétés de message utilisent un *descripteur de message* ou *Hmsg*.

Description des codes retour

Un code achèvement et un code raison sont renvoyés en tant que paramètres de sortie par chaque appel. Ces codes sont connus collectivement sous le nom de *codes retour*.

Pour indiquer si un appel aboutit, chaque appel renvoie un *code achèvement* lorsque l'appel est terminé. Le code achèvement est généralement soit MQCC_OK indiquant la réussite, soit MQCC_FAILED indiquant l'échec. Certains appels peuvent renvoyer un état intermédiaire, MQCC_WARNING, indiquant une réussite partielle.

Chaque appel renvoie également un *code anomalie* qui indique la raison de l'échec ou de la réussite partielle de l'appel. Il existe de nombreux codes raison, couvrant des circonstances telles qu'une file d'attente saturée, des opérations d'extraction non autorisées pour une file d'attente et une file d'attente particulière non définie pour le gestionnaire de files d'attente. Les programmes peuvent utiliser le code

raison pour décider comment procéder. Par exemple, ils peuvent inviter les utilisateurs à modifier leurs données d'entrée, puis à renouveler l'appel, ou ils peuvent renvoyer un message d'erreur à l'utilisateur.

Lorsque le code achèvement est MQCC_OK, le code anomalie est toujours MQRC_NONE.

Les codes achèvement et raison de chaque appel sont répertoriés avec la description de cet appel. Voir Descriptions des appels et sélectionnez l'appel approprié dans la liste.

Pour des informations plus détaillées, y compris des idées de mesures correctives, voir:

- z/OS Messages, codes achèvement et codes anomalie IBM MQ for z/OS pour IBM MQ for z/OS
- Messages et codes anomalie pour toutes les autres plateformes IBM MQ

Spécification de mémoires tampon

Le gestionnaire de files d'attente fait référence aux mémoires tampon uniquement si elles sont requises. Si vous n'avez pas besoin d'une mémoire tampon lors d'un appel ou si la mémoire tampon a une longueur de zéro, vous pouvez utiliser un pointeur NULL vers une mémoire tampon.

Utilisez toujours datalength lorsque vous spécifiez la taille de la mémoire tampon dont vous avez besoin.

Lorsque vous utilisez une mémoire tampon pour conserver la sortie d'un appel (par exemple, pour conserver les données de message d'un appel MQGET ou les valeurs des attributs demandés par l'appel MQINQ), le gestionnaire de files d'attente tente de renvoyer un code anomalie si la mémoire tampon que vous spécifiez n'est pas valide ou se trouve dans un stockage en lecture seule. Toutefois, il se peut qu'il ne soit pas toujours en mesure de renvoyer un code anomalie.

z/OS *z/OS batch considerations*

z/OS batch programs that call the MQI can be in either supervisor or problem state.

However, they must meet the following conditions:

- They must be in task mode, not service request block (SRB) mode.
- They must be in Primary address space control (ASC) mode (not Access Register ASC mode).
- They must not be in cross-memory mode. The primary address space number (ASN) must be equal to the secondary ASN and the home ASN.
- They must not be used as MPF exit programs.
- No z/OS locks can be held.
- There can be no function recovery routines (FRRs) on the FRR stack.
- Any program status word (PSW) key can be in force for the MQCONN or MQCONNX call (provided the key is compatible with using storage that is in the TCB key), but subsequent calls that use the connection handle returned by MQCONN or MQCONNX:
  - Must have the same PSW key that was used on the MQCONN or MQCONNX call
  - Must have parameters accessible (for write, where appropriate) under the same PSW key
  - Must be issued under the same task (TCB), but not in any subtask of the task
- They can be in either 24-bit or 31-bit addressing mode. However, if 24-bit addressing mode is in force, parameter addresses must be interpreted as valid 31-bit addresses.

If any of these conditions is not met, a program check might occur. In some cases the call will fail and a reason code will be returned.

Linux AIX *Remarques sur AIX and Linux*

Remarques à prendre en compte lors du développement d'applications AIX and Linux .

Linux AIX *L'appel système fork dans les systèmes AIX and Linux*

Tenez compte des remarques suivantes lorsque vous utilisez un appel système fork dans des applications IBM MQ .

Si votre application souhaite utiliser `fork`, le processus parent de cette application doit appeler `fork` avant d'effectuer des appels IBM MQ , par exemple `MQCONN`, ou créer un objet IBM MQ à l'aide de **`ImqQueueManager`**.

Si votre application souhaite créer un processus enfant après avoir effectué des appels IBM MQ , le code de l'application doit utiliser un `fork()` avec `exec()` pour s'assurer que l'enfant est une nouvelle instance et non une copie exacte du parent.

Si votre application n'utilise pas `exec()`, l'appel d'API IBM MQ effectué dans le processus enfant renvoie MQRC_ENVIRONMENT_ERROR.

Linux AIX *Traitement des signaux AIX and Linux*

En général, les systèmes AIX and Linux sont passés d'un environnement sans unités d'exécution (processus) à un environnement à unités d'exécution multiples. Dans de nombreux cas, les signaux et le traitement des signaux, bien que pris en charge, ne s'intègrent pas bien dans l'environnement à unités d'exécution multiples et diverses restrictions existent.

En général, les systèmes AIX and Linux sont passés d'un environnement sans unités d'exécution (processus) à un environnement à unités d'exécution multiples. Dans l'environnement non fileté, certaines fonctions ne pouvaient être implémentées qu'en utilisant des signaux, bien que la plupart des applications n'aient pas besoin de connaître les signaux et la gestion des signaux. Dans l'environnement à unités d'exécution multiples, les primitives basées sur des unités d'exécution prennent en charge certaines des fonctions qui étaient implémentées dans les environnements sans unités d'exécution à l'aide de signaux.

Dans de nombreux cas, les signaux et le traitement des signaux, bien que pris en charge, ne s'intègrent pas bien dans l'environnement à unités d'exécution multiples et diverses restrictions existent. Cela peut poser problème lorsque vous intégrez du code d'application à différentes bibliothèques de middleware (s'exécutant dans le cadre de l'application) dans un environnement à unités d'exécution multiples où chacune tente de gérer les signaux. L'approche traditionnelle de la sauvegarde et de la restauration des gestionnaires de signaux (définis par processus), qui fonctionnait lorsqu'il n'y avait qu'une seule unité d'exécution dans un processus, ne fonctionne pas dans un environnement à unités d'exécution multiples. En effet, de nombreuses unités d'exécution peuvent tenter d'enregistrer et de restaurer une ressource à l'échelle du processus, avec des résultats imprévisibles.

Linux AIX *Applications sans unités d'exécution*

Chaque fonction MQI configure son propre gestionnaire de signaux pour les signaux. Les gestionnaires de ces utilisateurs sont remplacés pendant la durée de l'appel de fonction MQI. D'autres signaux peuvent être interceptés de manière normale par des gestionnaires écrits par l'utilisateur.

Chaque fonction MQI définit son propre gestionnaire de signaux pour les signaux:

SIGALRM
SIGBUS
SIGFPE
SIGSEGV
SIGILL

Les gestionnaires de ces utilisateurs sont remplacés pendant la durée de l'appel de fonction MQI. D'autres signaux peuvent être interceptés de manière normale par des gestionnaires écrits par l'utilisateur. Si vous n'installez pas de gestionnaire, les actions par défaut (par exemple, ignorer, cliché du processus core ou quitter) sont laissées en place.

Une fois que IBM MQ traite un signal synchrone (SIGSEGV, SIGBUS, SIGFPE, SIGILL), il tente de transmettre le signal à n'importe quel gestionnaire de signaux enregistré avant de passer l'appel de fonction MQI.

Linux AIX *Applications à unités d'exécution*

Une unité d'exécution est considérée comme étant connectée à IBM MQ à partir de MQCONN (ou MQCONNX) jusqu'à MQDISC.

Signaux synchrones

Des signaux synchrones apparaissent dans une unité d'exécution spécifique.

Les systèmes AIX and Linux permettent en toute sécurité de configurer un gestionnaire de signaux pour ces signaux pour l'ensemble du processus. Toutefois, IBM MQ configure son propre gestionnaire pour les signaux suivants, dans le processus d'application, alors que toute unité d'exécution est connectée à IBM MQ:

SIGBUS
SIGFPE
SIGSEGV
SIGILL

Si vous écrivez des applications à unités d'exécution multiples, il n'y a qu'un seul gestionnaire de signal à l'échelle du processus pour chaque signal. Lorsque IBM MQ configure ses propres gestionnaires de signaux synchrones, il sauvegarde tous les gestionnaires précédemment enregistrés pour chaque signal. Une fois que IBM MQ a traité l'un des signaux répertoriés, IBM MQ tente d'appeler le gestionnaire de signaux qui était en vigueur au moment de la première connexion IBM MQ dans le processus. Les gestionnaires précédemment enregistrés sont restaurés lorsque toutes les unités d'exécution d'application se sont déconnectées de IBM MQ.

Etant donné que les gestionnaires de signaux sont sauvegardés et restaurés par IBM MQ, les unités d'exécution d'application ne doivent pas établir de gestionnaires de signaux pour ces signaux alors qu'il est possible qu'une autre unité d'exécution du même processus soit également connectée à IBM MQ.

Remarque : Lorsqu'une application ou une bibliothèque de middlewares (s'exécutant dans le cadre d'une application) établit un gestionnaire de signaux alors qu'une unité d'exécution est connectée à IBM MQ, le gestionnaire de signaux de l'application doit appeler le gestionnaire IBM MQ correspondant lors du traitement de ce signal.

Lors de l'établissement et de la restauration de gestionnaires de signaux, le principe général est que le dernier gestionnaire de signaux à sauvegarder doit être le premier à être restauré:

- Lorsqu'une application établit un gestionnaire de signaux après la connexion à IBM MQ, le gestionnaire de signaux précédent doit être restauré avant que l'application ne se déconnecte de IBM MQ.
- Lorsqu'une application établit un gestionnaire de signaux avant de se connecter à IBM MQ, elle doit se déconnecter de IBM MQ avant de restaurer son gestionnaire de signaux.

Remarque : Le fait de ne pas respecter le principe général selon lequel le dernier gestionnaire de signaux à sauvegarder doit être le premier à être restauré peut entraîner un traitement inattendu des signaux dans l'application et, potentiellement, la perte de signaux par l'application.

Signaux asynchrones

IBM MQ n'utilise pas de signaux asynchrones dans les applications à unités d'exécution, sauf s'il s'agit d'applications client.

Remarques supplémentaires concernant les applications client à unités d'exécution

IBM MQ gère les signaux suivants lors des entrées-sorties vers un serveur. Ces signaux sont définis par la pile de communication. L'application ne doit pas établir de gestionnaire de signaux pour ces signaux lorsqu'une unité d'exécution est connectée à un gestionnaire de files d'attente:

SIGPIPE (pour TCP/IP)

Linux AIX *Remarques supplémentaires lors de l'utilisation de la gestion des signaux AIX and Linux dans MQI*

Lorsque vous utilisez MQI pour le traitement des signaux sous AIX and Linux, vous devez prendre en compte les applications de raccourci, les appels de fonction MQI dans les gestionnaires de signaux, les

signaux lors des appels MQI, les exits utilisateur et les services installables, ainsi que les gestionnaires d'exit VMS.

Applications Fastpath (dignes de confiance)

Les applications Fastpath s'exécutent dans le même processus que IBM MQ et sont donc exécutées dans l'environnement à unités d'exécution multiples.

Dans cet environnement, IBM MQ gère les signaux synchrones SIGSEGV, SIGBUS, SIGFPE et SIGILL. Tous les autres signaux ne doivent pas être distribués à l'application Fastpath lorsqu'elle est connectée à IBM MQ. Au lieu de cela, ils doivent être bloqués ou gérés par l'application. Si une application Fastpath intercepte un tel événement, le gestionnaire de files d'attente doit être arrêté et redémarré, ou il peut être laissé dans un état non défini. Pour la liste complète des restrictions applicables aux applications Fastpath sous MQCONNX, voir «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 755.

Appels de fonction MQI dans les gestionnaires de signaux

Lorsque vous êtes dans un gestionnaire de signaux, n'appelez pas de fonction MQI.

Si vous tentez d'appeler une fonction MQI à partir d'un gestionnaire de signaux alors qu'une autre fonction MQI est active, MQRC_CALL_IN_PROGRESS est renvoyé. Si vous essayez d'appeler une fonction MQI à partir d'un gestionnaire de signaux alors qu'aucune autre fonction MQI n'est active, elle risque d'échouer au cours de l'opération en raison des restrictions du système d'exploitation où seuls des appels sélectifs peuvent être émis à partir ou à l'intérieur d'un gestionnaire.

Pour les méthodes de destructeur C++, qui peuvent être appelées automatiquement lors de l'exit de programme, il se peut que vous ne puissiez pas arrêter l'appel des fonctions MQI. Ignorez les erreurs liées à MQRC_CALL_IN_PROGRESS. Si un gestionnaire de signaux appelle exit (), IBM MQ annule les messages non validés dans le point de synchronisation comme d'habitude et ferme les files d'attente ouvertes.

Signaux lors des appels MQI

Les fonctions MQI ne renvoient pas le code EINTR ou un code équivalent aux programmes d'application.

Si un signal se produit lors d'un appel MQI et que le gestionnaire appelle *return*, l'appel continue de s'exécuter comme si le signal ne s'était pas produit. En particulier, MQGET ne peut pas être interrompu par un signal pour renvoyer immédiatement le contrôle à l'application. Si vous souhaitez sortir d'une requête MQGET, définissez la file d'attente sur GET_DISABLED; vous pouvez également utiliser une boucle autour d'un appel à MQGET avec une expiration de temps finie (MQGMO_WAIT avec gmo.WaitInterval défini) et utiliser votre gestionnaire de signaux (dans un environnement non à unités d'exécution) ou une fonction équivalente dans un environnement à unités d'exécution pour définir un indicateur qui rompt la boucle.

AIX Dans l'environnement AIX , IBM MQ requiert que les appels système interrompus par des signaux soient redémarrés. Lors de l'établissement de votre propre gestionnaire de signaux avec sigaction (2), définissez l'indicateur SA_RESTART dans la zone sa_flags de la nouvelle structure d'action, sinon IBM MQ risque de ne pas pouvoir terminer un appel interrompu par un signal.

Exits utilisateur et services installables

Les exits utilisateur et les services installables qui s'exécutent dans le cadre d'un processus IBM MQ dans un environnement à unités d'exécution multiples sont soumis aux mêmes restrictions que pour les applications à chemins d'accès rapide. Considérez qu'ils sont connectés en permanence à IBM MQ et qu'ils n'utilisent donc pas de signaux ou d'appels de système d'exploitation non autorisant les unités d'exécution multiples.

Connexion et déconnexion d'un gestionnaire de files d'attente

Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

La façon dont cette connexion est établie dépend de la plateforme et de l'environnement dans lequel le programme fonctionne:

Multi **IBM MQ for Multiplatforms**
Les programmes qui s'exécutent dans ces environnements peuvent utiliser l'appel MQI MQCONN pour se connecter à un gestionnaire de files d'attente et l'appel MQDISC pour se déconnecter de ce dernier. Les programmes peuvent également utiliser l'appel MQCONNX.

z/OS **IBM MQ for z/OS par lots**
Les programmes qui s'exécutent dans cet environnement peuvent utiliser l'appel MQI MQCONN pour se connecter à un gestionnaire de files d'attente et l'appel MQDISC pour se déconnecter de ce dernier. Les programmes peuvent également utiliser l'appel MQCONNX.

Les programmes batch z/OS peuvent se connecter, consécutivement ou simultanément, à plusieurs gestionnaires de files d'attente sur le même bloc de contrôle des tâches.

z/OS **IMS**
La région de contrôle IMS est connectée à un ou plusieurs gestionnaires de files d'attente lorsqu'elle démarre. Cette connexion est contrôlée par les commandes IMS . Pour plus d'informations sur le contrôle de l'adaptateur IMS sous z/OS, voir Administration de IBM MQ for z/OS. Toutefois, les programmes d'écriture des files d'attente de messages IMS doivent utiliser l'appel MQI MQCONN pour spécifier le gestionnaire de files d'attente auquel ils souhaitent se connecter. Ils peuvent utiliser l'appel MQDISC pour se déconnecter de ce gestionnaire de files d'attente.

A la suite d'un appel IMS qui établit un point de synchronisation et avant de traiter un message pour un autre utilisateur, l'adaptateur IMS s'assure que l'application ferme les descripteurs et se déconnecte du gestionnaire de files d'attente. Voir «Syncpoints in IMS applications», à la page 880.

Les programmes IMS peuvent se connecter, consécutivement ou simultanément, à plusieurs gestionnaires de files d'attente sur le même bloc de contrôle des tâches.

z/OS **CICS Transaction Server pour z/OS**
Les programmes CICS n'ont pas besoin d'effectuer de travail pour se connecter à un gestionnaire de files d'attente car le système CICS lui-même est connecté. Cette connexion est généralement établie automatiquement lors de l'initialisation, mais vous pouvez également utiliser la transaction CKQC fournie avec IBM MQ for z/OS. Pour plus d'informations sur CKQC, voir Administration de IBM MQ for z/OS.

Les tâches CICS ne peuvent se connecter qu'au gestionnaire de files d'attente auquel la région CICS est connectée.

Les programmes CICS peuvent également utiliser les appels de connexion et de déconnexion MQI (MQCONN et MQDISC). Vous souhaiterez peut-être effectuer cette opération afin de pouvoir porter ces applications dans des environnements nonCICS avec un minimum de recodage. Toutefois, ces appels *toujours* aboutissent dans un environnement CICS . Cela signifie que le code retour peut ne pas refléter l'état réel de la connexion au gestionnaire de files d'attente.

TXSeries for Windows et Open Systems
Ces programmes n'ont pas besoin d'effectuer de travail pour se connecter à un gestionnaire de files d'attente car le système CICS lui-même est connecté. Par conséquent, une seule connexion à la fois est prise en charge. Les applications CICS doivent émettre un appel MQCONN pour obtenir un descripteur de connexion et un appel MQDISC avant de quitter.

Utilisez les liens suivants pour en savoir plus sur la connexion et la déconnexion d'un gestionnaire de files d'attente:

- «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONN», à la page 754
- «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 755

- «Déconnexion de programmes d'un gestionnaire de files d'attente à l'aide de MQDISC», à la page 759

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONN

Utilisez ces informations pour apprendre à se connecter à un gestionnaire de files d'attente à l'aide de l'appel MQCONN.

En général, vous pouvez vous connecter à un gestionnaire de files d'attente spécifique ou au gestionnaire de files d'attente par défaut:

- z/OS Pour IBM MQ for z/OS, dans l'environnement de traitement par lots, le gestionnaire de files d'attente par défaut est spécifié dans le module CSQBDEFV.
- Multi Pour IBM MQ for Multiplatforms, le gestionnaire de files d'attente par défaut est spécifié dans le fichier mqs.ini .

z/OS Sinon, dans les environnements z/OS MVS par lots, TSO et RRS, vous pouvez vous connecter à n'importe quel gestionnaire de files d'attente au sein d'un groupe de partage de files d'attente. La demande MQCONN ou MQCONNX sélectionne l'un des membres actifs du groupe.

Lorsque vous vous connectez à un gestionnaire de files d'attente, il doit être local à la tâche. Il doit appartenir au même système que l'application IBM MQ .

z/OS Dans l'environnement IMS , le gestionnaire de files d'attente doit être connecté à la région de contrôle IMS et à la région dépendante utilisée par le programme. Le gestionnaire de files d'attente par défaut est spécifié dans le module CSQQDEFV lorsque IBM MQ for z/OS est installé.

Avec l'environnement TXSeries CICS et TXSeries pour Windows et AIX, le gestionnaire de files d'attente doit être défini en tant que ressource XA dans CICS.

Pour vous connecter au gestionnaire de files d'attente par défaut, appelez MQCONN en spécifiant un nom composé entièrement de blancs ou en commençant par un caractère null (X'00').

Une application doit être autorisée pour qu'elle puisse se connecter à un gestionnaire de files d'attente. Pour plus d'informations, voir Sécurisation.

La sortie de MQCONN est la suivante:

- Un descripteur de connexion (**Hconn**)
- Un code achèvement
- Un code anomalie

Utilisez le descripteur de connexion lors des appels MQI suivants.

Si le code anomalie indique que l'application est déjà connectée à ce gestionnaire de files d'attente, le descripteur de connexion renvoyé est identique à celui qui a été renvoyé lors de la première connexion de l'application. L'application ne doit pas émettre l'appel MQDISC dans cette situation car l'application appelante s'attend à rester connectée.

La portée du descripteur de connexion est identique à celle du descripteur d'objet (voir «Ouverture d'objets à l'aide de l'appel MQOPEN», à la page 762).

Les descriptions des paramètres sont fournies dans la description de l'appel MQCONN dans MQCONN.

L'appel MQCONN échoue si le gestionnaire de files d'attente est à l'état de mise au repos lorsque vous émettez l'appel ou si le gestionnaire de files d'attente est en cours d'arrêt.

Portée de MQCONN ou MQCONNX

La portée d'un appel MQCONN ou MQCONNX est généralement l'unité d'exécution qui l'a émis. Autrement dit, le descripteur de connexion renvoyé par l'appel n'est valide que dans l'unité d'exécution qui a émis l'appel. Un seul appel peut être effectué à la fois à l'aide de l'indicateur. S'il est utilisé à partir d'une autre unité d'exécution, il est rejeté comme non valide. Si vous disposez de plusieurs unités d'exécution dans votre application et que chacune d'elles souhaite utiliser des appels IBM MQ , chacune d'elles doit émettre MQCONN ou MQCONNX.

Il n'est pas nécessaire que chaque appel soit effectué vers le même gestionnaire de files d'attente lorsqu'un processus effectue plusieurs appels MQCONN. Toutefois, une seule connexion IBM MQ peut être établie à partir d'une unité d'exécution à la fois. Vous pouvez également envisager «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 757 pour autoriser l'utilisation de plusieurs connexions IBM MQ à partir d'une seule unité d'exécution et d'une connexion IBM MQ à partir de n'importe quelle unité d'exécution.[7]

Si votre application s'exécute en tant que client, elle peut se connecter à plusieurs gestionnaires de files d'attente au sein d'une unité d'exécution.

Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX

L'appel MQCONNX est similaire à l'appel MQCONN, mais inclut des options permettant de contrôler le mode de fonctionnement de l'appel.

En entrée de MQCONNX, vous pouvez fournir un nom de gestionnaire de files d'attente z/OS ou un nom de groupe de partage de files d'attente sur les z/OS systèmes de files d'attente partagées. Les options permettant de contrôler la manière dont la connexion est établie avec le gestionnaire de files d'attente sont fournies dans une structure appelée MQCNO.

La sortie de MQCONNX est la suivante:

- Un descripteur de connexion (Hconn)
- Un code achèvement

[7] Lorsque vous utilisez des applications à unités d'exécution multiples avec des systèmes IBM MQ for AIX or Linux , vous devez vous assurer que les applications ont une taille de pile suffisante pour les unités d'exécution. Envisagez d'utiliser une taille de pile supérieure ou égale à 256 Ko lorsque des applications à unités d'exécution multiples font des appels MQI, soit par elles-mêmes, soit avec d'autres gestionnaires de signaux (par exemple, CICS).

- Un code anomalie

Vous utilisez le descripteur de connexion lors des appels MQI suivants.

Les options de connexion, définies dans la zone *Options* de la structure MQCNO, permettent de contrôler plusieurs attributs de la connexion. Il convient de noter en particulier les groupes d'options suivants:

- Les options de liaison permettent de créer des *applications sécurisées* . Les applications sécurisées impliquent que l'application IBM MQ et l'agent du gestionnaire de files d'attente local deviennent le même processus. Etant donné que le processus d'agent n'a plus besoin d'utiliser une interface pour accéder au gestionnaire de files d'attente, ces applications deviennent une extension du gestionnaire de files d'attente. Ce comportement est demandé en spécifiant l'option MQCNO_FASTPATH_BINDING . Pour plus d'informations sur les restrictions qui s'appliquent aux applications sécurisées, voir «Restrictions pour les applications sécurisées», à la page 756.
- Les options de partage de descripteur permettent de créer des connexions partagées. Les connexions partagées peuvent partager des descripteurs entre différentes unités d'exécution au sein d'un même processus. Pour plus d'informations sur les connexions partagées, voir «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 757.

MQCNO permet également à l'application de contrôler la manière dont la connexion au gestionnaire de files d'attente est authentifiée. Les données d'authentification peuvent être spécifiées dans une structure MQCSP référencée à partir de la structure MQCNO.

Pour une description complète des paramètres de l'appel MQCONNX et des attributs de connexion qui peuvent être contrôlés, voir MQCONNX-Connect queue manager (extended).

Restrictions pour les applications sécurisées
Restrictions qui s'appliquent aux applications sécurisées. Certaines restrictions s'appliquent à toutes les plateformes et d'autres sont spécifiques à la plateforme.

T

- Vous devez explicitement déconnecter les applications sécurisées du gestionnaire de files d'attente.
- Vous devez arrêter les applications sécurisées avant d'arrêter le gestionnaire de files d'attente à l'aide de la commande **endmqm** .
- Vous ne devez pas utiliser de signaux asynchrones et d'interruptions de temporisateur (telles que `sigkill`) avec MQCNO_FASTPATH_BINDING.
- Sur toutes les plateformes, une unité d'exécution d'une application sécurisée ne peut pas se connecter à un gestionnaire de files d'attente alors qu'une autre unité d'exécution du même processus est connectée à un gestionnaire de files d'attente différent.
- Linux AIX Sur les systèmes AIX and Linux , vous devez utiliser mqm comme userID et groupID effectifs pour tous les appels MQI. Vous pouvez modifier ces ID avant d'effectuer un appel non MQI nécessitant une authentification (par exemple, ouverture d'un fichier), mais vous devez le remplacer par mqm avant d'effectuer le prochain appel MQI.
- IBM i Sous IBM i :
  1. Les applications sécurisées doivent s'exécuter sous le profil utilisateur QMQM. Il ne suffit pas que le profil utilisateur soit membre du groupe QMQM ou que le programme adopte les droits QMQM. Il se peut que le profil utilisateur QMQM ne puisse pas être utilisé pour la connexion à des travaux interactifs ou qu'il ne puisse pas être spécifié dans la description de travail pour des travaux exécutant des applications sécurisées. Dans ce cas, une approche consiste à utiliser les fonctions d'API de permutation de profil IBM i , QSYGETPH, QWTSETP et QSYRLSPH pour remplacer temporairement l'utilisateur en cours du travail par QMQM pendant l'exécution des programmes IBM MQ . Les détails de ces fonctions, ainsi qu'un exemple de leur utilisation, sont fournis dans la section Security APIs de la documentation IBM i*Application programming interfaces* .
  2. N'annulez pas les applications sécurisées à l'aide de l'option 2 de la demande système ou en mettant fin aux travaux dans lesquels elles s'exécutent à l'aide de la commande ENDJOB.

- ALW Sur les systèmes AIX, Linux, and Windows , les applications 32 bits sécurisées ne sont pas prises en charge. Si vous essayez d'exécuter une application 32 bits sécurisée, elle sera rétrogradée à une connexion liée standard.

Multi *Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX*

Utilisez ces informations pour en savoir plus sur les connexions partagées avec MQCONNX et sur les remarques d'utilisation à prendre en compte.

Remarque : z/OS Non pris en charge sous IBM MQ for z/OS.

Sur Multiplatforms, une connexion établie avec MQCONN est disponible uniquement pour l'unité d'exécution qui a établi la connexion. Les options de l'appel MQCONNX permettent de créer une connexion pouvant être partagée par toutes les unités d'exécution d'un processus. Si votre application s'exécute dans un environnement transactionnel qui requiert l'émission d'appels MQI sur la même unité d'exécution, vous devez utiliser l'option par défaut suivante:

MQCNO_HANDLE_SHARE_NONE
Crée une connexion non partagée.

Dans la plupart des autres environnements, vous pouvez utiliser l'une des options de connexion partagée indépendante des unités d'exécution suivantes:

MQCNO_HANDLE_SHARE_BLOCK
Crée une connexion partagée. Sur une connexion MQCNO_HANDLE_SHARE_BLOCK , si la connexion est actuellement utilisée par un appel MQI sur une autre unité d'exécution, l'appel MQI attend la fin de l'appel MQI en cours.

MQCNO_HANDLE_SHARE_NO_BLOCK
Crée une connexion partagée. Sur une connexion MQCNO_HANDLE_SHARE_NO_BLOCK , si la connexion est actuellement utilisée par un appel MQI sur une autre unité d'exécution, l'appel MQI échoue immédiatement avec la raison MQRC_CALL_IN_PROGRESS.

A l'exception de l'environnement MTS (Microsoft Transaction Server), la valeur par défaut est MQCNO_HANDLE_SHARE_NONE. Dans l'environnement MTS, la valeur par défaut est MQCNO_HANDLE_SHARE_BLOCK.

Un descripteur de connexion est renvoyé par l'appel MQCONNX . Le descripteur peut être utilisé par les appels MQI ultérieurs provenant de n'importe quelle unité d'exécution du processus, en associant ces appels au descripteur renvoyé par MQCONNX. Les appels MQI utilisant un seul descripteur partagé sont sérialisés entre les unités d'exécution.

Par exemple, la séquence d'activité suivante est possible avec un descripteur partagé:

1. L'unité d'exécution 1 émet MQCONNX et obtient un descripteur partagé *h1*
2. L'unité d'exécution 1 ouvre une file d'attente et émet une demande d'extraction à l'aide de *h1*
3. L'unité d'exécution 2 émet une demande d'insertion à l'aide de *h1*
4. L'unité d'exécution 3 émet une demande d'insertion à l'aide de *h1*
5. Problèmes liés à l'unité d'exécution 2 MQDISC avec *h1*

Pendant que le descripteur est utilisé par une unité d'exécution, l'accès à la connexion n'est pas disponible pour les autres unités d'exécution. Dans les cas où il est acceptable qu'une unité d'exécution attende la fin d'un appel précédent d'une autre unité d'exécution, utilisez MQCONNX avec l'option MQCNO_HANDLE_SHARE_BLOCK.

Cependant, le blocage peut entraîner des difficultés. Supposons qu'à l'étape «2», à la page 757, l'unité d'exécution 1 émet une demande d'obtention qui attend les messages qui ne sont pas encore arrivés (une demande d'obtention avec attente). Dans ce cas, les unités d'exécution 2 et 3 sont également laissées en attente (bloquées) tant que la demande d'obtention sur l'unité d'exécution 1 est acceptée. Si vous préférez qu'un appel MQI renvoie une erreur si un autre appel MQI est déjà en cours d'exécution sur le descripteur, utilisez MQCONNX avec l'option MQCNO_HANDLE_SHARE_NO_BLOCK.

Remarques sur l'utilisation des connexions partagées

1. Tous les descripteurs d'objet (Hobj) créés par l'ouverture d'un objet sont associés à un Hconn ; ainsi, pour un Hconn partagé, les Hobjs sont également partagés et utilisables par n'importe quelle unité d'exécution utilisant le Hconn. De même, toute unité de travail démarrée sous un Hconn est associée à ce Hconn ; par conséquent, cette unité est également partagée entre les unités d'exécution avec le Hconn partagé.
2. *N'importe quelle unité d'exécution* peut appeler MQDISC pour déconnecter un Hconn partagé, et pas seulement l'unité d'exécution qui a appelé le MQCONNX correspondant. Le MQDISC arrête le Hconn en le rendant indisponible pour toutes les unités d'exécution.
3. Une seule unité d'exécution peut utiliser plusieurs connexions Hconns partagées en série, par exemple utiliser MQPUT pour placer un message sous une connexion Hconn partagée, puis placer un autre message à l'aide d'une autre connexion Hconn partagée, chaque opération étant sous une unité de travail locale différente.
4. Les connexions Hconns partagées ne peuvent pas être utilisées dans une unité d'oeuvre globale.

Multi *Utilisation des options d'appel MQCONNX avec MQ_CONNECT_TYPE*

Utilisez ces informations pour comprendre les différentes options d'appel MQCONNX et la façon dont elles sont utilisées avec la variable d'environnement **MQ_CONNECT_TYPE** .

Remarque : MQ_CONNECT_TYPE n'a d'effet que pour les liaisons STANDARD. Pour les autres liaisons, **MQ_CONNECT_TYPE** est ignoré.

Sous IBM MQ for Multiplatforms, vous pouvez utiliser la variable d'environnement **MQ_CONNECT_TYPE** en combinaison avec le type de liaison spécifié dans la zone `Options` de la structure MQCNO utilisée sur un appel MQCONNX.

*Tableau 113. Utilisation des options d'appel MQCONNX avec la variable d'environnement **MQ_CONNECT_TYPE***

Option d'appel MQCONNX	variable d'environnement MQ_CONNECT_TYPE	Résultat
STANDARD	NON DEFINI	STANDARD
STANDARD	STANDARD	STANDARD
STANDARD	Fastpath	STANDARD
STANDARD	CLIENT	CLIENT
STANDARD	LOCAL	STANDARD

Si MQCNO_STANDARD_BINDING n'est pas spécifié, vous pouvez utiliser MQCNO_NONE, qui prend par défaut la valeur MQCNO_STANDARD_BINDING.

Authentification et identité pour MQCONN et MQCONNX

Utilisez cette tâche pour savoir comment les applications peuvent fournir des données d'identification qui sont utilisées pour l'authentification lorsqu'elles se connectent à IBM MQ.

Identité de l'utilisateur par défaut

Lorsqu'une application utilise l'interface de file d'attente de messages (MQI) pour se connecter à IBM MQ avec MQCONN ou MQCONNX, une identité utilisateur est toujours établie et associée à la connexion.

Par défaut, l'identité de l'utilisateur initial est toujours celle du processus du système d'exploitation sous lequel l'application s'exécute. Cette identité initiale peut être suffisante pour les connexions d'application liées localement ou accréditées.

Lorsqu'une application se connecte à un gestionnaire de files d'attente avec un appel MQCONN, elle ne peut pas modifier l'ID utilisateur par défaut. Toutefois, les mécanismes suivants peuvent modifier l'ID utilisateur associé à la connexion:

- Exit de sécurité côté client ou côté serveur.
- Règles d'authentification de canal sur le gestionnaire de files d'attente.
- ID utilisateur client établi lors de l'authentification mutuelle TLS.

Utilisation de MQCONNX pour fournir des données d'identification

MQCONNX permet à une application de mieux contrôler l'identité associée à la connexion. Une application peut fournir une structure MQCSP dans le cadre des options de connexion spécifiées dans le paramètre **ConnectOpts** à MQCONNX. La structure MQCSP peut contenir des données d'identification utilisées pour établir une identité d'utilisateur. IBM MQ prend en charge les données d'identification suivantes dans la structure MQCSP:

- ID utilisateur et mot de passe.
- V 9.4.0 Depuis IBM MQ 9.3.4, jeton d'authentification, si l'application se connecte à un gestionnaire de files d'attente qui s'exécute sur des systèmes AIX ou Linux .

La configuration d'authentification de connexion et d'authentification de canal du gestionnaire de files d'attente contrôle le traitement des données d'identification fournies par une application. Par exemple, cette configuration affecte les aspects suivants:

- Indique si les données d'identification de la structure MQCSP sont validées et comment elles sont validées.
- Indique si l'ID utilisateur dans les données d'identification de la structure MQCSP est mappé à un autre ID utilisateur.
- Indique si l'utilisateur authentifié est ensuite adopté comme contexte de l'application.

Pour plus d'informations sur l'authentification de connexion, voir Authentification de connexion. Pour plus d'informations sur l'authentification de canal, voir Enregistrements d'authentification de canal.

Plusieurs des exemples de programmes écrits en C qui utilisent l'interface MQI montrent comment la structure MQCSP est utilisée pour fournir des données d'authentification. Pour plus d'informations, voir les exemples de programme suivants:

- «Exemples de programmes Get», à la page 1122
- «Exemples de programmes Put», à la page 1135
- «Exemple de programme de navigateur», à la page 1110
- «Exemple de programme TLS», à la page 1152

Information associée

Identification et authentification des utilisateurs à l'aide de la structure MQCSP

MQCSP-Paramètres de sécurité

Identification et authentification des utilisateurs

Déconnexion de programmes d'un gestionnaire de files d'attente à l'aide de MQDISC

Utilisez ces informations pour en savoir plus sur la déconnexion des programmes d'un gestionnaire de files d'attente à l'aide de MQDISC.

Lorsqu'un programme qui s'est connecté à un gestionnaire de files d'attente à l'aide de l'appel MQCONN ou MQCONNX a terminé toute interaction avec le gestionnaire de files d'attente, il interrompt la connexion à l'aide de l'appel MQDISC, sauf:

- z/OS Sur les applications CICS Transaction Server for z/OS , où l'appel est facultatif sauf si MQCONNX a été utilisé et que vous souhaitez supprimer la balise de connexion avant la fin de l'application.

- IBM i Sous IBM MQ for IBM i , lorsque vous vous déconnectez du système d'exploitation, un appel MQDISC implicite est effectué.

En tant qu'entrée de l'appel MQDISC, vous devez fournir le descripteur de connexion (Hconn) qui a été renvoyé par MQCONN ou MQCONNX lorsque vous vous êtes connecté au gestionnaire de files d'attente.

Sous CICS exécuté sur Multiplatforms, une fois que MQDISC est appelé, le descripteur de connexion (Hconn) n'est plus valide et vous ne pouvez pas émettre d'autres appels MQI tant que vous n'appelez pas à nouveau MQCONN ou MQCONNX. MQDISC effectue une opération MQCLOSE implicite pour tous les objets qui sont encore ouverts à l'aide de cet identificateur.

z/OS Pour un client connecté à z/OS, lorsqu'un appel MQDISC est émis, une validation implicite est effectuée, mais les descripteurs de file d'attente toujours ouverts ne sont pas fermés tant que le canal n'est pas réellement arrêté.

z/OS Si vous utilisez MQCONNX pour vous connecter à IBM MQ for z/OS, MQDISC met également fin à la portée de la balise de connexion établie par MQCONNX. Toutefois, dans une application CICS, IMSou RRS, si une unité de récupération active est associée à une balise de connexion, le MQDISC est rejeté avec le code anomalie MQRC_CONN_TAG_NOT_RELÂCHÉE.

Les descriptions des paramètres sont fournies dans la description de l'appel MQDISC dans MQDISC.

Lorsqu'aucun MQDISC n'est émis

Une connexion standard non partagée (Hconn) est nettoyée lorsque l'unité d'exécution de création se termine. Une connexion partagée n'est implicitement annulée et déconnectée que lorsque l'ensemble du processus se termine. Si l'unité d'exécution qui a créé le Hconn partagé s'arrête alors que le Hconn existe encore, le Hconn est toujours utilisable.

Vérification des droits d'accès

Les appels MQCLOSE et MQDISC n'effectuent généralement aucune vérification des droits.

Dans le cours normal des événements, un travail qui a le droit d'ouvrir ou de se connecter à un objet IBM MQ se ferme ou se déconnecte de cet objet. Même si les droits d'accès d'un travail qui s'est connecté à ou a ouvert un objet IBM MQ sont révoqués, les appels MQCLOSE et MQDISC sont acceptés.

Ouverture et fermeture d'objets

Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

Pour effectuer l'une des opérations suivantes, vous devez d'abord *ouvrir* l'objet IBM MQ approprié:

- Insertion de messages dans une file d'attente
- Extraire (parcourir ou extraire) des messages d'une file d'attente
- Définir les attributs d'un objet
- Renseignez-vous sur les attributs de tout objet

Utilisez l'appel MQOPEN pour ouvrir l'objet, en utilisant les options de l'appel pour indiquer ce que vous souhaitez faire avec l'objet. La seule exception est si vous souhaitez placer un message unique dans une file d'attente, puis fermer la file d'attente immédiatement. Dans ce cas, vous pouvez ignorer l'étape *ouvrant* à l'aide de l'appel MQPUT1 (voir «Insertion d'un message dans une file d'attente à l'aide de l'appel MQPUT1», à la page 781).

Avant d'ouvrir un objet à l'aide de l'appel MQOPEN, vous devez connecter votre programme à un gestionnaire de files d'attente. Ceci est expliqué en détail, pour tous les environnements, dans «Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753.

Il existe quatre types d'objet IBM MQ que vous pouvez ouvrir:

- File d'attente

- Liste de noms
- Définition de processus
- Gestionnaire de files d'attente

Vous ouvrez tous ces objets de la même manière à l'aide de l'appel MQOPEN. Pour plus d'informations sur les objets IBM MQ , voir Types d'objet.

Vous pouvez ouvrir le même objet plusieurs fois et chaque fois que vous obtenez un nouveau descripteur d'objet. Vous pouvez parcourir les messages d'une file d'attente à l'aide d'un descripteur et supprimer les messages de la même file d'attente à l'aide d'un autre descripteur. Cela permet d'éviter d'utiliser des ressources pour fermer et rouvrir le même objet. Vous pouvez également ouvrir une file d'attente pour parcourir *et* supprimer des messages en même temps.

En outre, vous pouvez ouvrir plusieurs objets avec un seul MQOPEN et les fermer à l'aide de MQCLOSE. Pour plus d'informations sur la procédure à suivre, voir «Listes de diffusion», à la page 782.

Lorsque vous tentez d'ouvrir un objet, le gestionnaire de files d'attente vérifie que vous êtes autorisé à ouvrir cet objet pour les options que vous spécifiez dans l'appel MQOPEN.

Les objets sont fermés automatiquement lorsqu'un programme se déconnecte du gestionnaire de files d'attente. Dans l'environnement IMS , la déconnexion est forcée lorsqu'un programme démarre le traitement d'un nouvel utilisateur suite à un appel GU (get unique) IMS . Sur la plateforme IBM i , les objets sont fermés automatiquement à la fin d'un travail.

Il est recommandé de fermer les objets que vous avez ouverts. Pour ce faire, utilisez l'appel MQCLOSE.

Utilisez les liens suivants pour en savoir plus sur l'ouverture et la fermeture d'objets:

- «Ouverture d'objets à l'aide de l'appel MQOPEN», à la page 762
- «Création de files d'attente dynamiques», à la page 770
- «Ouverture des files d'attente distantes», à la page 770
- «Fermeture d'objets à l'aide de l'appel MQCLOSE», à la page 771

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Ouverture d'objets à l'aide de l'appel MQOPEN

Utilisez ces informations pour en savoir plus sur l'ouverture d'objets à l'aide de l'appel MQOPEN.

En tant qu'entrée de l'appel MQOPEN, vous devez fournir:

- Un descripteur de connexion. Pour les applications CICS sur z/OS, vous pouvez spécifier la constante MQHC_DEF_HCONN (qui a la valeur zéro) ou utiliser le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX. Pour Multiplatforms, utilisez toujours le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX.
- Description de l'objet à ouvrir, à l'aide de la structure de descripteur d'objet (MQOD).
- Une ou plusieurs options qui contrôlent l'action de l'appel.

La sortie de MQOPEN est la suivante:

- Descripteur d'objet qui représente votre accès à l'objet. Utilisez cette option lors de l'entrée dans les appels MQI ultérieurs.
- Une structure de descripteur d'objet modifiée, si vous créez une file d'attente dynamique (et qu'elle est prise en charge sur votre plateforme).
- Code achèvement.
- Code anomalie.

Portée d'un descripteur d'objet

La portée d'un descripteur d'objet (Hobj) est la même que celle d'un descripteur de connexion (Hconn).

Cette rubrique est traitée dans «Portée de MQCONN ou MQCONNX», à la page 755 et «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 757. Toutefois, des considérations supplémentaires sont à prendre en compte dans certains environnements:

CICS
Dans un programme CICS , vous pouvez utiliser le descripteur uniquement dans la même tâche CICS à partir de laquelle vous avez effectué l'appel MQOPEN.

IMS et z/OS par lots
Dans l'environnement IMS et z/OS par lots , vous pouvez utiliser le descripteur dans la même tâche, mais pas dans les sous-tâches.

Les descriptions des paramètres de l'appel MQOPEN sont fournies dans MQOPEN.

Les sections suivantes décrivent les informations que vous devez fournir en entrée à MQOPEN.

Identification des objets (structure MQOD)

Utilisez la structure MQOD pour identifier l'objet à ouvrir. Cette structure est un paramètre d'entrée pour l'appel MQOPEN. (La structure est modifiée par le gestionnaire de files d'attente lorsque vous utilisez l'appel MQOPEN pour créer une file d'attente dynamique.)

Pour plus de détails sur la structure MQOD, voir MQOD.

Pour plus d'informations sur l'utilisation de la structure MQOD pour les listes de distribution, voir «Utilisation de la structure MQOD», à la page 784 sous «Listes de diffusion», à la page 782.

Résolution de nom
Manière dont l'appel MQOPEN résout les noms des files d'attente et des gestionnaires de files d'attente.

Remarque : Un alias de gestionnaire de files d'attente est une définition de file d'attente éloignée sans zone RNAME .

Lorsque vous ouvrez une file d'attente IBM MQ , l'appel MQOPEN exécute une fonction de résolution de nom sur le nom de file d'attente que vous indiquez. Détermine la file d'attente dans laquelle le

gestionnaire de files d'attente effectue les opérations suivantes. Cela signifie que lorsque vous spécifiez le nom d'une file d'attente alias ou d'une file d'attente éloignée dans votre descripteur d'objet (MQOD), l'appel résout le nom en file d'attente locale ou en file d'attente de transmission. Si une file d'attente est ouverte pour n'importe quel type d'entrée, de navigation ou d'ensemble, elle est convertie en file d'attente locale s'il en existe une, et échoue s'il n'en existe pas. Elle se résout en file d'attente non locale uniquement si elle est ouverte uniquement pour la sortie, l'interrogation uniquement ou la sortie et l'interrogation uniquement. Voir Tableau 114, à la page 763 pour une présentation du processus de résolution de nom. Le nom que vous fournissez dans *ObjectQMgrName* est résolu *avant* dans *ObjectName*.

La Tableau 114, à la page 763 montre également comment utiliser une définition locale d'une file d'attente éloignée pour définir un alias pour le nom d'un gestionnaire de files d'attente. Cela vous permet de sélectionner la file d'attente de transmission à utiliser lorsque vous placez des messages dans une file d'attente éloignée. Par exemple, vous pouvez utiliser une file d'attente de transmission unique pour les messages destinés à de nombreux gestionnaires de files d'attente éloignées.

Pour utiliser le tableau suivant, commencez par lire les deux colonnes de gauche, sous l'en-tête **Input to MQOD**, puis sélectionnez la casse appropriée. Lisez ensuite la ligne correspondante, en suivant les instructions. En suivant les instructions des colonnes **Noms résolus** , vous pouvez soit revenir aux colonnes **Entrée dans MQOD** et insérer des valeurs comme indiqué, soit quitter la table avec les résultats fournis. Par exemple, vous devrez peut-être entrer *ObjectName*.

Tableau 114. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN

Entrée dans MQOD	**Entrée dans MQOD**	**Noms résolus**	**Noms résolus**	**Noms résolus**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
Gestionnaire de files d'attente vide ou local	File d'attente locale sans attribut CLUSTER	Gestionnaire de files d'attente locales	Entrez *ObjectName*	Non applicable (file d'attente locale utilisée)
Gestionnaire de files d'attente vide	File d'attente locale avec attribut CLUSTER	Gestionnaire de files d'attente de cluster sélectionné pour la gestion de charge de travail ou gestionnaire de files d'attente de cluster sélectionné pour l'opération PUT	Entrez *ObjectName*	SYSTEME SYSTEM.CLUSTER.TRANSMIT.QUEUE et file d'attente locale utilisée SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)
Gestionnaire de files d'attente locales	File d'attente locale avec attribut CLUSTER	Gestionnaire de files d'attente locales	Entrez *ObjectName*	Non applicable (file d'attente locale utilisée)
Gestionnaire de files d'attente vide ou local	File d'attente modèle	Gestionnaire de files d'attente locales	Nom généré	Non applicable (file d'attente locale utilisée)

Tableau 114. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)

Entrée dans MQOD	Entrée dans MQOD	Noms résolus	Noms résolus	Noms résolus
Gestionnaire de files d'attente vide ou local	File d'attente alias avec ou sans attribut CLUSTER	Réexécutez la résolution de nom avec *ObjectQMgrNom* inchangé et entrez *ObjectName* défini sur *BaseQName* dans l'objet de définition de file d'attente alias. Ne doit pas être résolu en un alias défini en local dans lequel le *ObjectQMgrdu gestionnaire de files d'attente* est spécifié, mais peut être résolu en un alias en cluster (hébergé sur d'autres gestionnaires de files d'attente) dans lequel le *ObjectQMgrdu gestionnaire de files d'attente d'objets* est vide.		
Gestionnaire de files d'attente locales	File d'attente alias avec l'attribut CLUSTER	L'alias ne doit pas être résolu en une file d'attente de cluster qui n'est pas définie localement ou en une file d'attente de cluster qui a le même *ObjectName* que l'alias.		
Gestionnaire de files d'attente vide	File d'attente alias avec l'attribut CLUSTER	L'alias peut être résolu en file d'attente de cluster avec le même *ObjectName* que l'alias.		
Gestionnaire de files d'attente vide ou local	définition locale d'une file d'attente éloignée	Réexécutez la résolution de nom avec *ObjectQMgrNom* défini sur *RemoteQMgrNom*et *ObjectName* défini sur *RemoteQName*. Les files d'attente éloignées ne doivent pas être résolues		Nom de l'attribut *XmitQName* , s'il n'est pas vide ; dans le cas contraire, *RemoteQMgrName* dans l'objet de définition de file d'attente éloignée. SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)

Tableau 114. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)

Entrée dans MQOD	Entrée dans MQOD	Noms résolus	Noms résolus	Noms résolus
Gestionnaire de files d'attente vide	Aucun objet local correspondant ; file d'attente de cluster trouvée	Gestionnaire de files d'attente de cluster sélectionné pour la gestion de charge de travail ou gestionnaire de files d'attente de cluster sélectionné pour l'opération PUT	Entrez *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)
Gestionnaire de files d'attente vide ou local	Aucun objet local correspondant ; file d'attente de cluster introuvable		Erreur, file d'attente introuvable	Non applicable
Nom du gestionnaire de files d'attente dans le même groupe de partage de files d'attente que le gestionnaire de files d'attente local	File d'attente partagée locale	Gestionnaire de files d'attente locales	Entrez *ObjectName*	Non applicable
Nom d'une file d'attente de transmission locale	(Non résolu)	Entrez *ObjectQMgrName*	Entrez *ObjectName*	Entrez *ObjectQMgrName* SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)
Définition d'alias de gestionnaire de files d'attente (*RemoteQMgrName* peut être le gestionnaire de files d'attente local)	(Non résolu, file d'attente éloignée)	Réexécutez la résolution de nom avec *ObjectQMgrName* défini sur *RemoteQMgrName*. Ne doit pas être résolu en files d'attente distantes	Entrez *ObjectName*	Nom de l'attribut *XmitQName* , s'il n'est pas vide ; dans le cas contraire, *RemoteQMgrName* dans l'objet de définition de file d'attente éloignée. SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)
Le gestionnaire de files d'attente n'est pas le nom d'un objet local ; des gestionnaires de files d'attente de cluster ou un alias de gestionnaire de files d'attente ont été trouvés	(Non résolu)	*ObjectQMgrNom* ou gestionnaire de files d'attente de cluster spécifique sélectionné sur PUT	Entrez *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)

Tableau 114. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)				
Entrée dans MQOD	**Entrée dans MQOD**	**Noms résolus**	**Noms résolus**	**Noms résolus**
Le gestionnaire de files d'attente n'est pas le nom d'un objet local ; aucun objet de cluster trouvé	(Non résolu)	Entrez *ObjectQMgrName*	Entrez *ObjectName*	*AttributDefXmitQName* du gestionnaire de files d'attente où *DefXmitQName* est pris en charge. SYSTEME SYSTEM.QSG.TRANSMIT.QUEUE (voir la remarque)

Remarques :

1. *BaseQName* est le nom de la file d'attente de base provenant de la définition de la file d'attente alias.
2. *RemoteQName* est le nom de la file d'attente éloignée à partir de la définition locale de la file d'attente éloignée.
3. *RemoteQMgrName* est le nom du gestionnaire de files d'attente éloignées à partir de la définition locale de la file d'attente éloignée.
4. *XmitQName* est le nom de la file d'attente de transmission provenant de la définition locale de la file d'attente éloignée.
5. z/OS Pour les gestionnaires de files d'attente IBM MQ for z/OS faisant partie d'un groupe de partage de files d'attente (QSG), le nom du groupe de partage de files d'attente peut être utilisé à la place du nom du gestionnaire de files d'attente local dans Tableau 114, à la page 763. Si le gestionnaire de files d'attente local ne parvient pas à ouvrir la file d'attente cible ou à insérer un message dans la file d'attente, le message est transféré dans le nom ObjectQMgrspécifié via, la mise en file d'attente intra-groupe ou un canal IBM MQ .
6. Dans la colonne *ObjectName* de la table, CLUSTER fait référence aux attributs CLUSTER et CLUSNL de la file d'attente.
7. SYSTEM.QSG.TRANSMIT.QUEUE est utilisé si les gestionnaires de files d'attente locales et éloignées se trouvent dans le même groupe de partage de files d'attente ; la mise en file d'attente intra-groupe est activée.
8. Si vous avez affecté une file d'attente de transmission de cluster différente à chaque canal émetteur de cluster, SYSTEM.CLUSTER.TRANSMIT.QUEUE peut ne pas être le nom de la file d'attente de transmission du cluster. Pour plus d'informations sur les files d'attente de transmission de cluster multiples, voir Mise en cluster: Planification de la configuration des files d'attente de transmission de cluster.
9. Dans le cas où le gestionnaire de files d'attente n'est pas le nom d'un objet local, des gestionnaires de files d'attente de cluster ou un alias de gestionnaire de files d'attente ont été trouvés.

   Lorsque vous avez indiqué un nom de gestionnaire de files d'attente à l'aide de **ObjectQMgrName**et qu'il existe plusieurs canaux de cluster avec des noms de cluster différents connus du gestionnaire de files d'attente local qui atteindraient cette destination, vous pouvez utiliser l'un de ces canaux pour déplacer le message, quel que soit le nom de cluster de la file d'attente de destination.

   Cela peut être inattendu, si vous prévoyez que les messages de cette file d'attente soient envoyés uniquement via un canal ayant le même nom de cluster que la file d'attente.

   Toutefois, **ObjectQMgrName** est prioritaire dans ce cas, et l'équilibrage de la charge de travail du cluster prend en compte tous les canaux pouvant atteindre ce gestionnaire de files d'attente, quel que soit le nom du cluster dans lequel ils se trouvent.

L'ouverture d'une file d'attente alias ouvre également la file d'attente de base dans laquelle l'alias est résolu, et l'ouverture d'une file d'attente éloignée ouvre également la file d'attente de transmission. Par

conséquent, vous ne pouvez pas supprimer la file d'attente que vous spécifiez ou la file d'attente dans laquelle elle est résolue lorsque l'autre file d'attente est ouverte.

Alors qu'une file d'attente alias ne peut pas être résolue en une autre file d'attente alias définie localement (partagée dans un cluster ou non), la résolution en une file d'attente alias de cluster définie à distance est autorisée et peut donc être spécifiée comme file d'attente de base.

Le nom de la file d'attente résolue et le nom du gestionnaire de files d'attente résolu sont stockés dans les zones *ResolvedQName* et *ResolvedQMgrName* du MQOD.

Pour plus d'informations sur la résolution de nom dans un environnement de mise en file d'attente répartie, voir Qu'est-ce que la résolution de nom de file d'attente?.

Utilisation des options de l'appel MQOPEN

Dans le paramètre **Options** de l'appel MQOPEN, vous devez choisir une ou plusieurs options pour contrôler l'accès que vous avez à l'objet que vous ouvrez. Avec ces options, vous pouvez:

- Ouvrez une file d'attente et indiquez que tous les messages insérés dans cette file d'attente doivent être dirigés vers la même instance de celle-ci
- Ouvrir une file d'attente pour vous permettre d'y placer des messages
- Ouvrir une file d'attente pour vous permettre de parcourir les messages qu'elle contient
- Ouvrir une file d'attente pour vous permettre d'en supprimer des messages
- Ouvrir un objet pour vous permettre de demander et de définir ses attributs (mais vous pouvez définir les attributs des files d'attente uniquement)
- Ouvrir une rubrique ou une chaîne de rubrique pour y publier des messages
- Association d'informations de contexte à un message
- Désigner un autre identificateur d'utilisateur à utiliser pour les contrôles de sécurité
- Contrôler l'appel si le gestionnaire de files d'attente est à l'état de mise au repos

Option MQOPEN pour la file d'attente de cluster
La liaison utilisée pour l'identificateur de file d'attente est extraite de l'attribut de file d'attente **DefBind** , qui peut prendre la valeur MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDou MQBND_BIND_ON_GROUP.

Pour acheminer tous les messages insérés dans une file d'attente à l'aide de MQPUT vers le même gestionnaire de files d'attente par la même route, utilisez l'option MQOO_BIND_ON_OPEN sur l'appel MQOPEN .

Pour indiquer qu'une destination doit être sélectionnée au moment de l'appel MQPUT, c'est-à-dire message par message, utilisez l'option MQOO_BIND_NOT_FIXED sur l'appel MQOPEN .

Pour indiquer que tous les messages d'un groupe de messages placés dans une file d'attente à l'aide de MQPUT sont alloués à la même instance de destination, utilisez l'option MQOO_BIND_ON_GROUP sur l'appel MQOPEN .

MQOO_BIND_ON_OPEN ou MQOO_BIND_ON_GROUP doit être spécifié lors de l'utilisation de groupes de messages avec des clusters pour garantir que tous les messages du groupe sont traités à la même destination.

Si vous ne spécifiez aucune de ces options, la valeur par défaut, MQOO_BIND_AS_Q_DEF, est utilisée.

Si vous spécifiez le nom d'un gestionnaire de files d'attente dans MQOD, la file d'attente de ce gestionnaire de files d'attente est sélectionnée. Si le nom du gestionnaire de files d'attente est vide, n'importe quelle instance peut être sélectionnée. Pour plus d'informations, voir «MQOPEN et clusters», à la page 910.

Si vous ouvrez une file d'attente de cluster à l'aide d'une définition QALIAS , certains attributs de file d'attente sont définis par la file d'attente alias et non par la file d'attente de base. Les attributs de cluster figurent parmi les attributs de la définition de file d'attente de base qui sont remplacés par la file d'attente alias. Par exemple, dans le fragment suivant, la file d'attente de cluster est ouverte avec MQOO_BIND_NOT

FIXED et non avec MQOO_BIND_ON_OPEN. La définition de file d'attente de cluster est annoncée dans tout le cluster, la définition de file d'attente alias est locale pour le gestionnaire de files d'attente.

```
DEFINE QLOCAL(CLQ1) CLUSTER(MYCLUSTER) DEFBIND(OPEN) REPLACE
DEFINE QALIAS(ACLQ1) TARGET(CLQ1) DEFBIND(NOTFIXED) REPLACE
```

Option MQOPEN pour l'insertion de messages
Pour ouvrir une file d'attente ou une rubrique afin d'y placer des messages, utilisez l'option MQOO_OUTPUT.

Option MQOPEN pour l'exploration des messages
Pour ouvrir une file d'attente afin de *parcourir* les messages qu'elle contient, utilisez l'appel MQOPEN avec l'option MQOO_BROWSE.

Cela crée un *curseur de navigation* que le gestionnaire de files d'attente utilise pour identifier le message suivant dans la file d'attente. Pour plus d'informations, voir «Recherche de messages dans une file d'attente», à la page 823.

Remarque :

1. Vous ne pouvez pas parcourir les messages d'une file d'attente éloignée ; n'ouvrez pas de file d'attente éloignée à l'aide de l'option MQOO_BROWSE.
2. Vous ne pouvez pas spécifier cette option lors de l'ouverture d'une liste de distribution. Pour plus d'informations sur les listes de distribution, voir «Listes de diffusion», à la page 782.
3. Utilisez MQOO_CO_OP avec MQOO_BROWSE si vous utilisez la navigation coopérative ; voir Options

Options MQOPEN pour la suppression de messages
Trois options contrôlent l'ouverture d'une file d'attente pour en supprimer des messages.

Vous ne pouvez utiliser qu'un seul d'entre eux dans un appel MQOPEN. Ces options définissent si votre programme dispose d'un accès exclusif ou partagé à la file d'attente. *Accès exclusif* signifie que, jusqu'à ce que vous fermiez la file d'attente, vous seul pouvez en supprimer des messages. Si un autre programme tente d'ouvrir la file d'attente pour supprimer des messages, son appel MQOPEN échoue. *Accès partagé* signifie que plusieurs programmes peuvent être supprimés messages de la file d'attente.

L'approche la plus conseillée consiste à accepter le type d'accès prévu pour la file d'attente lorsque celle-ci a été définie. La définition de file d'attente impliquait la définition de **Shareability** et Attributs **DefInputOpenOption** . Pour accepter cet accès, utilisez l'option MQOO_INPUT_AS_Q_DEF. Voir Tableau 115, à la page 768 pour voir comment la définition de ces attributs affecte le type d'accès qui vous sera accordé lorsque vous utiliserez cette option.

Tableau 115. Comment les attributs et les options de file d'attente de l'appel MQOPEN affectent l'accès aux files d'attente

Attributs File d'attente		**Type d'accès avec les options MQOPEN**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**PARTAGÉ**	**EXCLUSIVE**
Partageable	PARTAGÉ	partagés	partagés	exclusif
Partageable	EXCLUSIVE	exclusif	partagés	exclusif
NON_PARTAGEABLE*	PARTAGE*	exclusif	exclusif	exclusif
NON PARTAGEABLE	EXCLUSIVE	exclusif	exclusif	exclusif

Remarque : * Bien que vous puissiez définir une file d'attente avec cette combinaison d'attributs, l'option d'ouverture d'entrée par défaut est remplacée par l'attribut de partageabilité.

Autres possibilités :

- Si vous savez que votre application peut fonctionner correctement même si d'autres programmes peuvent supprimer des messages de la file d'attente en même temps, utilisez l'option

MQOO_INPUT_SHARED. Tableau 115, à la page 768 montre comment, dans certains cas, vous bénéficierez d'un accès exclusif à la file d'attente, même avec cette option.

- Si vous savez que votre application ne peut fonctionner correctement que si d'autres programmes ne peuvent pas supprimer simultanément des messages de la file d'attente, utilisez l'option MQOO_INPUT_EXCLUSIVE.

Remarque :

1. Vous ne pouvez pas supprimer des messages d'une file d'attente éloignée. Par conséquent, vous ne pouvez pas ouvrir une file d'attente éloignée à l'aide des options MQOO_INPUT_ *.
2. Vous ne pouvez pas spécifier cette option lors de l'ouverture d'une liste de distribution. Pour plus d'informations, reportez-vous à la section «Listes de diffusion», à la page 782.

Options MQOPEN pour la définition et l'interrogation des attributs
Pour ouvrir une file d'attente afin de pouvoir définir ses attributs, utilisez l'option MQOO_SET.

Vous ne pouvez pas définir les attributs d'un autre type d'objet (voir «Interrogation et définition des attributs d'objet», à la page 873).

Pour ouvrir un objet afin de pouvoir vous renseigner sur ses attributs, utilisez l'option MQOO_INQUIRE.

Remarque : Vous ne pouvez pas spécifier cette option lors de l'ouverture d'une liste de distribution.

Options MQOPEN relatives au contexte de message
Si vous souhaitez pouvoir associer des informations de contexte à un message lorsque vous le placez dans une file d'attente, vous devez utiliser l'une des options de contexte de message lorsque vous ouvrez la file d'attente.

Les options permettent de faire la distinction entre les informations contextuelles relatives à l' *utilisateur* à l'origine du message et celles relatives à l' *application* à l'origine du message. En outre, vous pouvez choisir de définir les informations de contexte lorsque vous placez le message dans la file d'attente ou vous pouvez choisir que le contexte soit extrait automatiquement d'un autre descripteur de file d'attente.

Concepts associés
«Contexte de message», à la page 49
Les informations de *contexte de message* permettent à l'application qui extrait le message de trouver l'émetteur du message.

«Contrôle des informations de contexte de message», à la page 779
Lorsque vous utilisez l'appel MQPUT ou MQPUT1 pour placer un message dans une file d'attente, vous pouvez indiquer que le gestionnaire de files d'attente doit ajouter des informations de contexte par défaut au descripteur de message. Les applications disposant du niveau de droits approprié peuvent ajouter des informations de contexte supplémentaires. Vous pouvez utiliser la zone d'options de la structure MQPMO pour contrôler les informations de contexte.

Option MQOPEN pour les droits utilisateur alternatifs
Lorsque vous tentez d'ouvrir un objet à l'aide de l'appel MQOPEN, le gestionnaire de files d'attente vérifie que vous disposez des droits permettant d'ouvrir cet objet. Si vous n'êtes pas autorisé, l'appel échoue.

Toutefois, les programmes serveur peuvent vouloir que le gestionnaire de files d'attente vérifie l'autorisation de l'utilisateur pour lequel il travaille, plutôt que la propre autorisation du serveur. Pour ce faire, ils doivent utiliser l'option MQOO_ALTERNATE_USER_AUTHORITY de l'appel MQOPEN et spécifier l'autre ID utilisateur dans la zone *AlternateUserId* de la structure MQOD. En règle générale, le serveur obtient l'ID utilisateur à partir des informations de contexte du message qu'il traite.

z/OS *MQOPEN option for queue manager quiescing*
If you use the MQOPEN call when the queue manager is in a quiescing state, the call might fail, depending on which environment you are using.

In the CICS environment on z/OS, if you use the MQOPEN call when the queue manager is in a quiescing state, the call always fails.

In other z/OS and Multiplatforms environments, the call fails when the queue manager is quiescing only if you use the MQOO_FAIL_IF_QUIESCING option of the MQOPEN call.

Option MQOPEN pour la résolution des noms de file d'attente locale
Lorsque vous ouvrez une file d'attente locale, alias ou modèle, la file d'attente locale est renvoyée.

Toutefois, lorsque vous ouvrez une file d'attente éloignée ou une file d'attente de cluster, les zones *ResolvedQName* et *ResolvedQMgrName* de la structure MQOD sont renseignées avec les noms de la file d'attente éloignée et du gestionnaire de files d'attente éloignées trouvés dans la définition de file d'attente éloignée ou avec la file d'attente de cluster éloignée choisie.

Utilisez l'option MQOO_RESOLVE_LOCAL_Q de l'appel MQOPEN pour remplir le *ResolvedQName* dans la structure MQOD avec le nom de la file d'attente locale qui a été ouverte. Le *ResolvedQMgrName* est également rempli avec le nom du gestionnaire de files d'attente local qui héberge la file d'attente locale. Cette zone est disponible uniquement avec la version 3 de la structure MQOD ; si la structure est antérieure à la version 3, MQOO_RESOLVE_LOCAL_Q est ignoré sans qu'une erreur soit renvoyée.

Si vous spécifiez MQOO_RESOLVE_LOCAL_Q lors de l'ouverture, par exemple, d'une file d'attente éloignée, *ResolvedQName* est le nom de la file d'attente de transmission dans laquelle les messages seront insérés. *ResolvedQMgrName* est le nom du gestionnaire de files d'attente local hébergeant la file d'attente de transmission.

Création de files d'attente dynamiques

Utilisez une file d'attente dynamique lorsque vous n'avez pas besoin de la file d'attente après la fin de votre application.

Par exemple, vous pouvez utiliser une file d'attente dynamique pour votre file d'attente de réponse. Vous spécifiez le nom de la file d'attente de réponse dans la zone *ReplyToQ* de la structure MQMD lorsque vous placez un message dans une file d'attente (voir «Définition de messages à l'aide de la structure MQMD», à la page 774).

Pour créer une file d'attente dynamique, vous utilisez un modèle appelé file d'attente modèle, avec l'appel MQOPEN. Vous créez une file d'attente modèle à l'aide des commandes IBM MQ ou des panneaux d'opérations et de contrôle. La file d'attente dynamique que vous créez utilise les attributs de la file d'attente modèle.

Lorsque vous appelez MQOPEN, indiquez le nom de la file d'attente modèle dans la zone *ObjectName* de la structure MQOD. Une fois l'appel terminé, la zone *ObjectName* est définie sur le nom de la file d'attente dynamique créée. De plus, la zone *ObjectQMgrName* est définie sur le nom du gestionnaire de files d'attente local.

Vous pouvez spécifier le nom de la file d'attente dynamique que vous créez de trois manières:

- Indiquez le nom complet de votre choix dans la zone *DynamicQName* de la structure MQOD.
- Indiquez un préfixe (moins de 33 caractères) pour le nom et autorisez le gestionnaire de files d'attente à générer le reste du nom. Cela signifie que le gestionnaire de files d'attente génère un nom unique, mais que vous disposez toujours d'un contrôle (par exemple, vous pouvez souhaiter que chaque utilisateur utilise un certain préfixe ou que vous souhaitiez attribuer une classification de sécurité spéciale aux files d'attente avec un certain préfixe dans leur nom). Pour utiliser cette méthode, indiquez un astérisque (*) pour le dernier caractère non blanc de la zone *DynamicQName* . Ne spécifiez pas d'astérisque (*) unique pour le nom de la file d'attente dynamique.
- Autorisez le gestionnaire de files d'attente à générer le nom complet. Pour utiliser cette méthode, indiquez un astérisque (*) à la première position de caractère de la zone *DynamicQName* .

Pour plus d'informations sur ces méthodes, voir la description de la zone DynamicQName .

Pour plus d'informations sur les files d'attente dynamiques, voir Files d'attente dynamiques et modèles.

Ouverture des files d'attente distantes

Une file d'attente éloignée est une file d'attente appartenant à un gestionnaire de files d'attente autre que celui auquel l'application est connectée.

Pour ouvrir une file d'attente éloignée, utilisez l'appel MQOPEN comme pour une file d'attente locale. Vous pouvez spécifier le nom de la file d'attente comme suit:

1. Dans la zone *ObjectName* de la structure MQOD, indiquez le nom de la file d'attente éloignée connue du gestionnaire de files d'attente *local* .

   Remarque : Laissez la zone *ObjectQMgrName* vide dans ce cas.
2. Dans la zone *ObjectName* de la structure MQOD, indiquez le nom de la file d'attente éloignée, tel qu'il est connu du gestionnaire de files d'attente *éloignées* . Dans la zone *ObjectQMgrName* , indiquez l'une des options suivantes:

   - Nom de la file d'attente de transmission portant le même nom que le gestionnaire de files d'attente éloignées. Le nom et la casse (majuscules, minuscules ou un mélange) doivent correspondre *exactement*.
   - Nom d'un objet alias de gestionnaire de files d'attente qui se résout en gestionnaire de files d'attente de destination ou en file d'attente de transmission.

   Cela indique au gestionnaire de files d'attente la destination du message ainsi que la file d'attente de transmission sur laquelle il doit être placé pour s'y rendre.
3. Si *DefXmitQname* est pris en charge, dans la zone *ObjectName* de la structure MQOD, indiquez le nom de la file d'attente éloignée, tel qu'il est connu par le gestionnaire de files d'attente *éloigné* .

   Remarque : Définissez la zone *ObjectQMgrName* sur le nom du gestionnaire de files d'attente éloignées (elle ne peut pas être vide dans ce cas).

Seuls les noms locaux sont validés lorsque vous appelez MQOPEN ; la dernière vérification concerne l'existence de la file d'attente de transmission à utiliser.

Ces méthodes sont résumées dans leTableau 114, à la page 763.

Fermeture d'objets à l'aide de l'appel MQCLOSE

Pour fermer un objet, utilisez l'appel MQCLOSE.

Si l'objet est une file d'attente, notez ce qui suit:

- Il n'est pas nécessaire de vider une file d'attente dynamique temporaire avant de la fermer.

  Lorsque vous fermez une file d'attente dynamique temporaire, la file d'attente est supprimée, ainsi que les messages qui peuvent encore s'y trouver. Cela est vrai même si des appels MQGET, MQPUT ou MQPUT1 non validés sont en attente sur la file d'attente.
- z/OS Sous IBM MQ for z/OS, si vous avez des demandes MQGET avec une option MQGMO_SET_SIGNAL en attente pour cette file d'attente, elles sont annulées.
- Si vous avez ouvert la file d'attente à l'aide de l'option MQOO_BROWSE, votre curseur de navigation est détruit.

La fermeture n'est pas liée au point de synchronisation, vous pouvez donc fermer les files d'attente avant ou après le point de synchronisation.

En tant qu'entrée de l'appel MQCLOSE, vous devez fournir:

- Un descripteur de connexion. Utilisez le même descripteur de connexion que celui utilisé pour l'ouvrir. Pour les applications CICS sous z/OS, vous pouvez également spécifier la constante MQHC_DEF_HCONN (qui a la valeur zéro).
- Descripteur de l'objet à fermer. Vous pouvez l'obtenir à partir de la sortie de l'appel MQOPEN.
- MQCO_NONE dans la zone *Options* (sauf si vous fermez une file d'attente dynamique permanente).
- Option de contrôle permettant de déterminer si le gestionnaire de files d'attente doit supprimer la file d'attente même si elle contient encore des messages (lors de la fermeture d'une file d'attente dynamique permanente).

La sortie de MQCLOSE est la suivante:

- Un code achèvement

- Un code anomalie
- Descripteur d'objet, réinitialisé à la valeur MQHO_UNUSABLE_HOBJ

La description des paramètres de l'appel MQCLOSE est fournie dans MQCLOSE.

Insertion de messages dans une file d'attente

Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

Utilisez l'appel MQPUT pour placer des messages dans la file d'attente. Vous pouvez utiliser MQPUT à plusieurs reprises pour placer de nombreux messages dans la même file d'attente, à la suite de l'appel MQOPEN initial. Appelez MQCLOSE lorsque vous avez terminé d'insérer tous vos messages dans la file d'attente.

Si vous souhaitez placer un message unique dans une file d'attente et fermer la file d'attente immédiatement après, vous pouvez utiliser l'appel MQPUT1 . MQPUT1 exécute les mêmes fonctions que la séquence d'appels suivante:

- MQOPEN
- MQPUT
- MQCLOSE

Toutefois, en règle générale, si vous avez plusieurs messages à insérer dans la file d'attente, il est plus efficace d'utiliser l'appel MQPUT. Cela dépend de la taille du message et de la plateforme sur laquelle vous travaillez.

Utilisez les liens suivants pour en savoir plus sur l'insertion de messages dans une file d'attente:

- «Insertion de messages dans une file d'attente locale à l'aide de l'appel MQPUT», à la page 773
- «Insertion de messages dans une file d'attente éloignée», à la page 778
- «Définition des propriétés d'un message», à la page 778
- «Contrôle des informations de contexte de message», à la page 779
- «Insertion d'un message dans une file d'attente à l'aide de l'appel MQPUT1», à la page 781
- «Listes de diffusion», à la page 782
- «Certains cas où les appels d'insertion échouent», à la page 787

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Insertion de messages dans une file d'attente locale à l'aide de l'appel MQPUT

Utilisez ces informations pour en savoir plus sur l'insertion de messages dans une file d'attente locale à l'aide de l'appel MQPUT.

En tant qu'entrée de l'appel MQPUT, vous devez fournir:

- Un descripteur de connexion (Hconn).
- Un descripteur de file d'attente (Hobj).
- Description du message que vous souhaitez placer dans la file d'attente. Il s'agit d'une structure de descripteur de message (MQMD).
- Informations de contrôle, sous la forme d'une structure d'options d'insertion de message (MQPMO).
- Longueur des données contenues dans le message (MQLONG).
- Les données de message elles-mêmes.

La sortie de l'appel MQPUT est la suivante:

- Un code anomalie (MQLONG)
- Un code achèvement (MQLONG)

Si l'appel aboutit, il renvoie également votre structure d'options et votre structure de descripteur de message. L'appel modifie votre structure d'options pour afficher le nom de la file d'attente et le gestionnaire de files d'attente auquel le message a été envoyé. Si vous demandez que le gestionnaire de files d'attente génère une valeur unique pour l'identificateur du message que vous insérez (en spécifiant un zéro binaire dans la zone *MsgId* de la structure MQMD), l'appel insère la valeur dans la zone *MsgId* avant de vous renvoyer cette structure. Réinitialisez cette valeur avant d'émettre un autre MQPUT.

Il existe une description de l'appel MQPUT dans MQPUT.

Pour plus de description sur les informations requises comme entrée dans l'appel MQPUT, voir les liens suivants:

- «Spécification de descripteurs», à la page 773
- «Définition de messages à l'aide de la structure MQMD», à la page 774
- «Spécification d'options à l'aide de la structure MQPMO», à la page 774
- «Les données de votre message», à la page 777
- «Insertion de messages: Utilisation de descripteurs de message», à la page 778

Spécification de descripteurs

Pour le descripteur de connexion (*Hconn*) dans CICS sur les applications z/OS , vous pouvez spécifier la constante MQHC_DEF_HCONN (qui a la valeur zéro) ou utiliser le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX. Pour les autres applications, utilisez toujours le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX.

Quel que soit l'environnement dans lequel vous travaillez, utilisez le même descripteur de file d'attente (*Hobj*) que celui renvoyé par l'appel MQOPEN.

Définition de messages à l'aide de la structure MQMD

La structure de descripteur de message (MQMD) est un paramètre d'entrée-sortie pour les appels MQPUT et MQPUT1 . Utilisez-le pour définir le message que vous mettez dans une file d'attente.

Si MQPRI_PRIORITY_AS_Q_DEF ou MQPER_PERSISTENCE_AS_Q_DEF est spécifié pour le message et que la file d'attente est une file d'attente de cluster, les valeurs utilisées sont celles de la file d'attente dans laquelle MQPUT est résolu. Si cette file d'attente est désactivée pour MQPUT, l'appel échouera. Pour plus d'informations, voir Configuration d'un cluster de gestionnaires de files d'attente .

Remarque : Utilisez MQPMO_NEW_MSG_ID et MQPMO_NEW_CORREL_ID avant de placer un nouveau message pour vous assurer que *MsgId* et *CorrelId* sont uniques. Les valeurs de ces zones sont renvoyées lors d'une opération MQPUT réussie.

Il existe une introduction aux propriétés de message que MQMD décrit dans «Messages IBM MQ», à la page 19et une description de la structure elle-même dans MQMD.

Spécification d'options à l'aide de la structure MQPMO

Utilisez la structure MQPMO (Put Message Option) pour transmettre des options aux appels MQPUT et MQPUT1 .

Les sections suivantes vous aident à remplir les zones de cette structure. Il existe une description de la structure dans MQPMO.

La structure inclut les zones suivantes:

- *StrucId*
- *Version*
- *Options*
- *Context*
- *ResolvedQName*
- *ResolvedQMgrName*
- *RecsPresent*
- *PutMsgRecsFields*
- *ResponseRecOffset and ResponseRecPtr*
- *OriginalMsgHandle*
- *NewMsgHandle*
- *Action*
- *PubLevel*

Le contenu de ces zones est le suivant:

StrucId
Cette opération identifie la structure en tant que structure d'options d'insertion de message. Il s'agit d'une zone à 4 caractères. Indiquez toujours MQPMO_STRUC_ID.

Version
Décrit le numéro de version de la structure. La valeur par défaut est MQPMO_VERSION_1. Si vous entrez MQPMO_VERSION_2, vous pouvez utiliser des listes de distribution (voir «Listes de diffusion», à la page 782). Si vous entrez MQPMO_VERSION_3, vous pouvez utiliser les descripteurs de message et les propriétés de message. Si vous entrez MQPMO_CURRENT_VERSION, votre application est toujours définie pour utiliser le niveau le plus récent.

Options
Cette commande contrôle les éléments suivants:

- Indique si l'opération d'insertion est incluse dans une unité de travail
- Quantité d'informations de contexte associée à un message

- Emplacement d'où proviennent les informations contextuelles
- Indique si l'appel échoue si le gestionnaire de files d'attente est à l'état de mise au repos
- Indique si le regroupement ou la segmentation est autorisé
- Génération d'un nouvel identificateur de message et d'un nouvel identificateur de corrélation
- Ordre dans lequel les messages et les segments sont placés dans une file d'attente
- Indique si les noms de file d'attente locale doivent être résolus

Si vous laissez la zone *Options* définie sur la valeur par défaut (MQPMO_NONE), des informations de contexte par défaut sont associées au message que vous placez.

La façon dont l'appel fonctionne avec les points de synchronisation est déterminée par la plateforme. La valeur par défaut du contrôle de point de synchronisation est yes pour z/OS et no pour Multiplatforms.

Contexte
Indique le nom de l'identificateur de file d'attente à partir duquel les informations de contexte doivent être copiées (si elles sont demandées dans la zone *Options*).

Pour une introduction au contexte de message, voir «Contexte de message», à la page 49. Pour plus d'informations sur l'utilisation de la structure MQPMO pour contrôler les informations de contexte dans un message, voir «Contrôle des informations de contexte de message», à la page 779.

ResolvedQName
Contient le nom (après résolution de tout nom d'alias) de la file d'attente qui a été ouverte pour recevoir le message. Il s'agit d'une zone de sortie.

ResolvedQMgrNom
Contient le nom (après résolution de tout nom d'alias) du gestionnaire de files d'attente propriétaire de la file d'attente dans *ResolvedQName*. Il s'agit d'une zone de sortie.

Le MQPMO peut également prendre en charge les zones requises pour les listes de distribution (voir «Listes de diffusion», à la page 782). Si vous souhaitez utiliser cette fonction, la version 2 de la structure MQPMO est utilisée. Cela inclut les zones suivantes:

RecsPresent
Cette zone contient le nombre de files d'attente dans la liste de distribution, c'est-à-dire le nombre d'enregistrements de message d'insertion (MQPMR) et d'enregistrements de réponse correspondants (MQRR) présents.

La valeur que vous entrez peut être identique au nombre d'enregistrements d'objet fournis dans MQOPEN. Toutefois, si la valeur est inférieure au nombre d'enregistrements d'objet indiqué dans l'appel MQOPEN ou si vous n'indiquez aucun enregistrement de message d'insertion, les valeurs des files d'attente qui ne sont pas définies sont extraites des valeurs par défaut fournies par le descripteur de message. En outre, si la valeur est supérieure au nombre d'enregistrements d'objet indiqué, les enregistrements de message d'insertion en excès sont ignorés.

Il est recommandé d'effectuer l'une des opérations suivantes:

- Si vous souhaitez recevoir un rapport ou une réponse de chaque destination, entrez la même valeur que celle qui apparaît dans la structure MQOR et utilisez des MQPMR contenant des zones *MsgId* . Initialisez ces zones *MsgId* à zéro ou spécifiez MQPMO_NEW_MSG_ID.

  Une fois que vous avez inséré le message dans la file d'attente, les valeurs *MsgId* créées par le gestionnaire de files d'attente deviennent disponibles dans les MQPMR ; vous pouvez les utiliser pour identifier la destination associée à chaque rapport ou réponse.
- Si vous ne souhaitez pas recevoir de rapports ou de réponses, choisissez l'une des options suivantes:

  1. Si vous souhaitez identifier les destinations qui échouent immédiatement, vous pouvez toujours entrer la même valeur dans la zone *RecsPresent* que celle qui apparaît dans la structure MQOR et fournir des MQRRs pour identifier ces destinations. Ne spécifiez pas de MQPMR.

2. Si vous ne souhaitez pas identifier les destinations ayant échoué, entrez zéro dans la zone *RecsPresent* et ne fournissez pas de MQPMR ni de MQRRs.

Remarque : Si vous utilisez MQPUT1, le nombre de pointeurs d'enregistrement de réponse et de décalages d'enregistrement de réponse doit être égal à zéro.

Pour obtenir une description complète des enregistrements de message d'insertion (MQPMR) et des enregistrements de réponse (MQRR), voir MQPMR et MQRR.

PutMsgRecFields
Indique les zones présentes dans chaque enregistrement de message d'insertion (MQPMR). Pour obtenir la liste de ces zones, voir «Utilisation de la structure MQPMR», à la page 786.

PutMsgRecOffset* et *PutMsgRecPtr
Les pointeurs (généralement en C) et les décalages (généralement en COBOL) sont utilisés pour traiter les enregistrements de message d'insertion (voir «Utilisation de la structure MQPMR», à la page 786 pour une présentation de la structure MQPMR).

Utilisez la zone *PutMsgRecPtr* pour indiquer un pointeur vers le premier enregistrement de message d'insertion ou la zone *PutMsgRecOffset* pour indiquer le décalage du premier enregistrement de message d'insertion. Il s'agit du décalage par rapport au démarrage du MQPMO. Selon la zone *PutMsgRecFields* , entrez une valeur non null pour *PutMsgRecOffset* ou *PutMsgRecPtr*.

ResponseRecOffset et ResponseRecPtr
Vous utilisez également des pointeurs et des décalages pour traiter les enregistrements de réponse (voir «Utilisation de la structure MQRR», à la page 785 pour plus d'informations sur les enregistrements de réponse).

Utilisez la zone *ResponseRecPtr* pour indiquer un pointeur vers le premier enregistrement de réponse ou la zone *ResponseRecOffset* pour indiquer le décalage du premier enregistrement de réponse. Décalage par rapport au début de la structure MQPMO. Entrez une valeur non null pour *ResponseRecOffset* ou *ResponseRecPtr*.

Remarque : Si vous utilisez MQPUT1 pour insérer des messages dans une liste de distribution, *ResponseRecPtr* doit avoir la valeur null ou zéro et *ResponseRecOffset* doit avoir la valeur zéro.

La version 3 de la structure MQPMO inclut en outre les zones suivantes:

Descripteur OriginalMsg
L'utilisation que vous pouvez faire de cette zone dépend de la valeur de la zone *Action* . Si vous mettez un nouveau message avec des propriétés de message associées, définissez cette zone sur le descripteur de message que vous avez créé précédemment et définissez des propriétés. Si vous transmettez, répondez ou générez un rapport en réponse à un message précédemment extrait, cette zone contient le descripteur de ce message.

NewMsgIdentificateur
Si vous spécifiez *NewMsgHandle*, toutes les propriétés associées aux propriétés de substitution de descripteur associées à *OriginalMsgHandle*. Pour plus d'informations, voir Action (MQLONG).

Action
Cette zone permet de spécifier le type d'insertion en cours d'exécution. Les valeurs possibles et leur signification sont les suivantes:

MQACTP_NOUVEAU
Il s'agit d'un nouveau message sans rapport avec aucun autre.

MQACTP_FORWARD
Ce message a été extrait précédemment et est en cours de transfert.

MQACTP_REPLY
Ce message est une réponse à un message précédemment extrait.

RAPPORT MQACTP_RAPPORT
Ce message est un rapport généré suite à un message précédemment extrait.

Pour plus d'informations, voir Action (MQLONG).

PubLevel
Si ce message est une publication, vous pouvez définir cette zone pour déterminer les abonnements qui la reçoivent. Seuls les abonnements dont le *SubLevel* est inférieur ou égal à cette valeur recevront cette publication. La valeur par défaut est 9, qui correspond au niveau le plus élevé et signifie que les abonnements avec un *SubLevel* peuvent recevoir cette publication.

Les données de votre message

Indiquez l'adresse de la mémoire tampon qui contient vos données dans le paramètre **Buffer** de l'appel MQPUT. Vous pouvez inclure n'importe quoi dans les données de vos messages. Toutefois, la quantité de données dans les messages affecte les performances de l'application qui les traite.

La taille maximale des données est déterminée par:

- Attribut **MaxMsgLength** du gestionnaire de files d'attente
- Attribut **MaxMsgLength** de la file d'attente dans laquelle vous avez placé le message
- Taille de tout en-tête de message ajouté par IBM MQ (y compris l'en-tête de rebut, MQDLH et l'en-tête de liste de distribution, MQDH)

L'attribut **MaxMsgLength** du gestionnaire de files d'attente contient la taille des messages que le gestionnaire de files d'attente peut traiter. La valeur par défaut est de 100 Mo pour tous les produits IBM MQ de la version V6 ou ultérieure.

Pour déterminer la valeur de cet attribut, utilisez l'appel MQINQ sur l'objet gestionnaire de files d'attente. Pour les messages volumineux, vous pouvez modifier cette valeur.

L'attribut **MaxMsgLength** d'une file d'attente détermine la taille maximale des messages que vous pouvez placer dans la file d'attente. Si vous tentez d'insérer un message dont la taille est supérieure à la valeur de cet attribut, votre appel MQPUT échoue. Si vous placez un message dans une file d'attente éloignée, la taille maximale du message que vous pouvez insérer est déterminée par l'attribut **MaxMsgLength** de la file d'attente éloignée, des files d'attente de transmission intermédiaires sur lesquelles le message est inséré le long de la route vers sa destination et des canaux utilisés.

Pour une opération MQPUT, la taille du message doit être inférieure ou égale à l'attribut **MaxMsgLength** de la file d'attente et du gestionnaire de files d'attente. Les valeurs de ces attributs sont indépendantes, mais il est recommandé de définir le *MaxMsgLength* de la file d'attente sur une valeur inférieure ou égale à celle du gestionnaire de files d'attente.

IBM MQ ajoute des informations d'en-tête aux messages dans les cas suivants:

- Lorsque vous placez un message dans une file d'attente éloignée, IBM MQ ajoute une structure d'en-tête de transmission (MQXQH) au message. Cette structure inclut le nom de la file d'attente de destination et son gestionnaire de files d'attente propriétaire.
- Si IBM MQ ne parvient pas à distribuer un message dans une file d'attente éloignée, il tente de placer le message dans la file d'attente de rebut (message non distribué). Il ajoute une structure MQDLH au message. Cette structure inclut le nom de la file d'attente de destination et la raison pour laquelle le message a été inséré dans la file d'attente de rebut.
- Si vous souhaitez envoyer un message à plusieurs files d'attente de destination, IBM MQ ajoute un en-tête MQDH au message. Décrit les données présentes dans un message, appartenant à une liste de distribution, sur une file d'attente de transmission. Tenez compte de ce point lorsque vous choisissez une valeur optimale pour la longueur maximale des messages.
- Si le message est un segment ou un message dans un groupe, IBM MQ peut ajouter un MQMDE.

Ces structures sont décrites dans MQDH et MQMDE.

Si vos messages ont la taille maximale autorisée pour ces files d'attente, l'ajout de ces en-têtes signifie que les opérations d'insertion échouent car les messages sont maintenant trop volumineux. Pour réduire les risques d'échec des opérations d'insertion:

- Réduisez la taille de vos messages par rapport à l'attribut **MaxMsgLength** des files d'attente de transmission et de rebut. Autorisez au moins la valeur de la constante MQ_MSG_HEADER_LENGTH (plus pour les listes de distribution volumineuses).
- Assurez-vous que l'attribut **MaxMsgLength** de la file d'attente de rebut est défini sur la valeur *MaxMsgLength* du gestionnaire de files d'attente propriétaire de la file d'attente de rebut.

Les attributs du gestionnaire de files d'attente et les constantes de mise en file d'attente des messages sont décrits dans Attributs du gestionnaire de files d'attente.

z/OS Pour plus d'informations sur la façon dont les messages non distribués sont gérés dans un environnement de mise en file d'attente répartie, voir Messages non distribués / non traités.

Insertion de messages: Utilisation de descripteurs de message

Deux descripteurs de message sont disponibles dans la structure MQPMO: *OriginalMsgHandle* et *NewMsgHandle*. La relation entre ces descripteurs de message est définie par la valeur de la zone *Action* MQPMO.

Pour plus de détails, voir Action (MQLONG). Un descripteur de message n'est pas nécessairement requis pour insérer un message. Son but est d'associer des propriétés à un message. Il n'est donc requis que si vous utilisez des propriétés de message.

Insertion de messages dans une file d'attente éloignée

Lorsque vous souhaitez placer un message dans une file d'attente éloignée (c'est-à-dire une file d'attente appartenant à un gestionnaire de files d'attente autre que celui auquel votre application est connectée) plutôt qu'une file d'attente locale, la seule considération supplémentaire à prendre en compte est la manière dont vous spécifiez le nom de la file d'attente lorsque vous l'ouvrez.Ceci est décrit dans «Ouverture des files d'attente distantes», à la page 770. La façon dont vous utilisez l'appel MQPUT ou MQPUT1 pour une file d'attente locale n'est pas modifiée.

Pour plus d'informations sur l'utilisation des files d'attente distantes et de transmission, voir Techniques de mise en file d'attente répartie IBM MQ.

Définition des propriétés d'un message

Appelez MQSETMP pour chaque propriété à définir. Lorsque vous placez le message, définissez le descripteur de message et les zones d'action de la structure MQPMO.

Pour associer des propriétés à un message, le message doit avoir un descripteur de message. Créez un descripteur de message à l'aide de l'appel de fonction MQCRTMH. Appelez MQSETMP en spécifiant ce descripteur de message pour chaque propriété à définir. Un exemple de programme, amqsstma.c, est fourni pour illustrer l'utilisation de MQSETMP.

S'il s'agit d'un nouveau message, lorsque vous le placez dans une file d'attente à l'aide de MQPUT ou de MQPUT1, définissez la zone Descripteur OriginalMsgdu MQPMO sur la valeur de ce descripteur de message et définissez la zone Action MQPMO sur MQACTP_NEW (il s'agit de la valeur par défaut).

S'il s'agit d'un message que vous avez précédemment extrait et que vous lui transmettez ou y répondez ou que vous lui envoyez un rapport en réponse, placez le descripteur de message d'origine dans la zone de descripteur OriginalMsgdu MQPMO et le nouveau descripteur de message dans la zone de descripteur NewMsg. Définissez la zone Action sur MQACTP_FORWARD, MQACTP_REPLY ou MQACTP_REPORT, selon le cas.

Si vous avez des propriétés dans un en-tête MQRFH2 à partir d'un message que vous avez précédemment extrait, vous pouvez les convertir en propriétés de descripteur de message à l'aide de l'appel MQBUFMH.

Si vous mettez votre message dans une file d'attente d'un gestionnaire de files d'attente à un niveau antérieur à IBM WebSphere MQ 7.0, qui ne peut pas traiter les propriétés de message, vous pouvez définir le paramètre PropertyControl dans la définition de canal pour spécifier comment les propriétés doivent être traitées.

Contrôle des informations de contexte de message

Lorsque vous utilisez l'appel MQPUT ou MQPUT1 pour placer un message dans une file d'attente, vous pouvez indiquer que le gestionnaire de files d'attente doit ajouter des informations de contexte par défaut au descripteur de message. Les applications disposant du niveau de droits approprié peuvent ajouter des informations de contexte supplémentaires. Vous pouvez utiliser la zone d'options de la structure MQPMO pour contrôler les informations de contexte.

Les informations de contexte de message permettent à l'application qui extrait le message de connaître l'émetteur du message. Toutes les informations de contexte sont stockées dans les zones de contexte du descripteur de message. Le type d'informations correspond aux informations d'identité, d'origine et de contexte utilisateur.

Pour contrôler les informations de contexte, utilisez la zone *Options* dans la structure MQPMO.

Si vous ne spécifiez aucune option pour les informations de contexte, le gestionnaire de files d'attente remplace les informations de contexte qui se trouvent déjà dans le descripteur de message par les informations d'identité et de contexte qu'il a générées pour votre message. Cela revient à spécifier l'option MQPMO_DEFAULT_CONTEXT.Vous pouvez avoir besoin de ces informations contextuelles par défaut lorsque vous créez un nouveau message (par exemple, lors du traitement d'une entrée utilisateur à partir d'un écran d'interrogation).

Si vous ne souhaitez pas d'informations de contexte associées à votre message, utilisez l'option MQPMO_NO_CONTEXT. Lors de l'insertion d'un message sans contexte, les vérifications de droits effectuées par IBM MQ sont effectuées à l'aide d'un ID utilisateur vide. Un ID utilisateur vide ne peut pas être affecté à des droits explicites sur les ressources IBM MQ , mais il est traité comme un membre du groupe spécial'personne'. Pour plus de détails sur le groupe spécial nobody, voir Informations de référence sur l'interface des services installables.

Vous pouvez définir le contexte à l'aide de MQOPEN suivi de MQPUT à l'aide de l'option MQOOO_ et de l'option MQPMO_ indiquées dans les sections suivantes. Vous pouvez également définir le contexte à l'aide d'un seul élément MQPUT1, auquel cas il vous suffit de sélectionner l'option MQPMO_ indiquée dans les sections ci-dessous.

Les sections suivantes de cette rubrique expliquent l'utilisation du contexte d'identité, du contexte utilisateur et de tous les contextes.

- «Transmission du contexte d'identité», à la page 779
- «Transmission du contexte utilisateur», à la page 780
- «Transmission de tous les contextes», à la page 780
- «Définition du contexte d'identité», à la page 780
- «Définition du contexte utilisateur», à la page 780
- «Définition de tous les contextes», à la page 781

Transmission du contexte d'identité

En général, les programmes doivent transmettre des informations de contexte d'identité de message à message autour d'une application jusqu'à ce que les données atteignent leur destination finale.

Les programmes doivent modifier les informations de contexte d'origine chaque fois qu'ils modifient les données. Toutefois, les applications qui souhaitent modifier ou définir des informations de contexte doivent disposer du niveau de droits approprié. Le gestionnaire de files d'attente vérifie ce droit lorsque les applications ouvrent les files d'attente ; il doit disposer du droit d'utiliser les options de contexte appropriées pour l'appel MQOPEN.

Si votre application obtient un message, traite les données du message, puis place les données modifiées dans un autre message (éventuellement pour traitement par une autre application), l'application doit transmettre les informations de contexte d'identité du message d'origine au nouveau message. Vous pouvez autoriser le gestionnaire de files d'attente à créer les informations de contexte d'origine.

Pour sauvegarder les informations de contexte du message d'origine, utilisez l'option MQOO_SAVE_ALL_CONTEXT lorsque vous ouvrez la file d'attente pour obtenir le message. Cela s'ajoute

aux autres options que vous utilisez avec l'appel MQOPEN. Notez toutefois que vous ne pouvez pas sauvegarder les informations de contexte si vous ne parcourez que le message.

Lorsque vous créez le deuxième message:

- Ouvrez la file d'attente à l'aide de l'option MQOO_PASS_IDENTITY_CONTEXT (en plus de l'option MQOO_OUTPUT).
- Dans la zone *Context* de la structure des options d'insertion de message, indiquez le descripteur de la file d'attente à partir de laquelle vous avez sauvegardé les informations de contexte.
- Dans la zone *Options* de la structure des options d'insertion de message, spécifiez l'option MQPMO_PASS_IDENTITY_CONTEXT.

Transmission du contexte utilisateur

Vous ne pouvez pas choisir de transmettre uniquement le contexte utilisateur. Pour transmettre le contexte utilisateur lors de l'insertion d'un message, spécifiez MQPMO_PASS_ALL_CONTEXT. Toutes les propriétés du contexte utilisateur sont transmises de la même manière que le contexte d'origine.

Lorsqu'une instruction MQPUT ou MQPUT1 est exécutée et que le contexte est transmis, toutes les propriétés du contexte utilisateur sont transmises du message extrait au message inséré. Toutes les propriétés de contexte utilisateur modifiées par l'application d'insertion sont placées avec leurs valeurs d'origine. Toutes les propriétés de contexte utilisateur supprimées par l'application d'insertion sont restaurées dans le message d'insertion. Toutes les propriétés de contexte utilisateur que l'application d'insertion a ajoutées au message sont conservées.

Transmission de tous les contextes

Si votre application reçoit un message et place les données du message (non modifiées) dans un autre message, elle doit transmettre toutes les informations de contexte (identité, origine et utilisateur) du message d'origine au nouveau message. Un exemple d'application qui peut effectuer cette opération est un dispositif de transfert de messages, qui déplace les messages d'une file d'attente à une autre.

Suivez la même procédure que pour la transmission du contexte d'identité, sauf que vous utilisez l'option MQOPEN MQOO_PASS_ALL_CONTEXT et l'option d'insertion de message MQPMO_PASS_ALL_CONTEXT.

Définition du contexte d'identité

Si vous souhaitez définir les informations de contexte d'identité pour un message:

- Ouvrez la file d'attente à l'aide de l'option MQOO_SET_IDENTITY_CONTEXT.
- Placez le message dans la file d'attente en spécifiant l'option MQPMO_SET_IDENTITY_CONTEXT. Dans le descripteur de message, indiquez les informations de contexte d'identité dont vous avez besoin.

Remarque : Lorsque vous définissez certaines (mais pas toutes) des zones de contexte d'identité à l'aide des options MQOO_SET_IDENTITY_CONTEXT et MQPMO_SET_IDENTITY_CONTEXT, il est important de savoir que le gestionnaire de files d'attente ne définit aucune des autres zones.

Pour modifier l'une des options de contexte de message, vous devez disposer des autorisations appropriées pour émettre l'appel. Par exemple, pour utiliser MQOO_SET_IDENTITY_CONTEXT ou MQPMO_SET_IDENTITY_CONTEXT, vous devez disposer du droit `+setid` .

Définition du contexte utilisateur

Pour définir une propriété dans le contexte utilisateur, définissez la zone Contexte du descripteur de propriété de message (MQPD) sur MQPD_USER_CONTEXT lorsque vous effectuez l'appel MQSETMP.

Vous n'avez pas besoin de droits spéciaux pour définir une propriété dans le contexte utilisateur. Le contexte utilisateur n'a pas d'options de contexte MQOO_SET_ * ou MQPMO_SET_ *.

Définition de tous les contextes

Si vous souhaitez définir à la fois les informations d'identité et de contexte d'origine pour un message:

1. Ouvrez la file d'attente à l'aide de l'option MQOO_SET_ALL_CONTEXT.
2. Placez le message dans la file d'attente en spécifiant l'option MQPMO_SET_ALL_CONTEXT. Dans le descripteur de message, indiquez les informations de contexte d'identité et d'origine dont vous avez besoin.

Des droits appropriés sont nécessaires pour chaque type de paramètre de contexte.

Concepts associés

«Contexte de message», à la page 49
Les informations de *contexte de message* permettent à l'application qui extrait le message de trouver l'émetteur du message.

Référence associée

«Options MQOPEN relatives au contexte de message», à la page 769
Si vous souhaitez pouvoir associer des informations de contexte à un message lorsque vous le placez dans une file d'attente, vous devez utiliser l'une des options de contexte de message lorsque vous ouvrez la file d'attente.

Insertion d'un message dans une file d'attente à l'aide de l'appel MQPUT1

Utilisez l'appel MQPUT1 lorsque vous souhaitez fermer la file d'attente immédiatement après avoir inséré un message unique dessus. Par exemple, une application serveur est susceptible d'utiliser l'appel MQPUT1 lorsqu'elle envoie une réponse à chacune des différentes files d'attente.

MQPUT1 est fonctionnellement équivalent à l'appel de MQOPEN suivi de MQPUT, suivi de MQCLOSE. La seule différence dans la syntaxe des appels MQPUT et MQPUT1 est que pour MQPUT, vous spécifiez un descripteur d'objet, tandis que pour MQPUT1 , vous spécifiez une structure de descripteur d'objet (MQOD) comme définie dans MQOPEN (voir «Identification des objets (structure MQOD)», à la page 762). En effet, vous devez fournir des informations à l'appel MQPUT1 sur la file d'attente qu'il doit ouvrir, alors que lorsque vous appelez MQPUT, la file d'attente doit déjà être ouverte.

En tant qu'entrée de l'appel MQPUT1 , vous devez fournir:

- Un descripteur de connexion.
- Description de l'objet à ouvrir. Il se présente sous la forme d'une structure de descripteur d'objet (MQOD).
- Description du message que vous souhaitez placer dans la file d'attente. Il s'agit d'une structure de descripteur de message (MQMD).
- Informations de contrôle sous la forme d'une structure d'options d'insertion de message (MQPMO).
- Longueur des données contenues dans le message (MQLONG).
- Adresse des données de message.

La sortie de MQPUT1 est la suivante:

- Un code achèvement
- Un code anomalie

Si l'appel aboutit, il renvoie également votre structure d'options et votre structure de descripteur de message. L'appel modifie votre structure d'options pour afficher le nom de la file d'attente et le gestionnaire de files d'attente auquel le message a été envoyé. Si vous demandez que le gestionnaire de files d'attente génère une valeur unique pour l'identificateur du message que vous insérez (en spécifiant un zéro binaire dans la zone *MsgId* de la structure MQMD), l'appel insère la valeur dans la zone *MsgId* avant de vous renvoyer cette structure.

Remarque : Vous ne pouvez pas utiliser MQPUT1 avec un nom de file d'attente modèle ; toutefois, une fois qu'une file d'attente modèle a été ouverte, vous pouvez émettre une commande MQPUT1 vers la file d'attente dynamique.

Les six paramètres d'entrée de MQPUT1 sont les suivants:

Hconn
Il s'agit d'un descripteur de connexion. Pour les applications CICS , vous pouvez spécifier la constante MQHC_DEF_HCONN (qui a la valeur zéro) ou utiliser le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX. Pour les autres programmes, utilisez toujours le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX.

ObjDesc
Il s'agit d'une structure de descripteur d'objet (MQOD).

Dans les zones *ObjectName* et *ObjectQMgrName* , indiquez le nom de la file d'attente dans laquelle vous souhaitez insérer un message et le nom du gestionnaire de files d'attente propriétaire de cette file d'attente.

La zone *DynamicQName* est ignorée pour l'appel MQPUT1 car elle ne peut pas utiliser les files d'attente modèles.

Utilisez la zone *AlternateUserId* si vous souhaitez désigner un autre ID utilisateur à utiliser pour tester les droits d'ouverture de la file d'attente.

MsgDesc
Il s'agit d'une structure de descripteur de message (MQMD). Comme pour l'appel MQPUT, utilisez cette structure pour définir le message que vous mettez dans la file d'attente.

PutMsgOpts
Il s'agit d'une structure d'options d'insertion de message (MQPMO). Utilisez-la comme vous le feriez pour l'appel MQPUT (voir «Spécification d'options à l'aide de la structure MQPMO», à la page 774).

Lorsque la zone *Options* est définie sur zéro, le gestionnaire de files d'attente utilise votre propre ID utilisateur lorsqu'il effectue des tests de droits d'accès à la file d'attente. En outre, le gestionnaire de files d'attente ignore tout autre ID utilisateur indiqué dans la zone *AlternateUserId* de la structure MQOD.

BufferLength
Il s'agit de la longueur de votre message.

Buffer
Il s'agit de la zone tampon qui contient le texte de votre message.

Lorsque vous utilisez des clusters, MQPUT1 fonctionne comme si MQOO_BIND_NOT_FIXED était actif. Les applications doivent utiliser les zones résolues dans la structure MQPMO plutôt que la structure MQOD pour déterminer où le message a été envoyé. Pour plus d'informations, voir Configuration d'un cluster de gestionnaires de files d'attente .

Une description de l'appel MQPUT1 est fournie dans MQPUT1.

Multi *Listes de diffusion*

Sous IBM MQ for Multiplatforms, les listes de distribution permettent d'insérer un message vers plusieurs destinations dans un seul appel MQPUT ou MQPUT1 . Un seul appel MQOPEN peut ouvrir plusieurs files d'attente et un seul appel MQPUT peut ensuite insérer un message dans chacune de ces files d'attente. Certaines informations génériques provenant des structures MQI utilisées pour ce processus peuvent être remplacées par des informations spécifiques relatives aux destinations individuelles incluses dans la liste de distribution.

Avertissement : Les listes de distribution ne prennent pas en charge les files d'attente alias qui pointent vers des objets de rubrique. Si une file d'attente alias renvoie vers un objet de rubrique dans une liste de distribution, IBM MQ renvoie MQRC_ALIAS_BASE_Q_TYPE_ERROR.

Lorsqu'un appel MQOPEN est émis, des informations génériques sont extraites du descripteur d'objet (MQOD). Si vous spécifiez MQOD_VERSION_2 dans la zone *Version* et une valeur supérieure à zéro dans la zone *RecsPresent* , *Hobj* peut être défini comme un descripteur d'une liste (d'une ou de plusieurs files d'attente) plutôt que d'une file d'attente. Dans ce cas, des informations spécifiques sont fournies via les enregistrements d'objet (MQOR), qui fournissent des détails sur la destination (c'est-à-dire, *ObjectName* et *ObjectQMgrName*).

Le descripteur d'objet (*Hobj*) est transmis à l'appel MQPUT, ce qui vous permet d'insérer dans une liste plutôt que dans une file d'attente unique.

Lorsqu'un message est inséré dans les files d'attente (MQPUT), des informations génériques sont extraites de la structure d'option d'insertion de message (MQPMO) et du descripteur de message (MQMD). Des informations spécifiques sont fournies sous la forme d'enregistrements de message d'insertion (MQPMR).

Les enregistrements de réponse (MQRR) peuvent recevoir un code achèvement et un code anomalie spécifiques à chaque file d'attente de destination.

La Figure 56, à la page 783 montre le fonctionnement des listes de distribution.

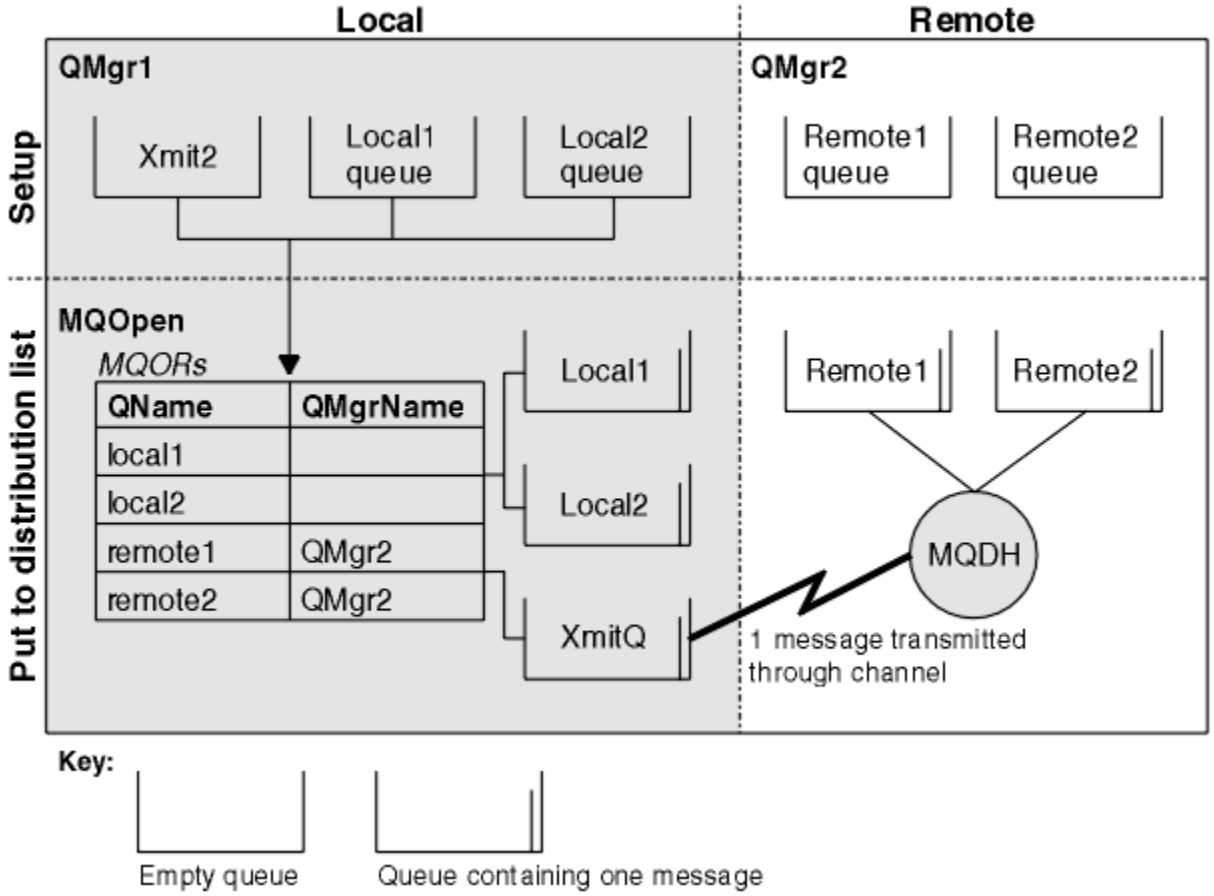


Figure 56. Fonctionnement des listes de distribution

Ouverture des listes de distribution
Utilisez l'appel MQOPEN pour ouvrir une liste de distribution et utilisez les options de l'appel pour indiquer ce que vous souhaitez faire avec la liste.

En tant qu'entrée de MQOPEN, vous devez fournir:

- Un descripteur de connexion (voir «Insertion de messages dans une file d'attente», à la page 772 pour une description)
- Informations génériques dans la structure de descripteur d'objet (MQOD)
- Nom de chaque file d'attente à ouvrir, à l'aide de la structure d'enregistrement d'objet (MQOR)

La sortie de MQOPEN est la suivante:

- Un descripteur d'objet qui représente votre accès à la liste de distribution
- Un code achèvement générique
- Code anomalie générique

- Enregistrements de réponse (facultatif), contenant un code achèvement et la raison de chaque destination

Utilisation de la structure MQOD

Utilisez la structure MQOD pour identifier les files d'attente que vous souhaitez ouvrir.

Pour définir une liste de distribution, vous devez spécifier MQOD_VERSION_2 dans la zone *Version* , une valeur supérieure à zéro dans la zone *RecsPresent* et MQOT_Q dans la zone *ObjectType* . Voir MQOD pour une description de toutes les zones de la structure MQOD.

Utilisation de la structure MQOR

Indiquez une structure MQOR pour chaque destination.

La structure contient les noms de file d'attente de destination et de gestionnaire de files d'attente. Les zones *ObjectName* et *ObjectQMgrName* du MQOD ne sont pas utilisées pour les listes de distribution. Il doit y avoir un ou plusieurs enregistrements d'objet. Si *ObjectQMgrName* est laissé vide, le gestionnaire de files d'attente local est utilisé. Pour plus d'informations sur ces zones, voir ObjectName et ObjectQMgrName .

Vous pouvez spécifier les files d'attente de destination de deux manières:

- En utilisant la zone de décalage *ObjectRecOffset*.

  Dans ce cas, l'application doit déclarer sa propre structure contenant une structure MQOD, suivie du tableau d'enregistrements MQOR (avec autant d'éléments de tableau que nécessaire), et définir *ObjectRecOffset* sur le décalage du premier élément du tableau à partir du début du MQOD. Vérifiez que ce décalage est correct.

  L'utilisation des fonctions intégrées fournies par le langage de programmation est recommandée, si elles sont disponibles dans tous les environnements dans lesquels l'application s'exécute. Le code suivant illustre cette technique pour le langage de programmation 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.
```

  Vous pouvez également utiliser la constante MQOD_CURRENT_LENGTH si le langage de programmation ne prend pas en charge les fonctions intégrées nécessaires dans tous les environnements concernés. Le code suivant illustre cette technique:

```
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.
```

  Toutefois, cela ne fonctionne correctement que si la structure MQOD et le tableau d'enregistrements MQOR sont contigus ; si le compilateur insère des octets entre le MQOD et le tableau MQOR, ils doivent être ajoutés à la valeur stockée dans *ObjectRecOffset*.

  L'utilisation de *ObjectRecOffset* est recommandée pour les langages de programmation qui ne prennent pas en charge le type de données de pointeur ou qui implémentent le type de données de pointeur d'une manière qui n'est pas portable dans des environnements différents (par exemple, le langage de programmation COBOL).

- En utilisant la zone de pointeur *ObjectRecPtr*.

Dans ce cas, l'application peut déclarer le tableau de structures MQOR séparément de la structure MQOD et définir *ObjectRecPtr* sur l'adresse du tableau. Le code suivant illustre cette technique pour le langage de programmation C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

L'utilisation de *ObjectRecPtr* est recommandée pour les langages de programmation qui prennent en charge le type de données de pointeur d'une manière portable dans différents environnements (par exemple, le langage de programmation C).

Quelle que soit la technique choisie, vous devez utiliser *ObjectRecOffset* et *ObjectRecPtr* ; l'appel échoue avec le code anomalie MQRC_OBJECT_RECORDS_ERROR si les deux valeurs sont nulles, ou les deux sont différentes de zéro.

Utilisation de la structure MQRR

Ces structures sont spécifiques à la destination ; chaque enregistrement de réponse contient une zone *CompCode* et *Reason* pour chaque file d'attente d'une liste de distribution. Vous devez utiliser cette structure pour vous permettre de distinguer les éventuels problèmes.

Par exemple, si vous recevez le code anomalie MQRC_MULTIPLE_MOTIFS et que votre liste de distribution contient cinq files d'attente de destination, vous ne saurez pas à quelles files d'attente les problèmes s'appliquent si vous n'utilisez pas cette structure. Toutefois, si vous disposez d'un code achèvement et d'un code raison pour chaque destination, vous pouvez localiser les erreurs plus facilement.

Pour plus d'informations sur la structure MQRR, voir MQRR .

La Figure 57, à la page 785 montre comment ouvrir une liste de distribution dans C.

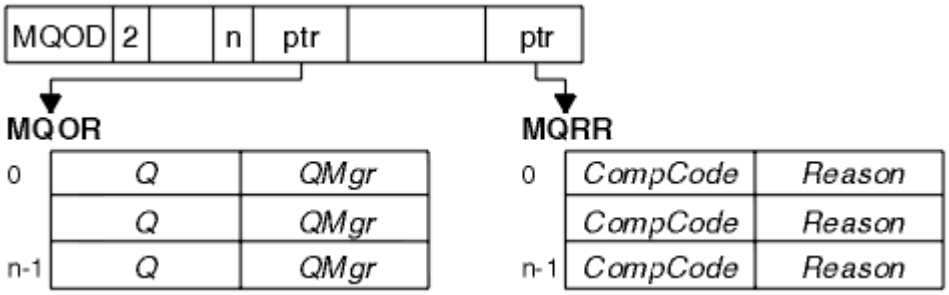


Figure 57. Ouverture d'une liste de distribution en C

La Figure 58, à la page 785 montre comment ouvrir une liste de distribution en COBOL.

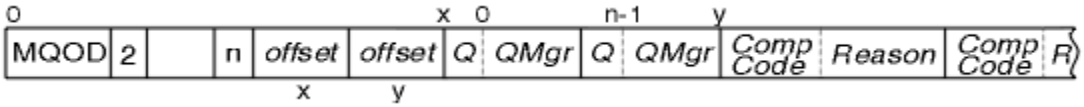


Figure 58. Ouverture d'une liste de distribution en COBOL

Utilisation des options MQOPEN

Vous pouvez spécifier les options suivantes lors de l'ouverture d'une liste de distribution:

- MQOO_SORTIE
- MQOO_FAIL_IF_QUIESCING (facultatif)
- MQOO_ALTERNATE_USER_AUTHORITY (facultatif)
- MQOO_ * _CONTEXT (facultatif)

Pour une description de ces options, voir «Ouverture et fermeture d'objets», à la page 760 .

Insertion de messages dans une liste de distribution
Pour insérer des messages dans une liste de distribution, vous pouvez utiliser MQPUT ou MQPUT1.

En tant qu'entrée, vous devez fournir:

- Un descripteur de connexion (voir «Insertion de messages dans une file d'attente», à la page 772 pour une description).
- Un descripteur d'objet. Si une liste de distribution est ouverte à l'aide de MQOPEN, *Hobj* vous permet uniquement d'insérer des données dans la liste.
- Une structure de descripteur de message (MQMD). Pour une description de cette structure, voir MQMD .
- Informations de contrôle sous la forme d'une structure d'option d'insertion de message (MQPMO). Pour plus d'informations sur le remplissage des zones de la structure MQPMO, voir «Spécification d'options à l'aide de la structure MQPMO», à la page 774 .
- Informations de contrôle sous forme d'enregistrements de message d'insertion (MQPMR).
- Longueur des données contenues dans le message (MQLONG).
- Les données de message elles-mêmes.

La sortie est la suivante :

- Un code achèvement
- Un code anomalie
- Enregistrements de réponse (facultatif)

Utilisation de la structure MQPMR

Cette structure est facultative et fournit des informations spécifiques à la destination pour certaines zones que vous pouvez identifier différemment de celles déjà identifiées dans le MQMD.

Pour obtenir une description de ces zones, voir MQPMR.

Le contenu de chaque enregistrement dépend des informations fournies dans la zone *PutMsgRecFields* du MQPMO. Par exemple, dans l'exemple de programme AMQSPTL0.C (voir «Exemple de programme Liste de distribution», à la page 1120 pour une description) illustrant l'utilisation des listes de distribution, l'exemple choisit de fournir des valeurs pour *MsgId* et *CorrelId* dans le MQPMR. Cette section de l'exemple de programme se présente comme suit:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

Cela implique que *MsgId* et *CorrelId* sont fournis pour chaque destination d'une liste de distribution. Les enregistrements de message d'insertion sont fournis sous la forme d'un tableau.

La Figure 59, à la page 786 montre comment placer un message dans une liste de distribution dans C.

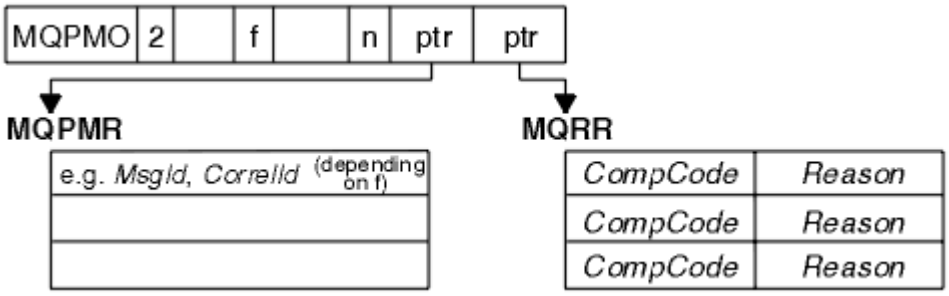


Figure 59. Insertion d'un message dans une liste de distribution en C

La Figure 60, à la page 787 montre comment placer un message dans une liste de distribution en COBOL.

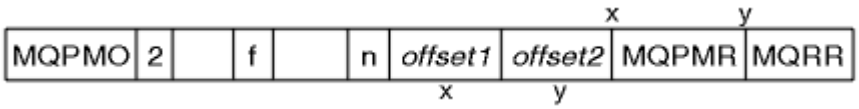


Figure 60. Insertion d'un message dans une liste de distribution en COBOL

Utilisation de MQPUT1

Si vous utilisez MQPUT1, tenez compte des points suivants:

1. Les valeurs des zones *ResponseRecOffset* et *ResponseRecPtr* doivent être nulles ou nulles.
2. Les enregistrements de réponse, si nécessaire, doivent être traités à partir du MQOD.

Certains cas où les appels d'insertion échouent

Si certains attributs d'une file d'attente sont modifiés à l'aide de l'option FORCE sur une commande pendant l'intervalle entre l'émission d'un appel MQOPEN et d'un appel MQPUT, l'appel MQPUT échoue et renvoie le code anomalie MQRC_OBJECT_CHANGED.

Le gestionnaire de files d'attente marque le descripteur d'objet comme n'étant plus valide. Cela se produit également si les modifications sont apportées lors du traitement d'un appel MQPUT1 ou si les modifications s'appliquent à une file d'attente dans laquelle le nom de la file d'attente est résolu. Les attributs qui affectent le descripteur de cette manière sont répertoriés dans la description de l'appel MQOPEN dans MQOPEN. Si votre appel renvoie le code anomalie MQRC_OBJECT_CHANGED, fermez la file d'attente, rouvrez-la, puis essayez d'insérer un message à nouveau.

Si les opérations d'insertion sont interdites pour une file d'attente dans laquelle vous tentez d'insérer des messages (ou toute file d'attente dans laquelle le nom de la file d'attente est résolu), l'appel MQPUT ou MQPUT1 échoue et renvoie le code anomalie MQRC_PUT_INHIBÉ. Vous pouvez insérer un message correctement si vous tentez d'appeler ultérieurement, si la conception de l'application est telle que d'autres programmes modifient régulièrement les attributs des files d'attente.

Plus loin, si la file d'attente dans laquelle vous tentez d'insérer votre message est saturée, l'appel MQPUT ou MQPUT1 échoue et renvoie MQRC_Q_FULL.

Si une file d'attente dynamique (temporaire ou permanente) a été supprimée, les appels MQPUT utilisant un descripteur d'objet acquis précédemment échouent et renvoient le code anomalie MQRC_Q_DELETED. Dans ce cas, il est recommandé de fermer la poignée de l'objet car elle n'est plus utile pour vous.

Dans le cas des listes de distribution, plusieurs codes achèvement et codes raison peuvent apparaître dans une même demande. Ils ne peuvent pas être traités uniquement à l'aide des zones de sortie *CompCode* et *Reason* sur MQOPEN et MQPUT.

Lorsque vous utilisez des listes de distribution pour placer des messages vers plusieurs destinations, les enregistrements de réponse contiennent les *CompCode* et *Reason* spécifiques à chaque destination. Si vous recevez le code achèvement MQCC_FAILED, aucun message n'est inséré dans une file d'attente de destination. Si le code achèvement est MQCC_WARNING, le message est inséré avec succès dans une ou plusieurs files d'attente de destination. Si vous recevez le code retour MQRC_MULTIPLE_MOTIFS, les codes anomalie ne sont pas tous identiques pour chaque destination. Par conséquent, il est recommandé d'utiliser la structure MQRR afin de déterminer la ou les files d'attente à l'origine de l'erreur et les raisons de chacune d'elles.

Obtention de messages à partir d'une file d'attente

Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

Vous pouvez extraire des messages d'une file d'attente de deux manières:

1. Vous pouvez supprimer un message de la file d'attente afin que d'autres programmes ne puissent plus le voir.

2. Vous pouvez copier un message en laissant le message d'origine dans la file d'attente. C'est ce que l'on appelle la *navigation*. Vous pouvez supprimer le message une fois que vous l'avez parcouru.

Dans les deux cas, vous utilisez l'appel MQGET, mais votre application doit d'abord être connectée au gestionnaire de files d'attente et vous devez utiliser l'appel MQOPEN pour ouvrir la file d'attente (pour l'entrée, l'exploration ou les deux). Ces opérations sont décrites dans «Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753 et «Ouverture et fermeture d'objets», à la page 760.

Une fois que vous avez ouvert la file d'attente, vous pouvez utiliser l'appel MQGET à plusieurs reprises pour parcourir ou supprimer des messages dans la même file d'attente. Appelez MQCLOSE lorsque vous avez terminé d'extraire tous les messages que vous souhaitez de la file d'attente.

Utilisez les liens suivants pour en savoir plus sur l'obtention de messages à partir d'une file d'attente:

- «Obtention de messages à partir d'une file d'attente à l'aide de l'appel MQGET», à la page 789
- «Ordre dans lequel les messages sont extraits d'une file d'attente», à la page 793
- «Obtention d'un message particulier», à la page 805
- «Amélioration des performances des messages non persistants», à la page 806
- z/OS «Type of index», à la page 811
- «Traitement des messages d'une longueur supérieure à 4 Mo», à la page 811
- «En attente de messages», à la page 817
- z/OS «Signaling», à la page 818
- z/OS «Annulation ignorée», à la page 820
- «Conversion des données d'application», à la page 822
- «Recherche de messages dans une file d'attente», à la page 823
- «Certains cas où l'appel MQGET échoue», à la page 829

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72

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

Obtention de messages à partir d'une file d'attente à l'aide de l'appel MQGET

L'appel MQGET extrait un message d'une file d'attente locale ouverte. Il ne peut pas extraire de message d'une file d'attente sur un autre système.

En tant qu'entrée de l'appel MQGET, vous devez fournir:

- Un descripteur de connexion.
- Un descripteur de file d'attente.
- Description du message à extraire de la file d'attente. Il s'agit d'une structure de descripteur de message (MQMD).
- Informations de contrôle sous la forme d'une structure MQGMO (Get Message Options).
- Taille de la mémoire tampon que vous avez affectée pour contenir le message (MQLONG).
- Adresse de la mémoire dans laquelle le message doit être inséré.

La sortie de MQGET est la suivante:

- Un code anomalie
- Un code achèvement
- Le message dans la zone de mémoire tampon que vous avez spécifiée, si l'appel aboutit
- Votre structure d'options, modifiée pour afficher le nom de la file d'attente à partir de laquelle le message a été extrait
- Votre structure de descripteur de message, avec le contenu des zones modifiées pour décrire le message qui a été extrait
- Longueur du message (MQLONG)

Il existe une description de l'appel MQGET dans MQGET.

Les sections suivantes décrivent les informations que vous devez fournir en entrée de l'appel MQGET.

- «Spécification des descripteurs de connexion», à la page 789
- «Description des messages à l'aide de la structure MQMD et de l'appel MQGET», à la page 789
- «Spécification des options MQGET à l'aide de la structure MQGMO», à la page 790
- «Spécification de la taille de la zone tampon», à la page 792

Spécification des descripteurs de connexion

z/OS Pour les applications CICS on z/OS , vous pouvez spécifier la constante MQHC_DEF_HCONN (qui a la valeur zéro) ou utiliser le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX. Pour les autres applications, utilisez toujours le descripteur de connexion renvoyé par l'appel MQCONN ou MQCONNX.

Utilisez le descripteur de file d'attente (*Hobj*) renvoyé lorsque vous appelez MQOPEN.

Description des messages à l'aide de la structure MQMD et de l'appel MQGET

Pour identifier le message que vous souhaitez extraire d'une file d'attente, utilisez la structure de descripteur de message (MQMD).

Il s'agit d'un paramètre d'entrée-sortie pour l'appel MQGET. Il existe une introduction aux propriétés de message que MQMD décrit dans «Messages IBM MQ», à la page 19et une description de la structure elle-même dans MQMD.

Si vous savez quel message vous souhaitez obtenir de la file d'attente, voir «Obtention d'un message particulier», à la page 805.

Si vous ne spécifiez pas de message particulier, MQGET extrait le *premier* message de la file d'attente. «Ordre dans lequel les messages sont extraits d'une file d'attente», à la page 793 décrit comment la priorité d'un message, l'attribut **MsgDeliverySequence** de la file d'attente et l'option MQGMO_LOGICAL_ORDER déterminent l'ordre des messages dans la file d'attente.

Remarque : Si vous souhaitez utiliser MQGET plusieurs fois (par exemple, pour parcourir les messages de la file d'attente), vous devez définir les zones *MsgId* et *CorrelId* de cette structure sur null après chaque appel. Cette opération efface ces zones des identificateurs du message qui a été extrait.

Toutefois, si vous souhaitez regrouper vos messages, le *GroupId* doit être le même pour les messages du même groupe, de sorte que l'appel recherche un message ayant les mêmes identificateurs que le message précédent afin de constituer l'ensemble du groupe.

Spécification des options MQGET à l'aide de la structure MQGMO

La structure MQGMO est une variable d'entrée-sortie permettant de transmettre des options à l'appel MQGET. Les sections suivantes vous aident à renseigner certaines des zones de cette structure.

Il existe une description de la structure MQGMO dans MQGMO.

StrucId
StrucId est une zone de 4 caractères utilisée pour identifier la structure en tant que structure d'options d'extraction de message. Indiquez toujours MQGMO_STRUC_ID.

Version
*Version*décrit le numéro de version de la structure. MQGMO_VERSION_1 est la valeur par défaut. Si vous souhaitez utiliser les zones de la version 2 ou extraire des messages dans l'ordre logique, indiquez MQGMO_VERSION_2. Si vous souhaitez utiliser les zones de la version 3 ou extraire des messages dans l'ordre logique, spécifiez MQGMO_VERSION_3. MQGMO_CURRENT_VERSION permet à votre application d'utiliser le niveau le plus récent.

Options
Dans votre code, vous pouvez sélectionner les options dans n'importe quel ordre ; chaque option est représentée par un bit dans la zone *Options* .

La zone *Options* contrôle:

- Indique si l'appel MQGET attend qu'un message arrive dans la file d'attente avant de se terminer (voir «En attente de messages», à la page 817)
- Indique si l'opération d'extraction est incluse dans une unité de travail.
- Indique si un message non persistant est extrait en dehors du point de synchronisation, ce qui permet une messagerie rapide
- z/OS Sous IBM MQ for z/OS, indique si le message extrait est marqué comme ignoré lors de l'annulation (voir «Annulation ignorée», à la page 820)
- Indique si le message est supprimé de la file d'attente ou simplement consulté
- Indique si un message doit être sélectionné à l'aide d'un curseur de navigation ou selon d'autres critères de sélection
- Indique si l'appel aboutit même si le message est plus long que votre mémoire tampon
- z/OS Sous IBM MQ for z/OS, indique s'il faut autoriser l'appel à se terminer. Cette option définit également un signal pour indiquer que vous souhaitez être averti lorsqu'un message arrive
- Indique si l'appel échoue si le gestionnaire de files d'attente est à l'état de mise au repos
- z/OS Sous IBM MQ for z/OS, indique si l'appel échoue si la connexion est à l'état de mise au repos
- Indique si la conversion des données de message d'application est requise (voir «Conversion des données d'application», à la page 822)
- Ordre dans lequel les messages et les segments sont extraits d'une file d'attente z/OS (sauf pour IBM MQ for z/OS)

- Indique si les messages logiques complets uniquement peuvent être extraits z/OS (sauf pour IBM MQ for z/OS)
- Indique si les messages d'un groupe peuvent être extraits uniquement lorsque *tous* les messages du groupe sont disponibles
- Indique si les segments d'un message logique peuvent être extraits uniquement lorsque *tous* les segments du message logique sont disponibles z/OS (sauf IBM MQ for z/OS)

Si vous laissez la zone *Options* définie sur la valeur par défaut (MQGMO_NO_WAIT), l'appel MQGET fonctionne comme suit:

- Si aucun message ne correspond à vos critères de sélection dans la file d'attente, l'appel n'attend pas l'arrivée d'un message, mais se termine immédiatement. z/OS En outre, dans IBM MQ for z/OS, l'appel ne définit pas de signal de demande de notification lorsqu'un tel message arrive.
- La façon dont l'appel fonctionne avec les points de synchronisation est déterminée par la plateforme:

Plateforme	**Sous contrôle de point de synchronisation**
IBM i	Non
Systèmes AIX and Linux	Non
z/OS z/OS z/OS	Oui
Systèmes Windows	Non

- z/OS Sous IBM MQ for z/OS, le message extrait n'est pas marqué comme ignorant l'annulation.
- Le message sélectionné est supprimé de la file d'attente (non consulté).
- Aucune conversion de données de message d'application n'est requise.
- L'appel échoue si le message est plus long que votre mémoire tampon.

WaitInterval
La zone *WaitInterval* indique la durée maximale (en millisecondes) pendant laquelle l'appel MQGET attend qu'un message arrive dans la file d'attente lorsque vous utilisez l'option MQGMO_WAIT. Si aucun message n'arrive dans le délai spécifié dans *WaitInterval*, l'appel se termine et renvoie un code anomalie indiquant qu'aucun message ne correspond à vos critères de sélection dans la file d'attente.

z/OS Sous IBM MQ for z/OS, si vous utilisez l'option MQGMO_SET_SIGNAL, la zone *WaitInterval* indique l'heure pour laquelle le signal est défini.

Pour plus d'informations concernant ces options, consultez «En attente de messages», à la page 817 z/OS et «Signaling», à la page 818.

z/OS ***Signal1***
Signal1 est pris en charge uniquement sous IBM MQ for z/OS.

Si vous utilisez l'option MQGMO_SET_SIGNAL pour demander que votre application soit avertie lorsqu'un message approprié arrive, vous spécifiez le type de signal dans la zone *Signal1* . Dans IBM MQ sur toutes les autres plateformes, la zone *Signal1* est réservée et sa valeur n'est pas significative.

z/OS Pour plus d'informations, voir «Signaling», à la page 818.

Signal2
La zone *Signal2* est réservée sur toutes les plateformes et sa valeur n'est pas significative.

z/OS Pour plus d'informations, voir «Signaling», à la page 818.

ResolvedQName
ResolvedQName est une zone de sortie dans laquelle le gestionnaire de files d'attente renvoie le nom de la file d'attente (après résolution de tout alias) à partir de laquelle le message a été extrait.

MatchOptions
MatchOptions contrôle les critères de sélection pour MQGET.

GroupStatus
GroupStatus indique si le message que vous avez extrait se trouve dans un groupe.

SegmentStatus
SegmentStatus indique si l'élément que vous avez extrait est un segment d'un message logique.

Segmentation
Segmentation indique si la segmentation est autorisée pour le message extrait.

MsgToken

MsgToken Identifie un message de manière unique.

ReturnedLength
ReturnedLength est une zone de sortie dans laquelle le gestionnaire de files d'attente renvoie la longueur des données de message renvoyées (en octets).

MsgHandle
Descripteur d'un message à remplir avec les propriétés du message extrait de la file d'attente. Le descripteur a déjà été créé par un appel MQCRTMH. Toutes les propriétés déjà associées à l'identificateur sont effacées avant l'extraction d'un message.

Spécification de la taille de la zone tampon

Dans le paramètre **BufferLength** de l'appel MQGET, indiquez la taille de la zone tampon devant contenir les données de message que vous extrayez. Vous décidez de la taille de cette zone de trois manières:

1. Il se peut que vous connaissiez déjà la longueur des messages à attendre de ce programme. Si tel est le cas, indiquez une mémoire tampon de cette taille.

   Toutefois, vous pouvez utiliser l'option MQGMO_ACCEPT_TRUNCATED_MSG dans la structure MQGMO si vous souhaitez que l'appel MQGET se termine même si le message est trop volumineux pour la mémoire tampon. Dans ce cas :

   - La mémoire tampon est remplie avec autant de messages qu'elle peut contenir
   - L'appel renvoie un code achèvement d'avertissement
   - Le message est supprimé de la file d'attente (suppression du reste du message) ou le curseur de navigation est avancé (si vous parcourez la file d'attente)
   - La longueur réelle du message est renvoyée dans *DataLength*

   Sans cette option, l'appel se termine toujours avec un avertissement, mais il ne supprime pas le message de la file d'attente (ou n'avance pas le curseur de navigation).
2. Estimez une taille pour la mémoire tampon (ou spécifiez même une taille de zéro octet) et *n'utilisez pas* l'option MQGMO_ACCEPT_TRUNCATED_MSG. Si l'appel MQGET échoue (par exemple, parce que la mémoire tampon est trop petite), la longueur du message est renvoyée dans le paramètre **DataLength** de l'appel. (La mémoire tampon est toujours remplie avec autant de messages qu'elle peut contenir, mais le traitement de l'appel n'est pas terminé.) Stockez le *MsgId* de ce message, puis répétez l'appel MQGET en spécifiant une zone tampon de la taille correcte et le *MsgId* que vous avez noté lors du premier appel.

   Si votre programme sert une file d'attente qui est également servie par d'autres programmes, l'un de ces autres programmes peut supprimer le message que vous souhaitez avant que votre programme puisse émettre un autre appel MQGET. Votre programme peut perdre du temps à rechercher un message qui n'existe plus. Pour éviter cela, parcourez d'abord la file d'attente jusqu'à ce que vous

trouviez le message de votre choix, en spécifiant un *BufferLength* égal à zéro et en utilisant l'option MQGMO_ACCEPT_TRUNCATED_MSG. Le curseur de navigation est positionné sous le message de votre choix. Vous pouvez ensuite extraire le message en appelant à nouveau MQGET en spécifiant l'option MQGMO_MSG_UNDER_CURSOR. Si un autre programme supprime le message entre vos appels de recherche et de suppression, votre deuxième MQGET échoue immédiatement (sans effectuer de recherche dans l'ensemble de la file d'attente), car il n'y a pas de message sous votre curseur de recherche.

3. L'attribut *MaxMsgLength queue* détermine la longueur maximale des messages acceptés pour cette file d'attente ; l'attribut *MaxMsgLength queue manager* détermine la longueur maximale des messages acceptés pour ce gestionnaire de files d'attente. Si vous ne savez pas quelle longueur de message attendre, vous pouvez vous renseigner sur l'attribut **MaxMsgLength** (à l'aide de l'appel MQINQ), puis spécifier une mémoire tampon de cette taille.

   Essayez de rendre la taille de la mémoire tampon aussi proche que possible de la taille réelle des messages afin d'éviter des performances réduites.

   Pour plus d'informations sur l'attribut **MaxMsgLength** , voir «Augmentation de la longueur maximale des messages», à la page 812.

Ordre dans lequel les messages sont extraits d'une file d'attente

Vous pouvez contrôler l'ordre dans lequel vous extrayez les messages d'une file d'attente. Cette section examine les options.

Priorité

Un programme peut affecter une priorité à un message lorsqu'il le place dans une file d'attente (voir «Priorités des messages», à la page 27). Les messages de priorité égale sont stockés dans une file d'attente par ordre d'arrivée, et non par ordre de validation.

Le gestionnaire de files d'attente gère les files d'attente soit dans la séquence FIFO stricte (premier entré, premier sorti), soit dans la séquence FIFO dans la séquence de priorité. Cela dépend de la définition de l'attribut **MsgDeliverySequence** de la file d'attente. Lorsqu'un message arrive dans une file d'attente, il est inséré immédiatement après le dernier message ayant la même priorité.

Les programmes peuvent soit extraire le premier message d'une file d'attente, soit extraire un message particulier d'une file d'attente, en ignorant la priorité de ces messages. Par exemple, un programme peut vouloir traiter la réponse à un message particulier qu'il a envoyé précédemment. Pour plus d'informations, voir «Obtention d'un message particulier», à la page 805.

Si une application insère une séquence de messages dans une file d'attente, une autre application peut extraire ces messages dans l'ordre dans lequel ils ont été insérés, à condition que:

- Les messages ont tous la même priorité
- Les messages ont tous été placés dans la même unité de travail ou à l'extérieur d'une unité de travail
- La file d'attente est locale pour l'application d'insertion

Si ces conditions ne sont pas remplies, et que les applications dépendent des messages extraits dans un certain ordre, les applications doivent soit inclure des informations de séquencement dans les données de message, soit établir un moyen d'accuser réception d'un message avant l'envoi suivant.

z/OS Sous IBM MQ for z/OS, vous pouvez utiliser l'attribut de file d'attente *IndexType*pour augmenter la vitesse des opérations MQGET sur la file d'attente. Pour plus d'informations, voir «Type of index», à la page 811.

Ordre logique et physique

Dans chaque niveau de priorité, les messages des files d'attente peuvent apparaître dans l'ordre *physique* ou *logique* .

L'ordre physique est l'ordre dans lequel les messages arrivent dans une file d'attente. L'ordre logique est lorsque tous les messages et segments au sein d'un groupe sont dans leur séquence logique, les uns à

côté des autres, dans la position déterminée par la position physique du premier élément appartenant au groupe.

Pour une description des groupes, des messages et des segments, voir «Groupes de messages», à la page 46. Ces ordres physiques et logiques peuvent être différents pour les raisons suivantes:

- Les groupes peuvent arriver à une destination à des moments similaires à partir de différentes applications, perdant ainsi tout ordre physique distinct.
- Même au sein d'un même groupe, les messages peuvent être dans le désordre en raison d'un réacheminement ou d'un retard de certains des messages du groupe.

Par exemple, l'ordre logique peut ressembler à la figure Figure 61, à la page 794:

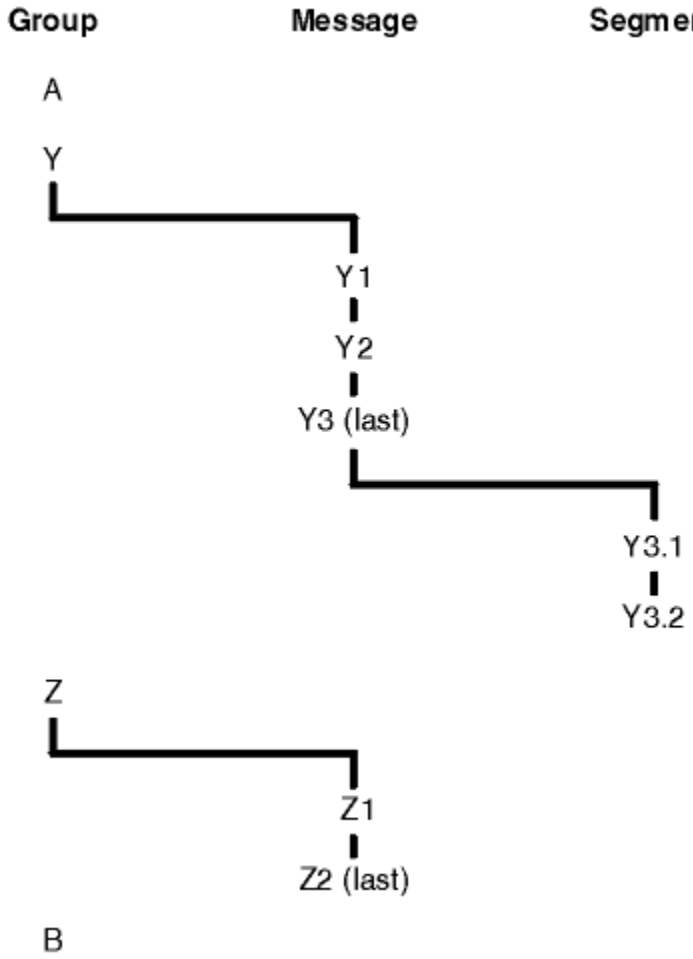


Figure 61. Ordre logique dans une file d'attente

Ces messages se produisent dans l'ordre logique suivant sur une file d'attente:

1. Message A (pas dans un groupe)
2. Message logique 1 du groupe Y
3. Message logique 2 du groupe Y
4. Segment 1 du (dernier) message logique 3 du groupe Y
5. (Dernier) segment 2 du (dernier) message logique 3 du groupe Y
6. Message logique 1 du groupe Z
7. (Dernier) message logique 2 du groupe Z
8. Message B (pas dans un groupe)

L'ordre physique, cependant, peut être tout à fait différent. La position physique du *premier* élément dans chaque groupe détermine la position logique de l'ensemble du groupe. Par exemple, si les groupes Y et Z arrivent à des heures similaires et que le message 2 du groupe Z dépasse le message 1 du même groupe, l'ordre physique ressemble à la figure Figure 62, à la page 795:

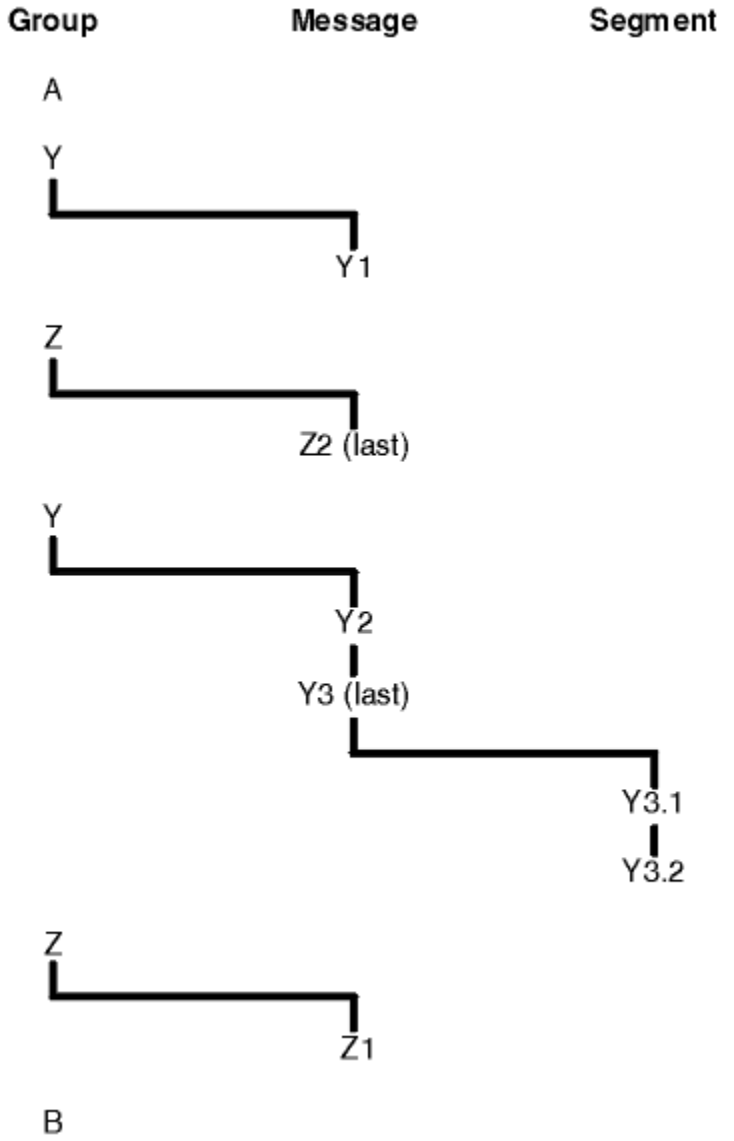


Figure 62. Ordre physique dans une file d'attente

Ces messages se produisent dans l'ordre physique suivant dans la file d'attente:

1. Message A (pas dans un groupe)
2. Message logique 1 du groupe Y
3. Message logique 2 du groupe Z
4. Message logique 2 du groupe Y
5. Segment 1 du (dernier) message logique 3 du groupe Y
6. (Dernier) segment 2 du (dernier) message logique 3 du groupe Y
7. Message logique 1 du groupe Z
8. Message B (pas dans un groupe)

Remarque : z/OS Sous IBM MQ for z/OS, l'ordre physique des messages dans la file d'attente n'est pas garanti si la file d'attente est indexée par GROUPID.

Lors de l'obtention de messages, vous pouvez spécifier MQGMO_LOGICAL_ORDER pour extraire les messages dans l'ordre logique plutôt que dans l'ordre physique.

Si vous émettez un appel MQGET avec MQGMO_BROWSE_FIRST et MQGMO_LOGICAL_ORDER, les appels MQGET suivants avec MQGMO_BROWSE_NEXT doivent également spécifier MQGMO_LOGICAL_ORDER. Inversement, si MQGET avec MQGMO_BROWSE_FIRST ne spécifie pas MQGMO_LOGICAL_ORDER, les MQGET suivants ne doivent pas non plus être associés à MQGMO_BROWSE_NEXT.

Les informations de groupe et de segment conservées par le gestionnaire de files d'attente pour les appels MQGET qui parcourent les messages de la file d'attente sont distinctes des informations de groupe et de segment conservées par le gestionnaire de files d'attente pour les appels MQGET qui suppriment des messages de la file d'attente. Lorsque vous spécifiez MQGMO_BROWSE_FIRST, le gestionnaire de files d'attente ignore les informations de groupe et de segment à parcourir et analyse la file d'attente comme s'il n'y avait pas de groupe en cours et de message logique en cours.

Remarque : N'utilisez pas d'appel MQGET pour parcourir *au-delà de la fin* d'un groupe de messages (ou d'un message logique ne figurant pas dans un groupe) sans spécifier MQGMO_LOGICAL_ORDER. Par exemple, si le dernier message du groupe *précède* le premier message du groupe dans la file d'attente, le fait d'utiliser MQGMO_BROWSE_NEXT pour parcourir le groupe au-delà de la fin, en spécifiant MQMO_MATCH_MSG_SEQ_NUMBER avec *MsgSeqNumber* défini sur 1 (pour trouver le premier message du groupe suivant) renvoie à nouveau le premier message du groupe déjà parcouru. Cela peut se produire immédiatement ou un certain nombre d'appels MQGET ultérieurement (s'il existe des groupes intermédiaires).

Evitez la possibilité d'une boucle sans fin en ouvrant la file d'attente *deux fois* pour la recherche:

- Utilisez le premier descripteur pour parcourir uniquement le premier message de chaque groupe.
- Utilisez le deuxième descripteur pour parcourir uniquement les messages d'un groupe spécifique.
- Utilisez les options MQMO_ * pour déplacer le deuxième curseur de navigation à la position du premier curseur de navigation, avant de parcourir les messages du groupe.
- N'utilisez pas la recherche MQGMO_BROWSE_NEXT au-delà de la fin d'un groupe.

Pour plus d'informations à ce sujet, voir MQGET, MQMDet Règles de validation des options MQI.

Pour la plupart des applications, vous choisirez probablement l'ordre logique ou physique lors de la navigation. Toutefois, si vous souhaitez passer d'un mode à l'autre, n'oubliez pas que lorsque vous émettez pour la première fois une commande de navigation avec MQGMO_LOGICAL_ORDER, votre position dans la séquence logique est établie.

Si le premier élément du groupe n'est pas présent à ce stade, le groupe dans lequel vous vous trouvez n'est pas considéré comme faisant partie de la séquence logique.

Une fois que le curseur de navigation se trouve dans un groupe, il peut continuer dans le même groupe, même si le premier message est supprimé. Toutefois, au départ, vous ne pouvez jamais vous déplacer dans un groupe à l'aide de MQGMO_LOGICAL_ORDER où le premier élément n'est pas présent.

MQPMO_LOGICAL_ORDER

L'option MQPMO indique au gestionnaire de files d'attente comment l'application insère des messages dans des groupes et des segments de messages logiques. Elle ne peut être spécifiée que dans l'appel MQPUT ; elle n'est pas valide dans l'appel MQPUT1 .

Si MQPMO_LOGICAL_ORDER est spécifié, il indique que l'application utilise des appels MQPUT successifs pour :

1. Placer les segments dans chaque message logique dans l'ordre croissant des décalages de segment, à partir de 0, sans écart.
2. Placer tous les segments dans un message logique avant de les placer dans le message logique suivant.
3. Placer les messages logiques dans chaque groupe de messages dans l'ordre croissant des numéros de séquence de message, à partir de 1, sans écart. IBM MQ incrémente le numéro de séquence de message automatiquement.

4. Placer tous les messages logiques dans un groupe de messages avant de les placer dans le groupe de messages suivant.

Etant donné que l'application a indiqué au gestionnaire de files d'attente comment elle insère des messages dans des groupes et des segments de messages logiques, l'application n'a pas besoin de gérer et de mettre à jour les informations de groupe et de segment relatives à chaque appel MQPUT, car le gestionnaire de files d'attente gère et met à jour ces informations. En particulier, cela signifie que l'application n'a pas besoin de définir les zones *GroupId*, *MsgSeqNumber*et *Offset* dans MQMD, car le gestionnaire de files d'attente définit ces zones avec les valeurs appropriées. L'application doit uniquement définir la zone *MsgFlags* dans MQMD, pour indiquer quand les messages appartiennent à des groupes ou sont des segments de messages logiques, et pour indiquer le dernier message dans un groupe ou le dernier segment d'un message logique.

Une fois qu'un groupe de messages ou un message logique a été démarré, les appels MQPUT suivants doivent spécifier les indicateurs MQMF_ * appropriés dans *MsgFlags* dans MQMD. Si l'application tente d'insérer un message qui ne se trouve pas dans un groupe lorsqu'il existe un groupe de messages sans fin, ou d'insérer un message qui n'est pas un segment lorsqu'il existe un message logique sans fin, l'appel échoue avec le code anomalie MQRC_INCOMPLETE_GROUP ou MQRC_INCOMPLETE_MSG, selon le cas. Toutefois, le gestionnaire de files d'attente conserve les informations relatives au groupe de messages en cours ou au message logique en cours, et l'application peut les arrêter en envoyant un message (éventuellement sans données de message d'application) en spécifiant MQMF_LAST_MSG_IN_GROUP ou MQMF_LAST_SEGMENT selon le cas, avant de réémettre l'appel MQPUT pour placer le message qui ne fait pas partie du groupe ou qui n'est pas un segment.

Le Figure 62, à la page 795 présente les combinaisons d'options et d'indicateurs valides, ainsi que les valeurs des zones *GroupId*, *MsgSeqNumber*et *Offset* utilisées par le gestionnaire de files d'attente dans chaque cas. Les combinaisons d'options et d'indicateurs qui ne sont pas affichées dans le tableau ne sont pas valides. Les colonnes du tableau ont les significations suivantes ; signifie Oui ou Non:

NOM DE JOURNAL
Indique si l'option MQPMO_LOGICAL_ORDER est spécifiée sur l'appel.

MIG
Indique si l'option MQMF_MSG_IN_GROUP ou MQMF_LAST_MSG_IN_GROUP est spécifiée dans l'appel.

SEG
Indique si l'option MQMF_SEGMENT ou MQMF_LAST_SEGMENT est spécifiée sur l'appel.

SEG OK
Indique si l'option MQMF_SEGMENTATION_ALLOWED est spécifiée sur l'appel.

Groupe de travail en cours
Indique si un groupe de messages en cours existe avant l'appel.

Message du journal en cours
Indique si un message logique en cours existe avant l'appel.

Autres colonnes
Affiche les valeurs utilisées par le gestionnaire de files d'attente. Précédent indique la valeur utilisée pour la zone dans le message précédent pour l'identificateur de file d'attente.

Tableau 116. Options MQPUT relatives aux messages dans les groupes et les segments de messages logiques

Options que vous spécifiez	Options que vous spécifiez	Options que vous spécifiez	Options que vous spécifiez	Statut du groupe et du message de journal avant l'appel	Statut du groupe et du message de journal avant l'appel	Valeurs utilisées par le gestionnaire de files d'attente	Valeurs utilisées par le gestionnaire de files d'attente	Valeurs utilisées par le gestionnaire de files d'attente
NOM DE JOURNAL	MIG	SEG	SEG OK	Groupe de travail en cours	Message du journal en cours	**GroupId**	**MsgSeqNumber**	**Offset**
Oui	Non	Non	Non	Non	Non	MQGI_AUCUN	1	0
Oui	Non	Non	Oui	Non	Non	Nouvel ID de groupe	1	0
Oui	Non	Oui	L'un ou l'autre	Non	Non	Nouvel ID de groupe	1	0
Oui	Non	Oui	L'un ou l'autre	Non	Oui	ID groupe précédent	1	Décalage précédent + longueur de segment précédent
Oui	Oui	L'un ou l'autre	L'un ou l'autre	Non	Non	Nouvel ID de groupe	1	0
Oui	Oui	L'un ou l'autre	L'un ou l'autre	Oui	Non	ID groupe précédent	Numéro de séquence précédent + 1	0
Oui	Oui	Oui	L'un ou l'autre	Oui	Oui	ID groupe précédent	Numéro de séquence précédent	Décalage précédent + longueur de segment précédent
Non	Non	Non	Non	L'un ou l'autre	L'un ou l'autre	MQGI_AUCUN	1	0
Non	Non	Non	Oui	L'un ou l'autre	L'un ou l'autre	Nouvel ID de groupe si MQGI_NONE, valeur else dans la zone	1	0

Tableau 116. Options MQPUT relatives aux messages dans les groupes et les segments de messages logiques (suite)

Options que vous spécifiez	Options que vous spécifiez	Options que vous spécifiez	Options que vous spécifiez	Statut du groupe et du message de journal avant l'appel	Statut du groupe et du message de journal avant l'appel	Valeurs utilisées par le gestionnaire de files d'attente	Valeurs utilisées par le gestionnaire de files d'attente	Valeurs utilisées par le gestionnaire de files d'attente
Non	Non	Oui	L'un ou l'autre	L'un ou l'autre	L'un ou l'autre	Nouvel ID de groupe si MQGI_NONE, valeur else dans la zone	1	Valeur dans la zone
Non	Oui	Non	L'un ou l'autre	L'un ou l'autre	L'un ou l'autre	Nouvel ID de groupe si MQGI_NONE, valeur else dans la zone	Valeur dans la zone	0
Non	Oui	Oui	L'un ou l'autre	L'un ou l'autre	L'un ou l'autre	Nouvel ID de groupe si MQGI_NONE, valeur else dans la zone	Valeur dans la zone	Valeur dans la zone

Remarque :

- MQPMO_LOGICAL_ORDER n'est pas valide sur l'appel MQPUT1 .
- Pour la zone *MsgId* , le gestionnaire de files d'attente génère un nouvel identificateur de message si MQPMO_NEW_MSG_ID ou MQMI_NONE est spécifié et utilise la valeur de la zone dans le cas contraire.
- Pour la zone *CorrelId* , le gestionnaire de files d'attente génère un nouvel identificateur de corrélation si MQPMO_NEW_CORREL_ID est spécifié et utilise la valeur de la zone dans le cas contraire.

Lorsque vous spécifiez MQPMO_LOGICAL_ORDER, le gestionnaire de files d'attente requiert que tous les messages d'un groupe et les segments d'un message logique soient insérés avec la même valeur dans la zone *Persistence* de MQMD, c'est-à-dire que tous les messages soient persistants ou non persistants. Si cette condition n'est pas satisfaite, l'appel MQPUT échoue avec le code anomalie MQRC_INCONSISTENT_PERSISTENCE.

L'option MQPMO_LOGICAL_ORDER affecte les unités de travail comme suit:

- Si le premier message physique d'un groupe ou d'un message logique est inséré dans une unité d'oeuvre, tous les autres messages physiques du groupe ou du message logique doivent être insérés dans une unité d'oeuvre, si le même descripteur de file d'attente est utilisé. Toutefois, il n'est pas nécessaire de les placer dans la même unité d'oeuvre, ce qui permet de diviser un groupe de messages ou un message logique composé de plusieurs messages physiques entre deux ou plusieurs unités d'oeuvre consécutives pour le descripteur de file d'attente.

- Si le premier message physique d'un groupe ou d'un message logique n'est pas inséré dans une unité d'oeuvre, aucun des autres messages physiques du groupe ou du message logique ne peut être inséré dans une unité d'oeuvre si le même descripteur de file d'attente est utilisé.

Si ces conditions ne sont pas satisfaites, l'appel MQPUT échoue avec le code anomalie MQRC_INCONSISTENT_UOW.

Lorsque MQPMO_LOGICAL_ORDER est spécifié, le MQMD fourni dans l'appel MQPUT ne doit pas être inférieur à MQMD_VERSION_2. Si cette condition n'est pas satisfaite, l'appel échoue avec le code anomalie MQRC_WRONG_MD_VERSION.

Si MQPMO_LOGICAL_ORDER n'est pas spécifié, les messages des groupes et des segments de messages logiques peuvent être insérés dans n'importe quel ordre et il n'est pas nécessaire d'insérer des groupes de messages complets ou des messages logiques complets. Il incombe à l'application de s'assurer que les zones *GroupId*, *MsgSeqNumber*, *Offset*et *MsgFlags* ont les valeurs appropriées.

Utilisez cette technique pour redémarrer un groupe de messages ou un message logique au milieu, après une défaillance du système. Lorsque le système redémarre, l'application peut définir les zones *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*set *Persistence* sur les valeurs appropriées, puis émettre l'appel MQPUT avec MQPMO_SYNCPOINT ou MQPMO_NO_SYNCPOINT défini comme requis, mais sans spécifier MQPMO_LOGICAL_ORDER. Si cet appel aboutit, le gestionnaire de files d'attente conserve les informations de groupe et de segment, et les appels MQPUT ultérieurs utilisant ce descripteur de file d'attente peuvent spécifier MQPMO_LOGICAL_ORDER comme normal.

Les informations de groupe et de segment que le gestionnaire de files d'attente conserve pour l'appel MQPUT sont distinctes des informations de groupe et de segment qu'il conserve pour l'appel MQGET.

Pour un descripteur de file d'attente donné, l'application peut combiner des appels MQPUT qui spécifient MQPMO_LOGICAL_ORDER avec des appels MQPUT qui ne le font pas, mais notez les points suivants:

- Si MQPMO_LOGICAL_ORDER n'est pas spécifié, chaque appel MQPUT réussi entraîne le gestionnaire de files d'attente à définir les informations de groupe et de segment pour l'identificateur de file d'attente sur les valeurs spécifiées par l'application, en remplaçant les informations de groupe et de segment existantes conservées par le gestionnaire de files d'attente pour l'identificateur de file d'attente.
- Si MQPMO_LOGICAL_ORDER n'est pas spécifié, l'appel n'échoue pas s'il existe un groupe de messages ou un message logique en cours ; l'appel peut aboutir avec un code achèvement MQCC_WARNING. Le Tableau 117, à la page 800 montre les différentes observations qui peuvent survenir. Dans ces cas, si le code achèvement n'est pas MQCC_OK, le code anomalie est l'un des suivants (selon le cas):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_PERSISTENCE
  - MQRC_INCONSISTENT_UOW

  Remarque : Le gestionnaire de files d'attente ne vérifie pas les informations de groupe et de segment pour l'appel MQPUT1 .

Tableau 117. Résultat lorsque l'appel MQPUT ou MQCLOSE n'est pas cohérent avec les informations de groupe et de segment

L'appel en cours est	L'appel précédent était MQPUT avec MQPMO_LOGICAL_ORDER	L'appel précédent était MQPUT sans MQPMO_LOGICAL_ORDER
MQPUT avec MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT sans MQPMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK

Tableau 117. Résultat lorsque l'appel MQPUT ou MQCLOSE n'est pas cohérent avec les informations de groupe et de segment (suite)

L'appel en cours est	L'appel précédent était MQPUT avec MQPMO_LOGICAL_ORDER	L'appel précédent était MQPUT sans MQPMO_LOGICAL_ORDER
MQCLOSE avec un groupe non terminé ou un message logique	MQCC_WARNING	MQCC_OK

Pour les applications qui placent des messages et des segments dans l'ordre logique, spécifiez MQPMO_LOGICAL_ORDER, car il s'agit de l'option la plus simple à utiliser. Cette option dispense l'application de la nécessité de gérer les informations de groupe et de segment, car le gestionnaire de files d'attente gère ces informations. Toutefois, les applications spécialisées peuvent avoir besoin de plus de contrôle que celui fourni par l'option MQPMO_LOGICAL_ORDER, qui peut être obtenue en ne spécifiant pas cette option ; dans ce cas, vous devez vous assurer que les zones *GroupId*, *MsgSeqNumber*, *Offset*et *MsgFlags* de MQMD sont correctement définies, avant chaque appel MQPUT ou MQPUT1 .

Par exemple, une application qui souhaite transmettre des messages physiques qu'elle reçoit, sans tenir compte du fait que ces messages se trouvent dans des groupes ou des segments de messages logiques, ne doit pas spécifier MQPMO_LOGICAL_ORDER, pour deux raisons:

- Si les messages sont extraits et placés dans l'ordre, la spécification de MQPMO_LOGICAL_ORDER affecte un nouvel identificateur de groupe aux messages, ce qui peut rendre difficile ou impossible pour l'émetteur des messages de corréler les messages de réponse ou de rapport qui résultent du groupe de messages.
- Dans un réseau complexe avec plusieurs chemins entre les gestionnaires de files d'attente d'envoi et de réception, les messages physiques peuvent arriver dans le désordre. En ne spécifiant pas MQPMO_LOGICAL_ORDER et MQGMO_LOGICAL_ORDER dans l'appel MQGET, l'application de réacheminement peut extraire et réacheminer chaque message physique dès qu'il arrive, sans attendre le message suivant dans l'ordre logique d'arrivée.

Les applications qui génèrent des messages de rapport pour des messages dans des groupes ou des segments de messages logiques ne doivent pas non plus spécifier MQPMO_LOGICAL_ORDER lors de l'insertion du message de rapport.

MQPMO_LOGICAL_ORDER peut être spécifié avec l'une des autres options MQPMO_ *.

Insertion de groupes ordonnés logiquement dans une file d'attente en cluster (MQOO_BIND_ON_GROUP)

L'option MQOO_BIND_ON_OPEN garantit que tous les messages de cette application, et donc tous les groupes, sont acheminés vers une seule instance. Cela présente l'inconvénient que le trafic de l'application n'est pas équilibré en charge sur plusieurs instances d'une file d'attente de cluster. Pour activer l'équilibrage de charge tout en conservant les groupes de messages intacts, vous devez définir les options suivantes:

- L'appel MQPUT doit spécifier MQPMO_LOGICAL_ORDER
- L'appel MQOPEN doit spécifier l'une des deux options suivantes:
  - MQOO_BIND_ON_GROUPE
  - MQOO_BIND_AS_Q_DEF et la définition de file d'attente doit spécifier DEFBIND (GROUP)

L'équilibrage de la charge de travail est ensuite piloté *entre les groupes* de messages sans nécessiter de MQCLOSE et MQOPEN de la file d'attente. *Entre les groupes* signifie que MQMF_MSG_IN_GROUP est défini dans MQMD (v2) ou MQMDE et qu'aucun groupe partiellement complet n'est en cours. Lorsqu'un groupe est en cours, le gestionnaire de files d'attente et le nom de file d'attente résolus dans l'identificateur d'objet sont réutilisés.

Si le message précédent était MQPMO_LOGICAL_ORDER et/ou que MQMF_MSG_IN_GROUP a été défini mais que le message en cours ne fait pas partie du groupe, l'appel PUT échoue avec MQRC_INCOMPLETE_GROUP.

Si un MQPUT individuel ne spécifie pas MQPMO_LOGICAL_ORDER et qu'aucun groupe en cours n'est actif, l'équilibrage de charge est géré pour ce message (comme si l'appel MQOPEN avait spécifié MQOO_BIND_NOT_FIXED).

Aucune réallocation n'est effectuée pour les messages liés à une destination à l'aide de MQOO_BIND_ON_GROUP. Pour plus d'informations sur la réallocation, voir «Groupes de messages», à la page 46.

Regroupement de messages logiques

Il existe deux raisons principales à l'utilisation de messages logiques dans un groupe:

- Vous devrez peut-être traiter les messages dans un ordre particulier.
- Il peut être nécessaire de traiter chaque message d'un groupe d'une manière associée.

Dans les deux cas, extrayez le groupe entier avec la même instance d'application d'obtention.

Par exemple, supposons que le groupe se compose de quatre messages logiques. L'application d'insertion se présente comme suit:

```
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
```

L'application d'obtention spécifie l'option MQGMO_ALL_MSGS_AVAILABLE pour le premier message du groupe. Cela garantit que le traitement ne démarre pas tant que tous les messages du groupe ne sont pas arrivés. L'option MQGMO_ALL_MSGS_AVAILABLE est ignorée pour les messages suivants au sein du groupe.

Lorsque le premier message logique du groupe est extrait, vous pouvez utiliser MQGMO_LOGICAL_ORDER pour vous assurer que les messages logiques restants du groupe sont extraits dans l'ordre.

Ainsi, l'application d'obtention se présente comme suit:

```
/* 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
```

Pour obtenir d'autres exemples de regroupement de messages, voir «Segmentation d'application des messages logiques», à la page 815 et «Mise en place et obtention d'un groupe couvrant des unités de travail», à la page 803.

Avertissement : Lors de l'utilisation de la fonction de publication / abonnement pour envoyer des messages à une rubrique (ou pour placer des messages dans un alias de rubrique), le regroupement et la segmentation de messages ne sont pas autorisés.

Etant donné que les abonnements peuvent être créés et supprimés indépendamment de l'activité de publication, il n'est pas certain qu'un abonné reçoive un groupe de messages complet ou tous les segments d'un message ; voir RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Pour plus d'informations sur l'autorisation d'une application à demander qu'un groupe de messages soit tous alloués à la même instance de destination pour les files d'attente de cluster, voir DefBind.

Mise en place et obtention d'un groupe couvrant des unités de travail

Dans le cas précédent, les messages ou les segments ne peuvent pas commencer à quitter le noeud (si sa destination est distante) ou à être extraits tant que le groupe entier n'a pas été inséré et que l'unité de travail n'a pas été validée. Cela peut ne pas être ce que vous souhaitez si l'insertion de l'ensemble du groupe prend beaucoup de temps ou si l'espace de file d'attente est limité sur le noeud. Pour y remédier, mettez le groupe dans plusieurs unités de travail.

Si le groupe est placé dans plusieurs unités de travail, il est possible qu'une partie du groupe soit validée même en cas d'échec de l'application d'insertion. L'application doit donc sauvegarder les informations de statut, validées avec chaque unité de travail, qu'elle peut utiliser après un redémarrage pour reprendre un groupe incomplet. L'emplacement le plus simple pour enregistrer ces informations se trouve dans une file d'attente STATUS. Si un groupe complet a été correctement inséré, la file d'attente STATUS est vide.

Si la segmentation est impliquée, la logique est similaire. Dans ce cas, **StatusInfo** doit inclure *Offset*.

Voici un exemple de placement du groupe dans plusieurs unités de travail:

```
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 toutes les unités de travail ont été validées, le groupe entier a été correctement inséré et la file d'attente STATUS est vide. Dans le cas contraire, le groupe doit être repris au point indiqué par les informations de statut. MQPMO_LOGICAL_ORDER ne peut pas être utilisé pour la première insertion, mais peut être utilisé par la suite.

Le processus de redémarrage se présente comme suit:

```
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
```

A partir de l'application d'extraction, vous pouvez commencer à traiter les messages d'un groupe avant l'arrivée de l'ensemble du groupe. Cela améliore les temps de réponse des messages au sein du groupe et signifie également que le stockage n'est pas requis pour l'ensemble du groupe. Afin de réaliser les avantages, utilisez plusieurs unités de travail pour chaque groupe de messages. Pour des raisons de reprise, vous devez extraire chaque message dans une unité de travail.

Comme pour l'application d'insertion correspondante, cela nécessite que les informations de statut soient enregistrées automatiquement quelque part au fur et à mesure que chaque unité de travail est validée. Là encore, l'emplacement le plus simple pour enregistrer ces informations se trouve dans une file d'attente STATUS. Si un groupe complet a été traité avec succès, la file d'attente STATUS est vide.

Remarque : Pour les unités de travail intermédiaires, vous pouvez éviter les appels MQGET de la file d'attente STATUS en spécifiant que chaque MQPUT dans la file d'attente d'état est un segment d'un message (c'est-à-dire en définissant l'indicateur MQMF_SEGMENT), au lieu d'insérer un nouveau message complet pour chaque unité de travail. Dans la dernière unité d'oeuvre, un segment final est placé dans la file d'attente de statut en spécifiant MQMF_LAST_SEGMENT, puis les informations de statut sont effacées avec une instruction MQGET spécifiant MQGMO_COMPLETE_MSG.

Lors du processus de redémarrage, au lieu d'utiliser une seule commande MQGET pour obtenir un message d'état possible, parcourez la file d'attente d'état avec MQGMO_LOGICAL_ORDER jusqu'à ce que vous atteigniez le dernier segment (c'est-à-dire jusqu'à ce qu'aucun autre segment ne soit renvoyé). Dans la première unité de travail après le redémarrage, indiquez également le décalage explicitement lors de l'insertion du segment de statut.

Dans l'exemple suivant, nous considérons uniquement les messages au sein d'un groupe, en supposant que la mémoire tampon de l'application est toujours suffisamment grande pour contenir l'intégralité du message, que le message ait été segmenté ou non. MQGMO_COMPLETE_MSG est donc spécifié sur chaque MQGET. Les mêmes principes s'appliquent si la segmentation est impliquée (dans ce cas, StatusInfo doit inclure *Offset*).

Par souci de simplicité, nous supposons qu'un maximum de 4 messages sont extraits dans une seule unité de travail:

```
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 toutes les unités de travail ont été validées, le groupe entier a été extrait avec succès et la file d'attente STATUS est vide. Dans le cas contraire, le groupe doit être repris au point indiqué par les informations de statut. MQGMO_LOGICAL_ORDER ne peut pas être utilisé pour la première extraction, mais peut l'être par la suite.

Le processus de redémarrage se présente comme suit:

```
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
```

Obtention d'un message particulier

Il existe plusieurs façons d'extraire un message particulier d'une file d'attente. Les options sont les suivantes: sélection sur MsgId et CorrelId, sélection sur GroupId, MsgSeqNumber and Offset et sélection sur MsgToken. Vous pouvez également utiliser une chaîne de sélection lorsque vous ouvrez la file d'attente.

Pour extraire un message particulier d'une file d'attente, utilisez les zones `MsgId` et `CorrelId` de la structure MQMD. Toutefois, les applications peuvent définir explicitement ces zones, de sorte que les valeurs que vous spécifiez peuvent ne pas identifier un message unique. Tableau 118, à la page 805 indique quel message est extrait pour les paramètres possibles de ces zones. Ces zones sont ignorées en entrée si vous spécifiez MQGMO_MSG_UNDER_CURSOR dans le paramètre **GetMsgOpts** de l'appel MQGET.

Tableau 118. Utilisation des identificateurs de message et de corrélation

Pour extraire ...	**MsgId**	**CorrelId**
Premier message de la file d'attente	MQMI_NONE	MQCI_NONE
Premier message correspondant à *MsgId*	Différent de zéro	MQCI_NONE
Premier message correspondant à *CorrelId*	MQMI_NONE	Différent de zéro
Premier message correspondant à la fois à *MsgId* et à *CorrelId*	Différent de zéro	Différent de zéro

Dans chaque cas, *premier* signifie le premier message qui répond aux critères de sélection (sauf si MQGMO_BROWSE_NEXT est spécifié, lorsqu'il signifie le message *suivant* dans la séquence qui répond aux critères de sélection).

En cas de retour, l'appel MQGET définit les zones *MsgId* et *CorrelId* sur les identificateurs de message et de corrélation du message renvoyé, le cas échéant.

Si vous définissez la zone *Version* de la structure MQMD sur 2, vous pouvez utiliser les zones *GroupId*, *MsgSeqNumber*et *Offset* . Tableau 119, à la page 806 indique quel message est extrait pour les paramètres possibles de ces zones.

Tableau 119. Utilisation de l'identificateur de groupe

Pour extraire ...	**options de mise en correspondance**
Premier message de la file d'attente	MQMO_AUCUN
Premier message correspondant à MsgId	ID MQMO_MATCH_MSG_ID
Premier message correspondant à CorrelId	ID_CORREL_MQMO_MATCH_
Premier message correspondant à GroupId	ID_GROUPE_MQMO_MATCH
Premier message correspondant à MsgSeqNumber	NUMERO MQMO_MATCH_MSG_SEQ_NO
Premier message correspondant à MsgToken	MQMO_MATCH_MSG_TOKEN
Premier message correspondant à Offset	MQMO_MATCH_OFFSET

Remarques :

1. MQMO_MATCH_XXX implique que la zone *XXX* de la structure MQMD est définie sur la valeur à mettre en correspondance.
2. Les indicateurs MQMO peuvent être utilisés en combinaison. Par exemple, MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER et MQMO_MATCH_OFFSET peuvent être utilisés ensemble pour fournir le segment identifié par les zones GroupId, MsgSeqNumberet Offset .
3. Si vous spécifiez MQGMO_LOGICAL_ORDER, le message que vous tentez d'extraire est affecté car l'option dépend des informations d'état contrôlées pour le descripteur de file d'attente. Pour plus d'informations, voir «Ordre logique et physique», à la page 793 et Options.

L'appel MQGET extrait généralement le premier message d'une file d'attente. Si vous spécifiez un message particulier lorsque vous utilisez l'appel MQGET, le gestionnaire de files d'attente doit effectuer une recherche dans la file d'attente jusqu'à ce qu'il trouve ce message. Cela peut affecter les performances de votre application.

Si vous utilisez la version 2 ou une version ultérieure de la structure MQGMO et que vous n'indiquez pas les indicateurs MQMO_MATCH_MSG_ID ou MQMO_MATCH_CORREL_ID, vous n'avez pas besoin de réinitialiser les zones MsgId ou CorrelId entre les MQGET.

z/OS Sous IBM MQ for z/OS, l'attribut de file d'attente IndexType peut être utilisé pour augmenter la vitesse des opérations MQGET sur la file d'attente. Pour plus d'informations, voir «Type of index», à la page 811.

Vous pouvez extraire un message spécifique d'une file d'attente en spécifiant MsgToken et MatchOption MQMO_MATCH_MSG_TOKEN dans la structure MQGMO. MsgToken est renvoyé par l'appel MQPUT qui a initialement inséré ce message dans la file d'attente ou par des opérations MQGET précédentes et reste constant sauf si le gestionnaire de files d'attente est redémarré.

Si vous n'êtes intéressé que par un sous-ensemble de messages de la file d'attente, vous pouvez spécifier les messages à traiter à l'aide d'une chaîne de sélection avec l'appel MQOPEN ou MQSUB. MQGET extrait ensuite le message suivant qui satisfait cette chaîne de sélection. Pour plus d'informations sur les chaînes de sélection, voir «Sélecteurs», à la page 32.

Amélioration des performances des messages non persistants

Lorsqu'un client requiert un message d'un serveur, il envoie une demande au serveur. Il envoie une demande distincte pour chacun des messages qu'il consomme. Pour améliorer les performances d'un client consommant des messages non persistants en évitant d'avoir à envoyer ces messages de

demande, un client peut être configuré pour utiliser la *lecture anticipée*. La lecture anticipée permet d'envoyer des messages à un client sans qu'une application n'ait à les demander.

Lorsque la lecture anticipée est activée, les messages sont envoyés à une mémoire tampon sur le client appelée *mémoire tampon de lecture anticipée*. Le client dispose d'une mémoire tampon de lecture anticipée pour chaque file d'attente ouverte avec la fonction de lecture anticipée activée. Les messages de la mémoire tampon de lecture anticipée ne sont pas conservés. Le client met régulièrement à jour le serveur avec des informations sur la quantité de données qu'il a consommée.

Lorsque vous appelez MQOPEN avec MQOO_READ_AHEAD, le client IBM MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- L'application client doit être compilée et liée dans les bibliothèques client IBM MQ MQI des unités d'exécution.
- Le canal client doit utiliser le protocole TCP/IP.
- Le paramètre SharingConversations (SHARECNV) du canal doit avoir une valeur différente de zéro dans la définition de canal serveur et dans la définition de canal client.

L'utilisation de la lecture anticipée peut améliorer les performances lors de la consommation de messages non persistants à partir d'une application client. Cette amélioration des performances est disponible pour les applications MQI et JMS . Les applications client utilisant MQGET ou la consommation asynchrone bénéficieront des améliorations de performances lors de la consommation de messages non persistants.

Toutes les conceptions d'application client ne sont pas adaptées à l'utilisation de la lecture anticipée car toutes les options ne sont pas prises en charge pour une utilisation avec la lecture anticipée et certaines options doivent être cohérentes entre les appels MQGET lorsque la lecture anticipée est activée. Si un client modifie ses critères de sélection entre les appels MQGET, les messages stockés dans la mémoire tampon de lecture anticipée restent bloqués dans la mémoire tampon de lecture anticipée du client.

Si un journal des messages bloqués avec les critères de sélection précédents n'est plus nécessaire, un intervalle de purge configurable peut être défini sur le client pour purger automatiquement ces messages du client. L'intervalle de purge fait partie d'un groupe d'options d'optimisation de la lecture anticipée déterminées par le client. Il est possible d'optimiser ces options pour répondre à vos besoins.

Si une application client est redémarrée, les messages de la mémoire tampon de lecture anticipée peuvent être perdus. A l'inverse, un message qui a été déplacé dans une mémoire tampon de lecture anticipée peut ensuite être supprimé de la file d'attente sous-jacente ; cela n'entraîne pas sa suppression de la mémoire tampon, de sorte qu'un appel MQGET utilisant la lecture anticipée peut renvoyer un message qui n'existe plus.

La lecture anticipée est effectuée uniquement pour les liaisons client. L'attribut est ignoré pour toutes les autres liaisons.

La lecture anticipée n'a aucun effet sur le déclenchement. Aucun message de déclenchement n'est généré lorsqu'un message est lu à l'avance par le client. La lecture anticipée ne génère pas d'informations de comptabilité et de statistiques lorsqu'elle est activée.

Utilisation de la lecture anticipée avec la messagerie de publication / abonnement

Lorsqu'une application d'abonnement spécifie une file d'attente de destination à laquelle les publications sont envoyées, la valeur DEFREADA de la file d'attente spécifiée est utilisée comme valeur de lecture anticipée par défaut.

Lorsqu'une application abonnée demande à IBM MQ de gérer la destination à laquelle les publications sont envoyées, une file d'attente gérée est créée en tant que file d'attente dynamique basée sur une file d'attente modèle prédéfinie. Il s'agit de la valeur DEFREADA de la file d'attente modèle utilisée comme valeur de lecture anticipée par défaut. Le modèle par défaut met en file d'attente SYSTEM.DURABLE.PUBLICATIONS.MODEL ou SYSTEM.NONDURABLE.PUBLICATIONS.MODEL est utilisé sauf si une file d'attente modèle est définie pour cette rubrique ou pour une rubrique parent.

Concepts associés

«Optimisation des performances des messages non persistants sous AIX», à la page 810
Si vous utilisez AIX V5.3 ou ultérieure, envisagez de définir votre paramètre d'optimisation afin d'utiliser des performances complètes pour les messages non persistants.

Tâches associées

«Activation et désactivation de la lecture anticipée», à la page 809
Par défaut, la lecture anticipée est désactivée. Vous pouvez activer la lecture anticipée au niveau de la file d'attente ou de l'application.

Référence associée

«Options MQGET et lecture anticipée», à la page 808
Toutes les options MQGET ne sont pas prises en charge lorsque la lecture anticipée est activée ; certaines options doivent être cohérentes entre les appels MQGET.

Options MQGET et lecture anticipée

Toutes les options MQGET ne sont pas prises en charge lorsque la lecture anticipée est activée ; certaines options doivent être cohérentes entre les appels MQGET.

Lorsque vous appelez MQOPEN avec MQOO_READ_AHEAD, le client IBM MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- L'application client doit être compilée et liée dans les bibliothèques client IBM MQ MQI des unités d'exécution.
- Le canal client doit utiliser le protocole TCP/IP.
- Le paramètre SharingConversations (SHARECNV) du canal doit avoir une valeur différente de zéro dans la définition de canal serveur et dans la définition de canal client.

Le tableau suivant indique quelles options sont prises en charge pour une utilisation avec la lecture anticipée et si elles peuvent être modifiées entre les appels MQGET.

Tableau 120. Options MQGET et lecture anticipée

Valeurs et options MQGET	**Autorisé lorsque la lecture anticipée est activée et peut être modifié entre les appels MQGET [5]**	**Admise lorsque la lecture anticipée est activée, mais non modifiable entre les appels MQGET[1]**	**Options MQGET non autorisées lorsque la lecture anticipée est activée[2]**
Valeurs MQGET MQMD	MsgId[3] CorrelId [3]	Codage CodedCharSetId	
Options 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_TRONQUÉ _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_ DISPONIBLE

Remarques :

1. Si ces options sont modifiées entre les appels MQGET, un code anomalie MQRC_OPTIONS_CHANGED est renvoyé.
2. Si ces options sont spécifiées lors du premier appel MQGET, la lecture anticipée est désactivée. Si ces options sont spécifiées lors d'un appel MQGET ultérieur, un code anomalie MQRC_OPTIONS_ERROR est renvoyé.

3. Si une application client modifie les valeurs MsgId et CorrelId entre les appels MQGET, il se peut que des messages avec les valeurs précédentes aient déjà été envoyés au client et qu'ils restent dans la mémoire tampon de lecture anticipée du client jusqu'à ce qu'ils soient consommés (ou automatiquement purgés).
4. MQGMO_MSG_UNDER_CURSOR n'est pas possible avec la lecture anticipée. La lecture anticipée est désactivée lorsque les options MQOO_BROWSE et MQOO_INPUT_SHARED ou MQOO_INPUT_EXCLUSIVE sont spécifiées lors de l'ouverture de la file d'attente.
5. Lorsque la lecture anticipée est activée, la première commande MQGET détermine si les messages doivent être consultés ou reçus d'une file d'attente. Si l'application client utilise alors MQGET avec des options modifiées, telles que la tentative de recherche après une extraction initiale ou la tentative de recherche après une exploration initiale, un code anomalie MQRC_OPTIONS_CHANGED est renvoyé.

Si un client modifie ses critères de sélection entre les appels MQGET, les messages stockés dans la mémoire tampon de lecture anticipée qui correspondent aux critères de sélection initiaux ne sont pas consommés par l'application client et restent bloqués dans la mémoire tampon de lecture anticipée du client. Dans les situations où la mémoire tampon de lecture anticipée du client contient de nombreux messages bloqués, les avantages associés à la lecture anticipée sont perdus et une demande distincte au serveur est requise pour chaque message consommé. Pour déterminer si la lecture anticipée est utilisée efficacement, vous pouvez utiliser le paramètre de statut de connexion READA.

La lecture anticipée peut être désactivée lorsqu'elle est demandée par une application en raison d'options incompatibles spécifiées dans le premier appel MQGET. Dans cette situation, l'état de la connexion indique que la lecture anticipée est désactivée.

Si, en raison de ces restrictions sur MQGET, vous décidez qu'une conception d'application client n'est pas adaptée à la lecture anticipée, spécifiez l'option MQOPEN MQOO_READ_AHEAD_NO. Vous pouvez également définir la valeur de lecture anticipée par défaut de la file d'attente en cours d'ouverture sur NO ou DISABLED.

Activation et désactivation de la lecture anticipée
Par défaut, la lecture anticipée est désactivée. Vous pouvez activer la lecture anticipée au niveau de la file d'attente ou de l'application.

Pourquoi et quand exécuter cette tâche

Lorsque vous appelez MQOPEN avec MQOO_READ_AHEAD, le client IBM MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- L'application client doit être compilée et liée dans les bibliothèques client IBM MQ MQI des unités d'exécution.
- Le canal client doit utiliser le protocole TCP/IP.
- Le paramètre SharingConversations (SHARECNV) du canal doit avoir une valeur différente de zéro dans la définition de canal serveur et dans la définition de canal client.

Pour activer la lecture anticipée:

- Pour configurer la lecture anticipée au niveau de la file d'attente, définissez l'attribut de file d'attente DEFREADA sur YES.
- Pour configurer la lecture anticipée au niveau de l'application:
  - pour utiliser la lecture anticipée dans la mesure du possible, utilisez l'option MQOO_READ_AHEAD sur l'appel de fonction MQOPEN. L'application client ne peut pas utiliser la lecture anticipée si l'attribut de file d'attente DEFREADA a été défini sur DISABLED.
  - pour utiliser la lecture anticipée uniquement lorsque la lecture anticipée est activée sur une file d'attente, utilisez l'option MQOO_READ_AHEAD_AS_Q_DEF sur l'appel de fonction MQOPEN.

Si une conception d'application client n'est pas adaptée à la lecture anticipée, vous pouvez la désactiver:

- au niveau de la file d'attente en définissant l'attribut de file d'attente, DEFREADA sur NO si vous ne voulez pas que la lecture anticipée soit utilisée à moins qu'elle ne soit demandée par une application

client, ou DISABLED si vous ne voulez pas que la lecture anticipée soit utilisée, que la lecture anticipée soit requise ou non par une application client.

- au niveau de l'application en utilisant l'option MQOO_NO_READ_AHEAD sur l'appel de fonction MQOPEN.

Deux options MQCLOSE vous permettent de configurer ce qu'il advient des messages qui sont stockés dans la mémoire tampon de lecture anticipée si la file d'attente est fermée.

- Utilisez MQCO_IMMEDIATE pour supprimer les messages dans la mémoire tampon de lecture anticipée.
- Utilisez MQCO_QUIESCE pour vous assurer que les messages de la mémoire tampon de lecture anticipée sont consommés par l'application avant la fermeture de la file d'attente. Lorsque MQCLOSE avec MQCO_QUIESCE est émis et qu'il reste des messages dans la mémoire tampon de lecture anticipée, MQRC_READ_AHEAD_MSGS est renvoyé avec MQCC_WARNING.

Optimisation des performances des messages non persistants sous AIX
Si vous utilisez AIX V5.3 ou ultérieure, envisagez de définir votre paramètre d'optimisation afin d'utiliser des performances complètes pour les messages non persistants.

Pour définir le paramètre d'optimisation afin qu'il prenne effet immédiatement, exécutez la commande suivante en tant que superutilisateur:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Pour définir le paramètre d'optimisation de sorte qu'il prenne effet immédiatement et qu'il persiste lors des réamorçages, exécutez la commande suivante en tant que superutilisateur:

```
/usr/sbin/ioo -p -o j2_nPagesPerWriteBehindCluster=0
```

En règle générale, les messages non persistants sont conservés uniquement en mémoire, mais dans certains cas, AIX peut planifier l'écriture de messages non persistants sur le disque. Les messages planifiés pour être écrits sur le disque ne sont pas disponibles pour MQGET tant que l'écriture sur le disque n'est pas terminée. La commande d'optimisation suggérée fait varier ce seuil ; au lieu de planifier l'écriture des messages sur le disque lorsque 16 kilooctets de données sont mis en file d'attente, l'écriture sur le disque se produit uniquement lorsque la mémoire réelle de la machine est presque saturée. Il s'agit d'une modification globale qui peut affecter d'autres composants logiciels.

Sous AIX, lors de l'utilisation d'applications à unités d'exécution multiples et en particulier lors de l'exécution sur des machines à processeurs multiples, il est fortement recommandé de définir AIXTHREAD_SCOPE=S dans l'ID mqm `.profile` ou de définir AIXTHREAD_SCOPE=S dans l'environnement avant de démarrer l'application, pour de meilleures performances et une planification plus fiable. Exemple :

```
export AIXTHREAD_SCOPE=S
```

La définition de AIXTHREAD_SCOPE=S signifie que les unités d'exécution utilisateur créées avec des attributs par défaut sont placées dans la portée des conflits à l'échelle du système. Si une unité d'exécution utilisateur est créée avec une portée de conflit à l'échelle du système, elle est liée à une unité d'exécution du noyau et elle est planifiée par le noyau. L'unité d'exécution du noyau sous-jacente n'est pas partagée avec une autre unité d'exécution utilisateur.

Descripteurs de fichier

Lorsque vous exécutez un processus à plusieurs unités d'exécution tel qu'un processus agent, vous risquez d'atteindre la limite souple des descripteurs de fichier. Cette limite vous donne le IBM MQ code anomalie `MQRC_UNEXPECTED_ERROR (2195)` et, s'il y a suffisamment de descripteurs de fichier, un fichier IBM MQ FFST™ .

Pour éviter ce problème, vous pouvez augmenter la limite de processus pour le nombre de descripteurs de fichier. Pour ce faire, modifiez l'attribut `nofiles` dans `/etc/security/limits` en lui attribuant la valeur 10 000 pour l'ID utilisateur mqm ou dans la strophe par défaut.

Limites des ressources du système

Définissez la limite des ressources système pour un segment de données et un segment de piles sur illimité en lançant les commandes suivantes dans une invite :

```
ulimit -d unlimited
ulimit -s unlimited
```

Type of index

The queue attribute, *IndexType*, specifies the type of index that the queue manager maintains to increase the speed of MQGET operations on the queue.

Note: Supported only on IBM MQ for z/OS.

You have five options:

Value	Description
NONE	No index is maintained. Use this when retrieving messages sequentially (see “Priorité” on page 793).
GROUPID	An index of group identifiers is maintained. You must use this index type if you want logical ordering of message groups (see “Ordre logique et physique” on page 793).
MSGID	An index of message identifiers is maintained. Use this when retrieving messages using the *MsgId* field as a selection criterion on the MQGET call (see “Obtention d'un message particulier” on page 805).
MSGTOKEN	An index of message tokens is maintained.
CORRELID	An index of correlation identifiers is maintained. Use this when retrieving messages using the *CorrelId* field as a selection criterion on the MQGET call (see “Obtention d'un message particulier” on page 805).

Note:

1. If you are indexing using the MSGID option or CORRELID option, set the relative **MsgId** or **CorrelId** parameters in the MQMD. It is not beneficial to set both.
2. Browse uses the index mechanism to find a message if a queue matches all the following conditions:
   - It has index type MSGID, CORRELID, or GROUPID
   - It is browsed with the same type of id
   - It has messages of only one priority
3. Avoid queues (indexed by *MsgId* or *CorrelId*) containing thousands of messages because this affects restart time. (This does not apply to nonpersistent messages as they are deleted at restart.)
4. MSGTOKEN is used to define queues managed by the z/OS workload manager.

For a full description of the **IndexType** attribute, see IndexType. For further information on the **IndexType** attribute, see “Design and performance considerations for z/OS applications” on page 68.

Traitement des messages d'une longueur supérieure à 4 Mo

Les messages peuvent être trop volumineux pour l'application, la file d'attente ou le gestionnaire de files d'attente. En fonction de l'environnement, IBM MQ fournit un certain nombre de façons de traiter les messages dont la taille est supérieure à 4 Mo.

Vous pouvez augmenter l'attribut **MaxMsgLength** jusqu'à 100 Mo sur tous les systèmes IBM MQ de la version V6 ou ultérieure. Définissez cette valeur pour refléter la taille des messages utilisant la file d'attente. Sous IBM MQ for Multiplatforms, vous pouvez également:

1. Utilisez des messages segmentés. (Les messages peuvent être segmentés par l'application ou le gestionnaire de files d'attente.)
2. Utilisez les messages de référence.

Chacune de ces approches est décrite dans la suite de cette section.

Augmentation de la longueur maximale des messages

L'attribut de gestionnaire de files d'attente **MaxMsgLength** définit la longueur maximale d'un message pouvant être traité par un gestionnaire de files d'attente. De même, l'attribut de file d'attente **MaxMsgLength** correspond à la longueur maximale d'un message pouvant être traité par une file d'attente. La longueur de message maximale par défaut prise en charge dépend de l'environnement dans lequel vous travaillez.

Multi Sous IBM MQ for Multiplatforms, vous pouvez définir ces deux attributs manuellement. Vous pouvez définir une valeur d'attribut de gestionnaire de files d'attente comprise entre 32768 octets et 100 Mo.

Avertissement : z/OS Sous IBM MQ for z/OS , l'attribut de gestionnaire de files d'attente **MaxMsgLength** est codé en dur à 100 Mo.

Après avoir modifié l'un des attributs **MaxMsgLength** ou les deux, redémarrez vos applications et canaux pour vous assurer que les modifications sont prises en compte.

Lorsque ces modifications sont effectuées, la longueur des messages doit être inférieure ou égale aux attributs **MaxMsgLength** de la file d'attente et du gestionnaire de files d'attente. Toutefois, les messages existants peuvent être plus longs que les deux attributs.

Si le message est trop volumineux pour la file d'attente, MQRC_MSG_TOO_BIG_FOR_Q est renvoyé. De même, si le message est trop volumineux pour le gestionnaire de files d'attente, MQRC_MSG_TOO_BIG_FOR_Q_MGR est renvoyé.

Cette méthode de traitement des messages volumineux est facile et pratique. Toutefois, tenez compte des facteurs suivants avant de l'utiliser:

- L'uniformité entre les gestionnaires de files d'attente est réduite. La taille maximale des données de message est déterminée par le *MaxMsgLength* pour chaque file d'attente (y compris les files d'attente de transmission) dans laquelle le message sera inséré. Cette valeur est souvent définie par défaut sur le *MaxMsgLength*du gestionnaire de files d'attente, en particulier pour les files d'attente de transmission. Il est donc difficile de prévoir si un message est trop volumineux lorsqu'il doit être acheminé vers un gestionnaire de files d'attente éloignées.
- L'utilisation des ressources système est accrue. Par exemple, les applications ont besoin de mémoires tampon plus grandes, et sur certaines plateformes, il peut y avoir une utilisation accrue du stockage partagé. Le stockage en file d'attente ne doit être affecté que s'il est réellement requis pour les messages plus volumineux.
- La création de lots de canaux est affectée. Un message volumineux est toujours comptabilisé comme un seul message dans le nombre de lots, mais il a besoin de plus de temps pour être transmis, ce qui augmente les temps de réponse pour les autres messages.

Multi *Segmentation des messages*

Utilisez ces informations pour en savoir plus sur la segmentation des messages. Cette fonction n'est pas prise en charge sous IBM MQ for z/OS ou par les applications utilisant IBM MQ classes for JMS.

L'augmentation de la longueur maximale des messages, comme expliqué dans la rubrique «Augmentation de la longueur maximale des messages», à la page 812 , a des implications négatives. De plus, le message peut être trop volumineux pour la file d'attente ou le gestionnaire de files d'attente. Dans ces

cas, vous pouvez segmenter un message. Pour plus d'informations sur les segments, voir «Groupes de messages», à la page 46.

Les sections suivantes présentent les utilisations courantes de la segmentation des messages. Pour l'insertion et l'extraction destructive, il est supposé que les appels MQPUT ou MQGET *toujours* fonctionnent dans une unité de travail. Pensez toujours à utiliser cette technique pour réduire la possibilité que des groupes incomplets soient présents dans le réseau. La validation en une phase par le gestionnaire de files d'attente est supposée, mais les autres techniques de coordination sont également valides.

De plus, dans les applications d'extraction, il est supposé que si plusieurs serveurs traitent la même file d'attente, chaque serveur exécute un code similaire, de sorte qu'un serveur ne trouve jamais un message ou un segment qu'il s'attend à y trouver (car il a spécifié MQGMO_ALL_MSGS_AVAILABLE ou MQGMO_ALL_SEGMENTS_AVAILABLE auparavant).

Avertissement : Lors de l'utilisation de la fonction de publication / abonnement pour envoyer des messages à une rubrique (ou pour placer des messages dans un alias de rubrique), le regroupement et la segmentation de messages ne sont pas autorisés.

Etant donné que les abonnements peuvent être créés et supprimés indépendamment de l'activité de publication, il n'est pas certain qu'un abonné reçoive un groupe de messages complet ou tous les segments d'un message ; voir RC2417: MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Insertion et obtention d'un message segmenté qui s'étend sur des unités de travail

Vous pouvez insérer et obtenir un message segmenté qui s'étend sur une unité de travail d'une manière similaire à «Mise en place et obtention d'un groupe couvrant des unités de travail», à la page 803.

Toutefois, vous ne pouvez pas insérer ou obtenir des messages segmentés dans une unité d'oeuvre globale.

Multi *Segmentation et réassemblage par gestionnaire de files d'attente*

Il s'agit du scénario le plus simple, dans lequel une application insère un message à extraire par une autre. Le message peut être volumineux: pas trop grand pour que l'application d'insertion ou d'extraction puisse le traiter dans une seule mémoire tampon, mais trop grand pour le gestionnaire de files d'attente ou une file d'attente dans laquelle le message doit être inséré.

Les seules modifications nécessaires pour ces applications sont que l'application d'insertion autorise le gestionnaire de files d'attente à effectuer une segmentation si nécessaire:

```
   PMO.Options = (existing options)
   MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
   MD.Version = MQMD_VERSION_2
   memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)
MQPUT
```

et pour que l'application d'extraction demande au gestionnaire de files d'attente de réassembler le message s'il a été segmenté:

```
   GMO.Options = MQGMO_COMPLETE_MSG | (existing options)
   MQGET
```

Dans ce scénario le plus simple, l'application doit réinitialiser la zone GroupId sur MQGI_NONE avant l'appel MQPUT, afin que le gestionnaire de files d'attente puisse générer un identificateur de groupe unique pour chaque message. Si tel n'est pas le cas, les messages non liés peuvent avoir le même identificateur de groupe, ce qui peut entraîner un traitement incorrect.

La mémoire tampon de l'application doit être suffisamment grande pour contenir le message réassemblé (sauf si vous incluez l'option MQGMO_ACCEPT_TRUNCATED_MSG).

Si l'attribut MAXMSGLEN d'une file d'attente doit être modifié pour tenir compte de la segmentation des messages, tenez compte des points suivants:

- Le nombre minimal de segments de message pris en charge dans une file d'attente locale est de 16 octets.
- Pour une file d'attente de transmission, MAXMSGLEN doit également inclure l'espace requis pour les en-têtes. Envisagez d'utiliser une valeur supérieure d'au moins 4000 octets à la longueur maximale attendue des données utilisateur dans tout segment de message pouvant être placé dans une file d'attente de transmission.

Si la conversion de données est nécessaire, l'application d'extraction peut avoir à le faire en spécifiant MQGMO_CONVERT. Cela doit être simple car l'exit de conversion de données est présenté avec le message complet. Ne tentez pas de convertir des données dans un canal émetteur si le message est segmenté et que le format des données est tel que l'exit de conversion de données ne peut pas effectuer la conversion sur des données incomplètes.

Multi *Segmentation de l'application*

La segmentation des applications est utilisée lorsque la segmentation du gestionnaire de files d'attente n'est pas adéquate ou lorsque les applications nécessitent une conversion de données avec des limites de segment spécifiques.

La segmentation des applications est utilisée pour deux raisons principales:

1. La segmentation du gestionnaire de files d'attente seule n'est pas adéquate car le message est trop volumineux pour être traité dans une seule mémoire tampon par les applications.
2. La conversion des données doit être effectuée par les canaux émetteurs, et le format est tel que la demande de placement doit préciser où doivent se trouver les limites des segments afin que la conversion d'un segment individuel soit possible.

Toutefois, si la conversion de données n'est pas un problème ou si l'application d'extraction utilise toujours MQGMO_COMPLETE_MSG, la segmentation du gestionnaire de files d'attente peut également être autorisée en spécifiant MQMF_SEGMENTATION_ALLOWED. Dans notre exemple, l'application segmente le message en quatre segments:

```
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 vous n'utilisez pas MQPMO_LOGICAL_ORDER, l'application doit définir *Offset* et la longueur de chaque segment. Dans ce cas, l'état logique n'est pas géré automatiquement.

L'application d'extraction ne peut pas garantir une mémoire tampon suffisamment grande pour contenir un message réassemblé. Elle doit donc être préparée pour traiter les segments individuellement.

Pour les messages segmentés, cette application ne souhaite pas commencer à traiter un segment tant que tous les segments qui constituent le message logique ne sont pas présents. MQGMO_ALL_SEGMENTS_AVAILABLE est donc spécifié pour le premier segment. Si vous spécifiez MQGMO_LOGICAL_ORDER et qu'il existe un message logique en cours, MQGMO_ALL_SEGMENTS_AVAILABLE est ignoré.

Une fois que le premier segment d'un message logique a été extrait, utilisez MQGMO_LOGICAL_ORDER pour vous assurer que les segments restants du message logique sont extraits dans l'ordre.

Les messages des différents groupes ne sont pas pris en compte. Si de tels messages se produisent, ils sont traités dans l'ordre dans lequel le premier segment de chaque message apparaît dans la file d'attente.

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
do while ( SegmentStatus == MQSS_SEGMENT )
   MQGET
```

```
   /* Process each remaining segment of the logical message */
   ...

MQCMIT
```

Multi *Segmentation d'application des messages logiques*

Les messages doivent être gérés dans l'ordre logique dans un groupe, et certains ou tous peuvent être tellement volumineux qu'ils nécessitent une segmentation de l'application.

Dans notre exemple, un groupe de quatre messages logiques doit être inséré. Tous les messages, à l'exception du troisième, sont volumineux et nécessitent une segmentation, qui est effectuée par l'application d'insertion:

```
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
```

Dans l'application d'obtention, MQGMO_ALL_MSGS_AVAILABLE est spécifié dans la première requête MQGET. Cela signifie qu'aucun message ou segment d'un groupe n'est extrait tant que le groupe entier n'est pas disponible. Lorsque le premier message physique d'un groupe a été extrait, MQGMO_LOGICAL_ORDER est utilisé pour s'assurer que les segments et les messages du groupe sont extraits dans l'ordre:

```
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
```

Remarque : Si vous spécifiez MQGMO_LOGICAL_ORDER et qu'il existe un groupe en cours, MQGMO_ALL_MSGS_AVAILABLE est ignoré.

Multi *Messages de référence et transferts d'objets LOB*

Les messages de référence permettent de transférer un objet LOB d'un noeud à un autre sans stocker l'objet dans des files d'attente IBM MQ sur les noeuds source ou de destination. Ceci est particulièrement intéressant lorsque les données existent sous une autre forme, par example pour des applications de messagerie.

Pour activer cette méthode de transfert, vous devez spécifier un exit de message aux deux extrémités d'un canal. Pour savoir comment procéder, voir «Programmes d'exit de message de canal», à la page 1006.

IBM MQ définit le format d'un en-tête de message de référence (MQRMH). Pour plus d'informations, voir MQRMH . Il est reconnu avec un nom de format défini et peut être suivi de données réelles.

Pour initier le transfert d'un objet de grande taille, une application peut placer un message constitué d'un en-tête de message de référence sans données le suivant. Lorsque ce message quitte le noeud, l'exit

de message extrait l'objet de manière appropriée et l'ajoute au message de référence. Il renvoie ensuite le message (désormais plus grand qu'avant) à l'agent MCA émetteur pour transmission à l'agent MCA récepteur.

Un autre exit de message est configuré au niveau de l'agent MCA récepteur. Lorsque cet exit de message reçoit l'un de ces messages, il crée l'objet à l'aide des données d'objet qui ont été ajoutées et transmet le message de référence *sans* le message. Le message de référence peut maintenant être reçu par une application et cette application sait que l'objet (ou au moins la partie de celui-ci représentée par ce message de référence) a été créé sur ce noeud.

La quantité maximale de données objet qu'un exit de message d'envoi peut ajouter au message de référence est limitée par la longueur maximale de message négociée pour le canal. L'exit ne peut renvoyer qu'un seul message à l'agent MCA pour chaque message qu'il transmet, de sorte que l'application d'insertion peut insérer plusieurs messages pour provoquer le transfert d'un objet. Chaque message doit identifier la longueur *logique* et le décalage de l'objet qui doit lui être ajouté. Cependant, dans les cas où il n'est pas possible de connaître la taille totale de l'objet ou la taille maximale autorisée par le canal, concevez l'exit d'envoi de message de sorte que l'application d'insertion ne place qu'un seul message, et l'exit lui-même place le message suivant dans la file d'attente de transmission lorsqu'il a ajouté autant de données que possible au message qu'il a transmis.

Avant d'utiliser cette méthode de traitement des messages volumineux, tenez compte des points suivants:

- L'agent MCA et l'exit de message s'exécutent sous un ID utilisateur IBM MQ . L'exit de message (et, par conséquent, l'ID utilisateur) doit accéder à l'objet pour l'extraire à l'extrémité émettrice ou le créer à l'extrémité réceptrice ; cela ne peut être possible que dans les cas où l'objet est universellement accessible. Cela pose un problème de sécurité.
- Si le message de référence auquel sont ajoutées des données non formatées doit passer par plusieurs gestionnaires de files d'attente avant d'atteindre sa destination, les données non formatées sont présentes dans les files d'attente IBM MQ sur les noeuds intermédiaires. Toutefois, dans ces cas, il n'est pas nécessaire de prévoir un soutien ou des sorties spécifiques.
- La conception de votre exit de message est rendue difficile si le réacheminement ou la mise en file d'attente des messages non livrés est autorisé. Dans ces cas, les parties de l'objet peuvent arriver dans le désordre.
- Lorsqu'un message de référence arrive à sa destination, l'exit de message de réception crée l'objet. Toutefois, cette opération n'est pas synchronisée avec l'unité d'oeuvre de l'agent MCA. Par conséquent, si le lot est annulé, un autre message de référence contenant cette même partie de l'objet sera envoyé dans un lot ultérieur et l'exit de message pourra tenter de recréer la même partie de l'objet. Si l'objet est, par exemple, une série de mises à jour de base de données, cela peut être inacceptable. Si tel est le cas, l'exit de message doit conserver un journal dont les mises à jour ont été appliquées ; cela peut nécessiter l'utilisation d'une file d'attente IBM MQ .
- En fonction des caractéristiques du type d'objet, les exits de message et les applications peuvent avoir besoin de coopérer pour gérer le nombre d'utilisations, de sorte que l'objet puisse être supprimé lorsqu'il n'est plus nécessaire. Un identificateur d'instance peut également être requis ; une zone est fournie à cet effet dans l'en-tête du message de référence (voir MQRMH).
- Si un message de référence est inséré en tant que liste de distribution, l'objet doit être extractible pour chaque liste de distribution résultante ou destination individuelle sur ce noeud. Vous devrez peut-être gérer les nombres d'utilisations. Envisagez également la possibilité qu'un noeud soit le noeud final pour certaines des destinations de la liste, mais un noeud intermédiaire pour d'autres.
- Les données non formatées ne sont généralement pas converties. En effet, la conversion a lieu *avant* que l'exit de message ne soit appelé. Pour cette raison, la conversion ne doit pas être demandée sur le canal émetteur d'origine. Si le message de référence passe par un noeud intermédiaire, les données non formatées sont converties lorsqu'elles sont envoyées à partir du noeud intermédiaire, le cas échéant.
- Les messages de référence ne peuvent pas être segmentés.

Utilisation des structures MQRMH et MQMD

Pour obtenir une description des zones de l'en-tête de message de référence et du descripteur de message, voir MQRMH et MQMD .

Dans la structure MQMD, définissez la zone *Format* sur MQFMT_REF_MSG_HEADER. Le format MQHREF, lorsqu'il est demandé sur MQGET, est converti automatiquement par IBM MQ avec les données non formatées qui suivent.

Voici un exemple d'utilisation des zones *DataLogicalOffset* et *DataLogicalLength* de MQRMH:

Une application d'insertion peut insérer un message de référence avec:

- Aucune donnée physique
- *DataLogicalLength* = 0 (ce message représente l'objet entier)
- *DataLogicalOffset* = 0.

En supposant que l'objet a une longueur de 70 000 octets, l'exit de message d'émission envoie les 40 000 premiers octets le long du canal dans un message de référence contenant:

- 40 000 octets de données physiques suivant le MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (à partir du début de l'objet).

Il place ensuite un autre message dans la file d'attente de transmission contenant:

- Aucune donnée physique
- *DataLogicalLength* = 0 (à la fin de l'objet). Vous pouvez spécifier une valeur de 30 000 ici.
- *DataLogicalOffset* = 40000 (à partir de ce point).

Lorsque cet exit de message est vu par l'exit de message d'envoi, les 30 000 octets de données restants sont ajoutés et les zones sont définies sur:

- 30 000 octets de données physiques après le MQRMH
- *DataLogicalLength* = 30000
- *DataLogicalOffset* = 40000 (à partir de ce point).

L'indicateur MQRMHF_LAST est également défini.

Pour obtenir une description des exemples de programmes fournis pour l'utilisation des messages de référence, voir «Utilisation des exemples de programme sur Multiplatforms», à la page 1086.

En attente de messages

Si vous souhaitez qu'un programme attende l'arrivée d'un message dans une file d'attente, spécifiez l'option MQGMO_WAIT dans la zone *Options* de la structure MQGMO.

Utilisez la zone *WaitInterval* de la structure MQGMO pour spécifier Durée maximale (en millisecondes) pendant laquelle vous souhaitez qu'un appel MQGET attende l'arrivée d'un message dans une file d'attente.

Si le message n'arrive pas dans ce délai, l'appel MQGET se termine avec le code anomalie MQRC_NO_MSG_AVAILABLE.

Vous pouvez spécifier un intervalle d'attente illimité à l'aide de la constante MQWI_UNLIMITED dans la zone *WaitInterval* . Cependant, les événements échappant à votre contrôle peuvent entraîner une longue attente de votre programme. Par conséquent, utilisez cette constante avec précaution. Les applications IMS ne doivent pas spécifier un intervalle d'attente illimité car cela empêcherait l'arrêt du système IMS . (Lorsque IMS s'arrête, toutes les régions dépendantes doivent s'arrêter.) A la place, les applications IMS peuvent spécifier un intervalle d'attente fini ; ensuite, si l'appel se termine sans extraire de message après cet intervalle, émettez un autre appel MQGET avec l'option d'attente.

Remarque : Si plusieurs programmes attendent dans la même file d'attente partagée pour *supprimer* un message, un seul programme est activé par l'arrivée d'un message. Toutefois, si plusieurs programmes

sont en attente de parcourir un message, tous les programmes peuvent être activés. Pour plus d'informations, voir la description de la zone *Options* de la structure MQGMO dans MQGMO.

Si l'état de la file d'attente ou du gestionnaire de files d'attente change avant l'expiration de l'intervalle d'attente, les actions suivantes se produisent:

- Si le gestionnaire de files d'attente passe à l'état de mise au repos et que vous avez utilisé l'option MQGMO_FAIL_IF_QUIESCING, l'attente est annulée et l'appel MQGET se termine avec le code anomalie MQRC_Q_MGR_QUIESCING. Sans cette option, l'appel reste en attente.
- z/OS Sous z/OS, si la connexion (pour une application CICS ou IMS) passe à l'état de mise au repos et que vous avez utilisé l'option MQGMO_FAIL_IF_QUIESCING, l'attente est annulée et l'appel MQGET se termine avec le code anomalie MQRC_CONN_QUIESCING. Sans cette option, l'appel reste en attente.
- Si l'arrêt du gestionnaire de files d'attente est forcé ou annulé, l'appel MQGET se termine avec le code anomalie MQRC_Q_MGR_STOPPING ou MQRC_CONNECTION_BROKEN.
- Si les attributs de la file d'attente (ou d'une file d'attente dans laquelle le nom de la file d'attente est résolu) sont modifiés de sorte que les demandes d'extraction sont désormais interdites, l'attente est annulée et l'appel MQGET se termine avec le code anomalie MQRC_GET_INHIBÉ.
- Si les attributs de la file d'attente (ou une file d'attente dans laquelle le nom de la file d'attente est résolu) sont modifiés de telle sorte que l'option FORCE est requise, l'attente est annulée et l'appel MQGET se termine avec le code anomalie MQRC_OBJECT_CHANGED.

z/OS Si vous souhaitez que votre application attende sur plusieurs files d'attente, utilisez la fonction de signal de IBM MQ for z/OS (voir «Signaling», à la page 818). Pour plus d'informations sur les circonstances dans lesquelles ces actions se produisent, voir MQGMO.

Signaling

Signaling is supported only on IBM MQ for z/OS.

Signaling is an option on the MQGET call to allow the operating system to notify (or *signal*) a program when an expected message arrives on a queue. This is like the *get with wait* function described in topic “En attente de messages” on page 817 because it allows your program to continue with other work while waiting for the signal. However, if you use signaling, you can free the application thread and rely on the operating system to notify the program when a message arrives.

To set a signal

To set a signal, do the following in the MQGMO structure that you use on your MQGET call:

1. Set the MQGMO_SET_SIGNAL option in the *Options* field.
2. Set the maximum life of the signal in the *WaitInterval* field. This sets the length of time (in milliseconds) for which you want IBM MQ to monitor the queue. Use the MQWI_UNLIMITED value to specify an unlimited life.

   Note: IMS applications must not specify an unlimited wait interval because this would prevent the IMS system from terminating. (When IMS terminates, it requires all dependent regions to end.) Instead, IMS applications can examine the state of the ECB at regular intervals (see step 3). A program can have signals set on several queue handles at the same time:
3. Specify the address of the *Event Control Block* (ECB) in the *Signal1* field. This notifies you of the result of your signal. The ECB storage must remain available until the queue is closed.

Note: You cannot use the MQGMO_SET_SIGNAL option with the MQGMO_WAIT option.

When the message arrives

When a suitable message arrives, a completion code is returned to the ECB.

The completion code describes one of the following:

- The message that you set the signal for has arrived on the queue. The message is not reserved for the program that requested a signal, so the program must issue an MQGET call again to get the message.

  Note: Another application could get the message in the time between your receiving the signal and issuing another MQGET call.
- The wait interval you set has expired and the message you set the signal for did not arrive on the queue. IBM MQ has canceled the signal.
- The signal has been canceled. This happens, for example, if the queue manager stops, or the attribute of the queue is changed, so that MQGET calls are no longer allowed.

When a suitable message is already on the queue, the MQGET call completes in the same way as an MQGET call without signaling. Also, if an error is detected immediately, the call completes and the return codes are set.

When the call is accepted and no message is immediately available, control is returned to the program so that it can continue with other work. None of the output fields in the message descriptor are set, but the **CompCode** parameter is set to MQCC_WARNING and the **Reason** parameter is set to MQRC_SIGNAL_REQUEST_ACCEPTED.

For information about what IBM MQ can return to your application when it makes an MQGET call using signaling, see MQGET.

If the program has no other work to do while it is waiting for the ECB to be posted, it can wait for the ECB using:

- For a CICS Transaction Server for z/OS program, the EXEC CICS WAIT EXTERNAL command
- For batch and IMS programs, the z/OS WAIT macro

If the state of the queue or the queue manager changes while the signal is set (that is, the ECB has not yet been posted), the following actions occur:

- If the queue manager enters the quiescing state, and you used the MQGMO_FAIL_IF_QUIESCING option, the signal is canceled. The ECB is posted with the MQEC_Q_MGR_QUIESCING completion code. Without this option, the signal remains set.
- If the queue manager is forced to stop, or is canceled, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.
- If the attributes of the queue (or a queue to which the queue name resolves) are changed so that get requests are now inhibited, the signal is canceled. The signal is delivered with the MQEC_WAIT_CANCELED completion code.

Note:

1. If more than one program has set a signal on the same shared queue to remove a message, only one program is activated by a message arriving. However, if more than one program is waiting to browse a message, all the programs can be activated. The rules that the queue manager follows when deciding which applications to activate are the same as those for waiting applications: for more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
2. If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. For more information, see the description of the *Options* field of the MQGMO structure in MQGMO - Get-message options.
3. Under some conditions, it is possible both for an MQGET call to retrieve a message and for a signal (resulting from the arrival of the same message) to be delivered. This means that when your program issues another MQGET call (because the signal was delivered), there could be no message available. Design your program to test for this situation.

For information about how to set a signal, see the description of the MQGMO_SET_SIGNAL option and the *Signal1* field in Signal1.

z/OS *Annulation ignorée*

Vous pouvez empêcher un programme d'application d'entrer dans une boucle *MQGET-error-backout* en spécifiant l'option **MQGMO_MARK_SKIP_BACKOUT** dans l'appel MQGET.

Dans le cadre d'une unité de travail, un programme d'application peut émettre un ou plusieurs appels MQGET pour extraire des messages d'une file d'attente. Si le programme d'application détecte une erreur, il peut rétablir l'unité d'oeuvre. Ainsi, toutes les ressources mises à jour au cours de cette unité de travail sont restaurées à l'état dans lequel elles se trouvaient avant le démarrage de l'unité de travail et rétablissent les messages extraits par les appels MQGET.

Une fois rétablis, ces messages sont disponibles pour les appels MQGET ultérieurs émis par le programme d'application. Dans de nombreux cas, cela ne pose pas de problème pour le programme d'application. Toutefois, dans les cas où l'erreur entraînant l'annulation ne peut pas être contournée, le fait que le message soit réintégré dans la file d'attente peut entraîner l'entrée du programme d'application dans une boucle *MQGET-error-backout* .

Pour éviter ce problème, spécifiez l'option MQGMO_MARK_SKIP_BACKOUT dans l'appel MQGET. Cela marque la demande MQGET comme n'étant pas impliquée dans l'annulation initiée par l'application, c'est-à-dire qu'elle ne doit pas être annulée. L'utilisation de cette option signifie que lorsqu'une annulation se produit, les mises à jour d'autres ressources sont annulées selon les besoins, mais le message marqué est traité comme s'il avait été extrait sous une nouvelle unité d'oeuvre.

Le programme d'application doit émettre un appel IBM MQ pour valider la nouvelle unité de travail ou pour l'éliminer. Par exemple, le programme peut traiter les exceptions, par exemple en informant l'émetteur que le message a été supprimé et en validant l'unité d'oeuvre de sorte à supprimer le message de la file d'attente. Si la nouvelle unité d'oeuvre est annulée (pour une raison quelconque), le message est réintégré dans la file d'attente.

Dans une unité de travail, il ne peut y avoir qu'une seule demande MQGET marquée comme ayant ignoré l'annulation ; cependant, il peut y avoir plusieurs autres messages qui ne sont pas marqués comme ayant ignoré l'annulation. Une fois qu'un message a été marqué comme ignoré, tous les autres appels MQGET dans l'unité d'oeuvre qui spécifient MQGMO_MARK_SKIP_BACKOUT échouent avec le code anomalie MQRC_SECOND_MARK_NOT_ALLOWED.

Remarque :

1. Le message marqué ignore l'annulation uniquement si l'unité de travail qui le contient est arrêtée par une demande d'application pour l'annuler. Si l'unité d'oeuvre est annulée pour une autre raison, le message est annulé dans la file d'attente de la même manière que s'il n'était pas marqué pour ignorer l'annulation.
2. L'annulation de l'omission n'est pas prise en charge dans les procédures stockées Db2 qui participent à des unités de travail contrôlées par RRS. Par exemple, un appel MQGET avec l'option MQGMO_MARK_SKIP_BACKOUT échouera avec le code anomalie MQRC_OPTION_ENVIRONMENT_ERROR.

La Figure 63, à la page 821 illustre une séquence typique d'étapes qu'un programme d'application peut contenir lorsqu'une demande MQGET est requise pour ignorer l'annulation.

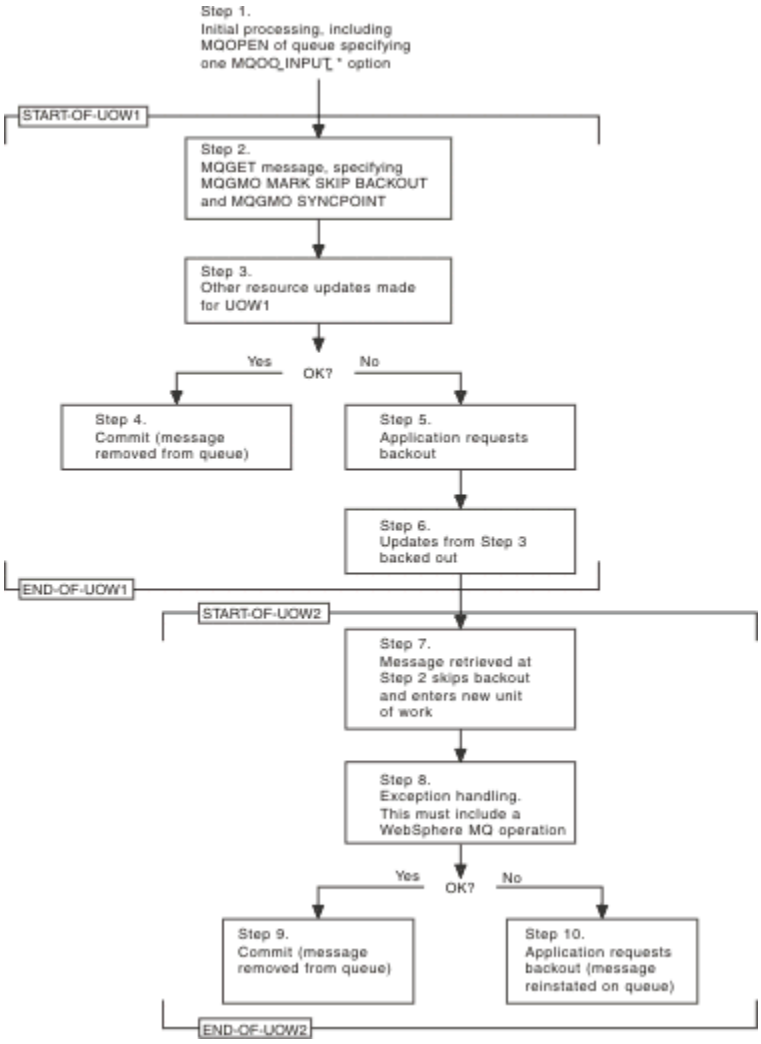


Figure 63. Annulation ignorée à l'aide de MQGMO_MARK_SKIP_BACKOUT

Les étapes de la rubrique Figure 63, à la page 821 sont les suivantes:

Etape 1
Le traitement initial se produit au sein de la transaction, y compris un appel MQOPEN pour ouvrir la file d'attente (en spécifiant l'une des options MQOO_INPUT_ * afin d'obtenir les messages de la file d'attente à l'étape 2).

Etape 2
MQGET est appelé avec MQGMO_SYNCPOINT et MQGMO_MARK_SKIP_BACKOUT. MQGMO_SYNCPOINT est obligatoire car MQGET doit se trouver dans une unité de travail pour que MQGMO_MARK_SKIP_BACKOUT soit effectif. Dans Figure 63, à la page 821 , cette unité de travail est appelée UOW1.

Etape 3
D'autres mises à jour de ressources sont effectuées dans le cadre de UOW1. Ils peuvent inclure d'autres appels MQGET (émis sans MQGMO_MARK_SKIP_BACKOUT).

Etape 4
Toutes les mises à jour des étapes 2 et 3 sont effectuées selon les besoins. Le programme d'application valide les mises à jour et UOW1 se termine. Le message extrait à l'étape 2 est supprimé de la file d'attente.

Etape 5
Certaines des mises à jour des étapes 2 et 3 ne sont pas effectuées comme requis. Le programme d'application demande que les mises à jour effectuées au cours de ces étapes soient annulées.

Etape 6
Les mises à jour effectuées à l'étape 3 sont annulées.

Etape 7
La demande MQGET effectuée à l'étape 2 ignore l'annulation et fait partie d'une nouvelle unité de travail, UOW2.

Étape 8
UOW2 effectue le traitement des exceptions en réponse à l'annulation de UOW1 . (Par exemple, un appel MQPUT à une autre file d'attente, indiquant qu'un problème a provoqué l'annulation de UOW1 .)

Étape 9
L'étape 8 se termine selon les besoins, le programme d'application valide l'activité et UOW2 se termine. Comme la demande MQGET fait partie de UOW2 (voir l'étape 7), cette validation entraîne la suppression du message de la file d'attente.

Etape 10
L'étape 8 ne s'exécute pas comme requis et le programme d'application annule UOW2. Etant donné que la demande d'obtention de message fait partie de UOW2 (voir l'étape 7), elle est également annulée et réintégrée dans la file d'attente. Il est désormais disponible pour d'autres appels MQGET émis par ce programme d'application ou par un autre (de la même manière que tout autre message de la file d'attente).

Conversion des données d'application

Si nécessaire, les agents MCA convertissent le descripteur de message et les données d'en-tête dans le jeu de caractères et le codage requis. Chaque extrémité de la liaison (c'est-à-dire l'agent MCA local ou l'agent MCA éloigné) peut effectuer la conversion.

Lorsqu'une application insère des messages dans une file d'attente, le gestionnaire de files d'attente local ajoute des informations de contrôle aux descripteurs de message afin de faciliter le contrôle des messages lorsqu'ils sont traités par les gestionnaires de files d'attente et les agents MCA. Selon l'environnement, les zones de données d'en-tête de message sont créées dans le jeu de caractères et le codage du système local.

Lorsque vous déplacez des messages entre des systèmes, vous devez parfois convertir les données d'application dans le jeu de caractères et le codage requis par le système récepteur. Cette opération peut être effectuée à partir de programmes d'application sur le système récepteur ou par les agents MCA sur le système émetteur. Si la conversion de données est prise en charge sur le système récepteur, utilisez des programmes d'application pour convertir les données d'application, plutôt que de dépendre de la conversion déjà effectuée sur le système émetteur.

Les données d'application sont converties dans un programme d'application lorsque vous spécifiez l'option MQGMO_CONVERT dans la zone *Options* de la structure MQGMO transmise à un appel MQGET et que *toutes* les instructions suivantes sont vraies:

- Les zones *CodedCharSetId* ou *Encoding* définies dans la structure MQMD associée au message dans la file d'attente diffèrent des zones *CodedCharSetId* ou *Encoding* définies dans la structure MQMD spécifiée dans l'appel MQGET.
- La zone *Format* de la structure MQMD associée au message n'est pas MQFMT_NONE.
- Le *BufferLength* spécifié dans l'appel MQGET est différent de zéro.
- La longueur des données de message n'est pas égale à zéro.
- Le gestionnaire de files d'attente prend en charge la conversion entre les zones *CodedCharSetId* et *Encoding* spécifiées dans les structures MQMD associées au message et à l'appel MQGET. Voir CodedCharSetId et Encoding pour plus de détails sur les identificateurs de jeu de caractères codés et les codages de machine pris en charge.
- Le gestionnaire de files d'attente prend en charge la conversion du format de message. Si la zone *Format* de la structure MQMD associée au message est l'un des formats intégrés, le gestionnaire de files d'attente peut convertir le message. Si *Format* n'est pas un des formats intégrés, vous devez écrire un exit de conversion de données pour convertir le message.

Si l'agent MCA émetteur doit convertir les données, indiquez le mot clé CONVERT (YES) dans la définition de chaque canal émetteur ou serveur pour lequel une conversion est requise. Si la conversion de données échoue, le message est envoyé à la file d'attente des messages non livrés au niveau du gestionnaire de files d'attente émetteur et la zone *Feedback* de la structure MQDLH en indique la raison. Si le message ne peut pas être inséré dans le DLQ, le canal se ferme et le message non converti reste dans la file d'attente de transmission. La conversion de données dans les applications plutôt que lors de l'envoi d'agents MCA évite cette situation.

En règle générale, les données du message qui sont décrites en tant que données *caractère* par le format intégré ou l'exit de conversion de données sont converties à partir du jeu de caractères codés utilisé par le message vers les données demandées, et les zones *numériques* sont converties au codage demandé.

Pour plus de détails sur les conventions de traitement de conversion utilisées lors de la conversion des formats intégrés et pour plus d'informations sur l'écriture de vos propres exits de conversion de données, voir «Ecriture des exits de conversion de données», à la page 1010. Voir aussi Langues nationales et Codages de machine pour plus d'informations sur les tables de support de langue et sur les codages de machine pris en charge.

Conversion des caractères de retour à la ligne EBCDIC

Si vous devez vous assurer que les données que vous envoyez d'une plateforme EBCDIC vers une plateforme ASCII sont identiques à celles que vous recevez à nouveau, vous devez contrôler la conversion des caractères de nouvelle ligne EBCDIC.

Vous pouvez effectuer cette opération à l'aide d'un commutateur dépendant de la plateforme qui force IBM MQ à utiliser les tables de conversion non modifiées, mais vous devez être conscient du comportement incohérent qui peut en résulter.

Le problème est dû au fait que le caractère de retour à la ligne EBCDIC n'est pas converti de manière cohérente entre les plateformes ou les tables de conversion. Par conséquent, si les données sont affichées sur une plateforme ASCII, le formatage peut être incorrect. Cela rendrait difficile, par exemple, l'administration à distance d'un système IBM i à partir d'une plateforme ASCII à l'aide de RUNMQSC.

Voir Conversion de données pour plus d'informations sur la conversion de données au format EBCDIC en format ASCII.

Recherche de messages dans une file d'attente

Utilisez ces informations pour découvrir comment parcourir les messages d'une file d'attente à l'aide de l'appel MQGET.

Pour utiliser l'appel MQGET afin de parcourir les messages d'une file d'attente:

1. Appelez MQOPEN pour ouvrir la file d'attente à des fins de consultation, en spécifiant l'option MQOO_BROWSE.
2. Pour parcourir le premier message de la file d'attente, appelez MQGET avec l'option MQGMO_BROWSE_FIRST. Pour trouver le message de votre choix, appelez MQGET à plusieurs reprises avec l'option MQGMO_BROWSE_NEXT pour parcourir plusieurs messages.

   Vous devez définir les zones *MsgId* et *CorrelId* de la structure MQMD sur null après chaque appel MQGET afin d'afficher tous les messages.
3. Appelez MQCLOSE pour fermer la file d'attente.

Le curseur de navigation

Lorsque vous ouvrez (MQOPEN) une file d'attente pour la navigation, l'appel établit un curseur de navigation à utiliser avec les appels MQGET qui utilisent l'une des options de navigation. Vous pouvez considérer le curseur de navigation comme un pointeur logique positionné avant le premier message de la file d'attente.

Vous pouvez activer plusieurs curseurs de navigation (à partir d'un seul programme) en émettant plusieurs demandes MQOPEN pour la même file d'attente.

Lorsque vous appelez MQGET pour la navigation, utilisez l'une des options suivantes dans votre structure MQGMO:

MQGMO_BROWSE_FIRST
Extrait une copie du premier message qui satisfait les conditions spécifiées dans votre structure MQMD.

MQGMO_BROWSE_NEXT
Obtient une copie du message suivant qui répond aux conditions spécifiées dans votre structure MQMD.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Extrait une copie du message actuellement désigné par le curseur, c'est-à-dire celui qui a été extrait pour la dernière fois à l'aide de l'option MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT.

Dans tous les cas, le message reste dans la file d'attente.

Lorsque vous ouvrez une file d'attente, le curseur de navigation est positionné logiquement juste avant le premier message de la file d'attente. Cela signifie que si vous effectuez votre appel MQGET immédiatement après votre appel MQOPEN, vous pouvez utiliser l'option MQGMO_BROWSE_NEXT pour parcourir le premier message ; vous n'avez pas besoin d'utiliser l'option MQGMO_BROWSE_FIRST.

L'ordre dans lequel les messages sont copiés à partir de la file d'attente est déterminé par l'attribut **MsgDeliverySequence** de la file d'attente. (Pour plus d'informations, voir «Ordre dans lequel les messages sont extraits d'une file d'attente», à la page 793.)

- «Files d'attente dans la séquence FIFO (premier entré, premier sorti)», à la page 824
- «Files d'attente dans l'ordre de priorité», à la page 824
- «Messages non validés», à la page 825
- «Modification de la séquence de la file d'attente», à la page 825
- z/OS «Utilisation de l'index de file d'attente», à la page 825

Files d'attente dans la séquence FIFO (premier entré, premier sorti)

Le premier message d'une file d'attente de cette séquence est le message qui a été le plus long dans la file d'attente.

Utilisez MQGMO_BROWSE_NEXT pour lire les messages de manière séquentielle dans la file d'attente. Vous verrez tous les messages placés dans la file d'attente pendant que vous parcourez, car une file d'attente de cette séquence contient des messages placés à la fin. Lorsque le curseur reconnaît qu'il a atteint la fin de la file d'attente, le curseur de navigation reste là où il se trouve et est renvoyé avec MQRC_NO_MSG_AVAILABLE. Vous pouvez ensuite le laisser en attente de messages supplémentaires ou le réinitialiser au début de la file d'attente avec un appel MQGMO_BROWSE_FIRST.

Files d'attente dans l'ordre de priorité

Le premier message dans une file d'attente de cette séquence est le message qui a été dans la file d'attente la plus longue et qui a la priorité la plus élevée au moment de l'émission de l'appel MQOPEN.

Utilisez MQGMO_BROWSE_NEXT pour lire les messages de la file d'attente.

Le curseur de navigation pointe vers le message suivant, en partant de la priorité du premier message pour terminer avec le message à la priorité la plus basse. Il parcourt tous les messages insérés dans la file d'attente pendant cette période tant qu'ils ont une priorité égale ou inférieure à celle du message identifié par le curseur de navigation en cours.

Les messages placés dans la file d'attente de priorité plus élevée peuvent être consultés uniquement par:

- Ouverture de la file d'attente pour l'exploration à nouveau, à partir de laquelle un nouveau curseur de navigation est établi
- Utilisation de l'option MQGMO_BROWSE_FIRST

Messages non validés

Un message non validé n'est jamais visible par une navigation ; le curseur de navigation l'ignore.

Les messages d'une unité de travail ne peuvent pas être consultés tant que l'unité de travail n'est pas validée. Les messages ne changent pas leur position dans la file d'attente lorsqu'ils sont validés. Par conséquent, les messages ignorés et non validés ne seront pas affichés, même lorsqu'ils *sont* validés, sauf si vous utilisez l'option MQGMO_BROWSE_FIRST et que vous utilisez à nouveau la file d'attente.

Modification de la séquence de la file d'attente

Si la séquence de distribution des messages passe de priorité à FIFO alors qu'il y a des messages dans la file d'attente, l'ordre des messages déjà mis en file d'attente n'est pas modifié. Les messages ajoutés ultérieurement à la file d'attente ont la priorité par défaut de la file d'attente.

Utilisation de l'index de file d'attente

z/OS

Sous IBM MQ for z/OS, lorsque vous parcourez une file d'attente indexée qui ne contient que des messages d'une seule priorité (persistante ou non persistante ou les deux), le gestionnaire de files d'attente utilise l'index pour naviguer lorsque certaines formes de navigation sont utilisées.

Les formes de navigation suivantes sont utilisées lorsqu'une file d'attente indexée contient uniquement des messages de priorité unique:

1. Si la file d'attente est indexée par MSGID, les demandes de navigation qui transmettent un MSGID dans la structure MQMD sont traitées à l'aide de l'index pour trouver le message cible.
2. Si la file d'attente est indexée par CORRELID, les demandes de navigation qui transmettent un CORRELID dans la structure MQMD sont traitées à l'aide de l'index pour trouver le message cible.
3. Si la file d'attente est indexée par GROUPID, les demandes de navigation qui passent un GROUPID dans la structure MQMD sont traitées à l'aide de l'index pour trouver le message cible.

Si la demande de navigation ne transmet pas de MSGID, CORRELID ou GROUPID dans la structure MQMD, la file d'attente est indexée et un message est renvoyé, l'entrée d'index du message doit être trouvée et les informations qu'elle contient doivent être utilisées pour mettre à jour le curseur de navigation. Si vous utilisez une large sélection de valeurs d'index, cela n'ajoute pas de traitement supplémentaire significatif à la demande de navigation.

Navigation dans les messages lorsque la longueur de message est inconnue

Pour parcourir un message lorsque vous ne connaissez pas la taille du message et que vous ne souhaitez pas utiliser les zones *MsgId*, *CorrelId*ou *GroupId* pour localiser le message, vous pouvez utiliser l'option MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Emettez une requête MQGET avec:
   - L'option MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT
   - Option MQGMO_ACCEPT_TRUNCATED_MSG
   - Longueur de la mémoire tampon zéro

   Remarque : Si un autre programme est susceptible d'obtenir le même message, envisagez également d'utiliser l'option MQGMO_LOCK. MQRC_TRUNCATED_MSG_ACCEPTED doit être renvoyé.
2. Utilisez le *DataLength* renvoyé pour allouer le stockage nécessaire.
3. Emettez une instruction MQGET avec MQGMO_BROWSE_MSG_UNDER_CURSOR.

Le message pointé est le dernier qui a été extrait ; le curseur de navigation n'a pas été déplacé. Vous pouvez choisir de verrouiller le message à l'aide de l'option MQGMO_LOCK ou de déverrouiller un message verrouillé à l'aide de l'option MQGMO_UNLOCK.

L'appel échoue si aucune instruction MQGET avec les options MQGMO_BROWSE_FIRST ou MQGMO_BROWSE_NEXT n'a été émise depuis l'ouverture de la file d'attente.

Suppression d'un message que vous avez parcouru
Vous pouvez supprimer de la file d'attente un message que vous avez déjà parcouru, à condition que vous ayez ouvert la file d'attente pour supprimer des messages ainsi que pour le parcourir. (Vous devez spécifier l'une des options MQOO_INPUT_ *, ainsi que l'option MQOO_BROWSE, sur votre appel MQOPEN.)

Pour supprimer le message, appelez à nouveau MQGET, mais dans la zone *Options* de la structure MQGMO, indiquez MQGMO_MSG_UNDER_CURSOR.Dans ce cas, l'appel MQGET ignore les zones *MsgId*, *CorrelId*et *GroupId* de la structure MQMD.

Entre les étapes de navigation et de suppression, il se peut qu'un autre programme ait supprimé des messages de la file d'attente, y compris le message sous votre curseur de navigation. Dans ce cas, votre appel MQGET renvoie un code anomalie indiquant que le message n'est pas disponible.

Exploration des messages dans l'ordre logique

Le «Ordre logique et physique», à la page 793 explique la différence entre l'ordre logique et physique des messages dans une file d'attente. Cette distinction est particulièrement importante lors de la navigation dans une file d'attente, car, en général, les messages ne sont pas supprimés et les opérations de navigation ne commencent pas nécessairement au début de la file d'attente.

Si une application parcourt les différents messages d'un groupe (en utilisant l'ordre logique), il est important de suivre l'ordre logique pour atteindre le début du groupe suivant, car le dernier message d'un groupe peut apparaître physiquement *après* le premier message du groupe suivant. L'option MQGMO_LOGICAL_ORDER garantit que l'ordre logique est respecté lors de l'analyse d'une file d'attente.

Utilisez MQGMO_ALL_MSGS_AVAILABLE (ou MQGMO_ALL_SEGMENTS_AVAILABLE) avec précaution pour les opérations de navigation. Prenons le cas des messages logiques avec MQGMO_ALL_MSGS_AVAILABLE. En conséquence, un message logique n'est disponible que si tous les messages restants du groupe sont également présents. Si ce n'est pas le cas, le message est transmis. Cela peut signifier que lorsque les messages manquants arrivent par la suite, ils ne sont pas remarqués par une opération de navigation suivante.

Par exemple, si les messages logiques suivants sont présents:

```
Logical message 1 (not last) of group 123
Logical message 1 (not last) of group 456
Logical message 2 (last)     of group 456
```

et une fonction de navigation est émise avec MQGMO_ALL_MSGS_AVAILABLE, le premier message logique du groupe 456 est renvoyé, laissant le curseur de navigation sur ce message logique. Si le deuxième (dernier) message du groupe 123 arrive:

```
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
```

et que la même fonction browse-next est émise, il n'est pas remarqué que le groupe 123 est maintenant terminé, car le premier message de ce groupe est *avant* le curseur browse.

Dans certains cas (par exemple, si les messages sont extraits de façon destructive lorsque le groupe est présent dans son intégralité), vous pouvez utiliser MQGMO_ALL_MSGS_AVAILABLE avec MQGMO_BROWSE_FIRST. Sinon, vous devez répéter l'analyse de navigation pour prendre en compte les messages nouvellement arrivés qui ont été manqués ; l'émission de MQGMO_WAIT avec MQGMO_BROWSE_NEXT et MQGMO_ALL_MSGS_AVAILABLE ne prend pas en compte ces messages. (Cela se produit également pour les messages de priorité plus élevée qui peuvent arriver après la fin de l'analyse des messages.)

Les sections suivantes présentent des exemples de navigation qui traitent des messages non segmentés ; les messages segmentés suivent des principes similaires.

Exploration des messages dans les groupes
Dans cet exemple, l'application parcourt chaque message de la file d'attente, dans l'ordre logique.

Les messages de la file d'attente peuvent être regroupés. Pour les messages groupés, l'application ne souhaite pas démarrer le traitement d'un groupe tant que tous les messages qu'elle contient ne sont pas arrivés. MQGMO_ALL_MSGS_AVAILABLE est donc indiquée pour le premier message du groupe ; pour les messages suivants du groupe, cette option n'est pas nécessaire.

MQGMO_WAIT est utilisée dans cet exemple. Toutefois, bien que l'attente puisse être satisfaite si un nouveau groupe arrive, pour les raisons indiquées dans «Exploration des messages dans l'ordre logique», à la page 826, elle n'est pas satisfaite si le curseur de navigation a déjà passé le premier message logique d'un groupe et que les messages restants arrivent. Néanmoins, l'attente d'un intervalle approprié garantit que l'application ne boucle pas en permanence en attendant de nouveaux messages ou segments.

MQGMO_LOGICAL_ORDER est utilisée partout, pour s'assurer que l'analyse est dans l'ordre logique. Cela contraste avec l'exemple de requête MQGET destructive, où, chaque groupe étant supprimé, MQGMO_LOGICAL_ORDER n'est pas utilisé lors de la recherche du premier (ou du seul) message d'un groupe.

On suppose que la mémoire tampon de l'application est toujours suffisamment grande pour contenir l'intégralité du message, que le message ait été segmenté ou non. MQGMO_COMPLETE_MSG est donc spécifié sur chaque MQGET.

Voici un exemple de navigation dans les messages logiques d'un groupe:

```
/* 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 */
   ...
```

Le groupe est répété jusqu'à ce que MQRC_NO_MSG_AVAILABLE soit renvoyé.

Exploration et extraction de façon destructive
Dans cet exemple, l'application parcourt chacun des messages logiques d'un groupe avant de décider s'il convient d'extraire ce groupe de façon destructive.

La première partie de cet exemple est similaire à la précédente. Cependant, dans ce cas, après avoir parcouru tout un groupe, nous décidons de revenir en arrière et de le récupérer de façon destructive.

Comme chaque groupe est supprimé dans cet exemple, MQGMO_LOGICAL_ORDER n'est pas utilisé lors de la recherche du premier ou du seul message d'un groupe.

Voici un exemple de navigation, puis d'extraction destructive:

```
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 */
...
```

Eviter la distribution répétée de messages parcourus
A l'aide de certaines options d'ouverture et d'obtention de message, vous pouvez marquer les messages comme ayant été consultés de sorte qu'ils ne soient pas extraits à nouveau par l'application en cours ou d'autres applications associées. Les messages peuvent être démarrés explicitement ou automatiquement afin de les rendre à nouveau disponibles pour la navigation.

Si vous parcourez les messages d'une file d'attente, vous pouvez les extraire dans un ordre différent de celui dans lequel vous les extrayez si vous les avez extraits de façon destructive. En particulier, vous pouvez parcourir le même message plusieurs fois, ce qui n'est pas possible s'il est supprimé de la file d'attente. Pour éviter cela, vous pouvez *marquer* les messages au fur et à mesure qu'ils sont consultés, et éviter d'extraire les messages marqués. On parle parfois de *navigation avec marque*. Pour marquer les messages consultés, utilisez l'option d'obtention de message MQGMO_MARK_BROWSE_HANDLE et pour extraire uniquement les messages non marqués, utilisez MQGMO_UNMARKED_BROWSE_MSG. Si vous utilisez la combinaison des options MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG et MQGMO_MARK_BROWSE_HANDLE et que vous émettez des demandes MQGET répétées, vous extrayez chaque message de la file d'attente à tour de rôle. Cela permet d'éviter la distribution répétée de messages même si MQGMO_BROWSE_FIRST est utilisé pour s'assurer que les messages ne sont pas ignorés. Cette combinaison d'options peut être représentée par la constante unique MQGMO_BROWSE_HANDLE. Lorsqu'aucun message de la file d'attente n'a été consulté, MQRC_NO_MSG_AVAILABLE est renvoyé.

Si plusieurs applications explorent la même file d'attente, elles peuvent l'ouvrir avec les options MQOO_CO_OP et MQOO_BROWSE. Le descripteur d'objet renvoyé par chaque MQOPEN est considéré comme faisant partie d'un groupe coopérant. Tout message renvoyé par un appel MQGET spécifiant l'option MQGMO_MARK_BROWSE_CO_OP est considéré comme marqué pour cet ensemble de descripteurs coopérant.

Si un message a été marqué depuis un certain temps, il peut être automatiquement démarqué par le gestionnaire de files d'attente et rendu disponible pour être à nouveau consulté. L'attribut de gestionnaire de files d'attente MsgMarkBrowseInterval indique le temps en millisecondes pendant lequel un message doit rester marqué pour l'ensemble de descripteurs coopérant. Un MsgMarkBrowseInterval de -1 signifie que les messages ne sont jamais automatiquement démarqués.

Lorsque le processus unique ou l'ensemble de processus coopératifs marquant les messages s'arrête, les messages marqués ne sont plus marqués.

Exemples de navigation coopérative

Vous pouvez exécuter plusieurs copies d'une application de répartiteur pour parcourir les messages d'une file d'attente et initier un consommateur en fonction du contenu de chaque message. Dans chaque répartiteur, ouvrez la file d'attente avec MQOO_CO_OP. Cela indique que les répartiteurs coopèrent et connaissent les messages marqués de l'autre. Chaque répartiteur effectue ensuite des appels MQGET répétés en spécifiant les options MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG et MQGMO_MARK_BROWSE_CO_OP (vous pouvez utiliser la constante unique MQGMO_BROWSE_CO_OP

pour représenter cette combinaison d'options). Chaque application de répartiteur extrait ensuite uniquement les messages qui n'ont pas encore été marqués par d'autres répartiteurs associés. Le répartiteur initialise un destinataire et transmet le MsgToken renvoyé par MQGET au destinataire, qui extrait de façon destructive le message de la file d'attente. Si le destinataire annule l'opération MQGET du message, le message est disponible pour que l'un des navigateurs le répartit à nouveau, car il n'est plus marqué. Si le destinataire n'effectue pas d'opération MQGET sur le message, le gestionnaire de files d'attente, une fois que le message MsgMarkBrowseInterval a été transmis, démarque le message pour l'ensemble de descripteurs coopérant et il peut être redistribué.

Au lieu de disposer de plusieurs copies d'une même application de répartiteur, vous pouvez avoir un certain nombre d'applications de répartiteur différentes qui parcourent la file d'attente, chacune étant adaptée pour traiter un sous-ensemble des messages de la file d'attente. Dans chaque répartiteur, ouvrez la file d'attente avec MQOO_CO_OP. Cela indique que les répartiteurs coopèrent et connaissent les messages marqués de l'autre.

- Si l'ordre de traitement des messages pour un répartiteur unique est important, chaque répartiteur effectue des appels MQGET répétés, en spécifiant les options MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG et MQGMO_MARK_BROWSE_HANDLE (ou MQGMO_BROWSE_HANDLE). Si le message consulté convient à ce répartiteur, il effectue un appel MQGET en spécifiant MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP et le MsgToken renvoyé par l'appel MQGET précédent. Si l'appel aboutit, le répartiteur initialise le consommateur en lui transmettant MsgToken .
- Si l'ordre de traitement des messages n'est pas important et que le répartiteur est censé traiter la plupart des messages qu'il rencontre, utilisez les options MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG et MQGMO_MARK_BROWSE_CO_OP (ou MQGMO_BROWSE_CO_OP). Si le répartiteur parcourt un message qu'il ne peut pas traiter, il annule le marquage du message en appelant MQGET avec l'option MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP et le MsgToken renvoyé précédemment.

Certains cas où l'appel MQGET échoue

Si certains attributs d'une file d'attente sont modifiés à l'aide de l'option FORCE sur une commande entre l'émission d'un appel MQOPEN et d'un appel MQGET, l'appel MQGET échoue et renvoie le code anomalie MQRC_OBJECT_CHANGED.

Le gestionnaire de files d'attente marque le descripteur d'objet comme n'étant plus valide. Cela se produit également si les modifications s'appliquent à une file d'attente dans laquelle le nom de la file d'attente est résolu. Les attributs qui affectent le descripteur de cette manière sont répertoriés dans la description de l'appel MQOPEN dans MQOPEN. Si votre appel renvoie le code anomalie MQRC_OBJECT_CHANGED, fermez la file d'attente, rouvrez-la, puis essayez d'obtenir un message à nouveau.

Si les opérations d'extraction sont interdites pour une file d'attente à partir de laquelle vous tentez d'extraire des messages (ou toute file d'attente dans laquelle le nom de la file d'attente est résolu), l'appel MQGET échoue et renvoie le code anomalie MQRC_GET_INHIBÉ. Cela se produit même si vous utilisez l'appel MQGET pour la navigation. Vous pouvez obtenir un message correctement si vous tentez d'appeler MQGET ultérieurement, si la conception de l'application est telle que d'autres programmes modifient régulièrement les attributs des files d'attente.

Si une file d'attente dynamique (temporaire ou permanente) a été supprimée, les appels MQGET utilisant un descripteur d'objet précédemment acquis échouent et renvoient le code anomalie MQRC_Q_DELETED.

Ecriture d'applications de publication / abonnement

Commencez à écrire des applications IBM MQ de publication / abonnement.

Pour une présentation des concepts de publication / abonnement, voir Messagerie de publication / abonnement.

Pour plus d'informations sur l'écriture de différents types d'applications de publication / abonnement, voir les rubriques suivantes:

- «Ecriture d'applications de publication», à la page 830

- «Ecriture d'applications d'abonné», à la page 837
- «Cycles de vie de publication / abonnement», à la page 856
- «Propriétés des messages de publication / abonnement», à la page 861
- «Ordre des messages», à la page 863
- «Interception des publications», à la page 863
- «Options de publication», à la page 871
- «Options d'abonnement», à la page 871

Concepts associés

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

«Ecriture d'une application de procédure pour la mise en file d'attente», à la page 738
La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

«Ecriture d'applications de procédure client», à la page 937
Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

«Création d'une application procédurale», à la page 1027
Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

«Traitement des erreurs de programme de procédure», à la page 1065
Ces informations expliquent les erreurs associées aux appels MQI de vos applications lors de l'exécution d'un appel ou de la distribution de son message à sa destination finale.

Tâches associées

«Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085
Ces exemples de programmes sont écrits dans des langages procéduraux et présentent des utilisations typiques de l'interface MQI (Message Queue Interface). Programmes IBM MQ sur différentes plateformes.

Ecriture d'applications de publication

Commencez par écrire des applications d'éditeur en étudiant deux exemples. La première est modélisée de la manière la plus proche possible sur une application point à point en plaçant des messages dans une file d'attente, et la seconde illustre la création dynamique de rubriques-un modèle plus courant pour les applications de diffuseur de publications.

L'écriture d'une application de diffuseur de publications IBM MQ simple est similaire à l'écriture d'une application de point à point IBM MQ qui insère des messages dans une file d'attente (Tableau 121, à la page 831). La différence est que les messages MQPUT sont envoyés à une rubrique et non à une file d'attente.

Tableau 121. Point à point par rapport au modèle de programme IBM MQ de publication / abonnement.		
Etape	**Appel de point à point MQ**	**Publier l'appel MQ**
Connexion à un gestionnaire de files d'attente	MQCONN	MQCONN
Ouvrir la file d'attente	MQOPEN	
Ouvrir la rubrique		MQOPEN
Message (s) d'insertion	MQPUT	MQPUT
Fermer la rubrique		MQCLOSE
Fermer la file d'attente	MQCLOSE	
Se déconnecter du gestionnaire de files d'attente	MQDISC	MQDISC

Pour le rendre concret, il existe deux exemples d'applications pour la publication des cours des actions. Dans le premier exemple («Exemple 1: diffuseur de publications vers une rubrique fixe», à la page 831), qui est modélisé de manière très précise lors de l'insertion de messages dans une file d'attente, l'administrateur crée une définition de rubrique de la même manière que lors de la création d'une file d'attente. Le programmeur code MQPUT pour écrire des messages dans la rubrique au lieu de les écrire dans une file d'attente. Dans le deuxième exemple («Exemple 2: diffuseur de publications vers une rubrique de variable», à la page 834), le modèle d'interaction du programme avec IBM MQ est similaire. La différence est que le programmeur fournit la rubrique dans laquelle le message est écrit, plutôt que l'administrateur. En pratique, cela signifie généralement que la chaîne de rubrique est un contenu défini ou fourni par une autre source, telle qu'une entrée humaine via un navigateur.

Concepts associés

«Ecriture d'applications d'abonné», à la page 837
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application IBM MQ consommant des messages à partir d'une file d'attente, une application qui crée un abonnement et ne nécessite aucune connaissance de la mise en file d'attente, et enfin un exemple qui utilise à la fois la mise en file d'attente et les abonnements.

Référence associée

DEFINE TOPIC
DISPLAYTOPIC
STATUT D'AFFICHAGE

Exemple 1: diffuseur de publications vers une rubrique fixe
Un programme IBM MQ pour illustrer la publication dans une rubrique définie par l'administrateur.

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

Voir la sortie dans Figure 65, à la page 832

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "IBMSTOCKPRICE";
    char     publicationDefault[] = "129";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle                    */
    MQHOBJ   Hobj = MQHO_NONE;            /* object handle sub queue              */
    MQLONG   CompCode = MQCC_OK;          /* completion code                      */
    MQLONG   Reason = MQRC_NONE;          /* reason code                          */
    MQOD     td = {MQOD_DEFAULT};         /* Object descriptor                    */
    MQMD     md = {MQMD_DEFAULT};         /* Message Descriptor                   */
    MQPMO   pmo = {MQPMO_DEFAULT};        /* put message options                  */
    MQCHAR   resTopicStr[151];            /* Returned vale of topic string        */
    char *   topicName   = topicNameDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){                         /* replace defaults with args if provided */
        default:
            publication = argv[2];
        case(2):
            topicName = argv[1];
        case(1):
            printf("Optional parameters:  TopicObject Publication\n");
    }
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;        /* Object is a topic                    */
        td.Version = MQOD_VERSION_4;       /* Descriptor needs to be V4            */
        strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode,
&Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" using topic \"%s\" to topic string \"%s\"\n",
                publication, td.ObjectName, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figure 64. Publieur IBM MQ simple vers une rubrique fixe.

```
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
```

Figure 65. Exemple de sortie du premier diffuseur de publications

Les lignes de code sélectionnées suivantes illustrent les aspects de l'écriture d'une application de publication pour IBM MQ.

char topicNameDefault[] = "IBMSTOCKPRICE";
Un nom de rubrique par défaut est défini dans le programme. Vous pouvez le remplacer en indiquant le nom d'un autre objet de rubrique comme premier argument du programme.

MQCHAR resTopicStr[151];
`resTopicStr` est pointé par `td.ResObjectString.VSPtr` et est utilisé par `MQOPEN` pour renvoyer la chaîne de rubrique résolue. Faites en sorte que la longueur de `resTopicStr` soit supérieure de un à la longueur transmise dans `td.ResObjectString.VSBufSize` afin de libérer de l'espace pour la terminaison nulle.

memset (resTopicStr, 0, sizeof(resTopicStr));
Initialisez `resTopicStr` avec des valeurs NULL pour vous assurer que la chaîne de rubrique résolue renvoyée dans un `MQCHARV` est terminée par une valeur NULL.

td.ObjectType = MQOT_TOPIC
Il existe un nouveau type d'objet pour la publication / l'abonnement: l' *objet de rubrique*.

td.Version = MQOD_VERSION_4;
Pour utiliser le nouveau type d'objet, vous devez utiliser au moins *version 4* du descripteur d'objet.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
`topicName` est le nom d'un objet de rubrique, parfois appelé objet de rubrique d'administration. Dans l'exemple, l'objet de rubrique doit être créé à l'avance, à l'aide de IBM MQ Explorer ou de cette commande MQSC,

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
La chaîne de rubrique résolue est répercutée dans le fichier `printf` final du programme. Configurez la structure `MQCHARV ResObjectString` pour IBM MQ afin de renvoyer la chaîne résolue au programme.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
Ouvrez la rubrique pour la sortie ; tout comme l'ouverture d'une file d'attente pour la sortie.

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
Vous voulez que les nouveaux abonnés puissent recevoir la publication, et en spécifiant `MQPMO_RETAIN` dans le diffuseur de publications, lorsque vous démarrez un abonné, il reçoit la dernière publication, publiée avant le démarrage de l'abonné, en tant que première publication correspondante. L'alternative consiste à fournir aux abonnés des publications publiées uniquement après le démarrage de l'abonné. De plus, un abonné a la possibilité de refuser de recevoir une publication conservée en spécifiant `MQSO_NEW_PUBLICATIONS_ONLY` dans son abonnement.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
Ajoutez 1 à la longueur de la chaîne transmise à `MQPUT` pour transmettre le caractère de fin null à IBM MQ dans le cadre de la mémoire tampon de messages.

Que démontre le premier exemple? L'exemple imite aussi étroitement que possible le modèle traditionnel essayé et testé pour l'écriture point à point des programmes IBM MQ . Une caractéristique importante du modèle de programmation IBM MQ est que le programmeur n'est pas concerné lorsque des messages sont envoyés. La tâche du programmeur consiste à se connecter à un gestionnaire de files d'attente et à lui transmettre les messages à distribuer aux destinataires. Dans le paradigme point à point, le programmeur ouvre une file d'attente (probablement une file d'attente alias) que l'administrateur a configurée. La file d'attente alias achemine les messages vers une file d'attente cible, soit sur le gestionnaire de files d'attente local, soit vers un gestionnaire de files d'attente éloignées. Pendant que les messages sont en attente de distribution, ils sont stockés dans des files d'attente entre la source et la destination.

Dans le modèle de publication / abonnement, au lieu d'ouvrir une file d'attente, le programmeur ouvre une rubrique. Dans notre exemple, la rubrique est associée à une chaîne de rubrique par un administrateur. Le gestionnaire de files d'attente transfère la publication, à l'aide de files d'attente, aux abonnés locaux ou distants dont les abonnements correspondent à la chaîne de rubrique de la

publication. Si des publications sont conservées, le gestionnaire de files d'attente conserve la dernière copie de la publication, même s'il n'a pas d'abonnés maintenant. La publication conservée est disponible pour être réutilisée par les futurs abonnés. L'application de publication ne joue aucun rôle dans la sélection ou le routage de la publication vers une destination ; sa tâche consiste à créer et à placer des publications dans les rubriques définies par l'administrateur.

Cet exemple de rubrique fixe est atypique pour de nombreuses applications de publication / abonnement: il est statique. Il nécessite qu'un administrateur définisse les chaînes de rubrique et modifie les rubriques sur lesquelles elles sont publiées. Généralement, les applications de publication / abonnement doivent connaître tout ou partie de l'arborescence de rubriques. Les rubriques changent peut-être fréquemment, ou bien même si les rubriques ne changent pas beaucoup, le nombre de combinaisons de rubriques est important et il est trop coûteux pour un administrateur de définir un noeud de rubrique pour chaque chaîne de rubrique sur laquelle il peut être nécessaire de publier. Les chaînes de rubrique ne sont peut-être pas connues avant la publication ; une application de diffuseur de publications peut utiliser les informations du contenu de la publication pour spécifier une chaîne de rubrique, ou elle peut avoir des informations sur les chaînes de rubrique à publier à partir d'une autre source, telle que l'entrée humaine à partir d'un navigateur. Pour prendre en charge des styles de publication plus dynamiques, l'exemple suivant montre comment créer des rubriques de manière dynamique, dans le cadre de l'application de publication.

Les sujets coupler les éditeurs et les abonnés ensemble. La conception des règles, ou de l'architecture, pour la désignation des rubriques et leur organisation dans des arborescences de rubriques est une étape importante du développement d'une solution de publication / abonnement. Examinez attentivement la mesure dans laquelle l'organisation de l'arborescence de rubriques lie les programmes de diffuseur de publications et d'abonné ensemble et les lie au contenu de l'arborescence de rubriques. Posez-vous la question de savoir si les modifications apportées à l'arborescence de rubriques affectent les applications de diffuseur de publications et d'abonné et comment vous pouvez réduire les effets. La notion d'objet de rubrique d'administration qui fournit la partie racine, ou sous-arborescence racine, d'une rubrique est intégrée dans l'architecture du modèle de publication / abonnement IBM MQ . L'objet de rubrique vous permet de définir la partie racine de l'arborescence de rubriques de manière administrative, ce qui simplifie la programmation et les opérations de l'application et améliore par conséquent la facilité de maintenance. Par exemple, si vous déployez plusieurs applications de publication / abonnement ayant des arborescences de rubriques isolées, en définissant administrativement la partie racine de l'arborescence de rubriques, vous pouvez garantir l'isolement des arborescences de rubriques, même s'il n'y a pas de cohérence dans les conventions de dénomination des rubriques adoptées par les différentes applications.

Dans la pratique, les applications d'éditeur couvrent un spectre allant de l'utilisation exclusive de sujets fixes, comme dans cet exemple, et de sujets variables, comme dans le suivant. «Exemple 2: diffuseur de publications vers une rubrique de variable», à la page 834 illustre également la combinaison de l'utilisation de rubriques et de chaînes de rubrique.

Concepts associés

«Exemple 2: diffuseur de publications vers une rubrique de variable», à la page 834
Un programme WebSphere MQ pour illustrer la publication dans une rubrique définie à l'aide d'un programme.

«Ecriture d'applications d'abonné», à la page 837
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application IBM MQ consommant des messages à partir d'une file d'attente, une application qui crée un abonnement et ne nécessite aucune connaissance de la mise en file d'attente, et enfin un exemple qui utilise à la fois la mise en file d'attente et les abonnements.

Exemple 2: diffuseur de publications vers une rubrique de variable
Un programme WebSphere MQ pour illustrer la publication dans une rubrique définie à l'aide d'un programme.

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

La sortie est illustrée Figure 67, à la page 835.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
    char     topicNameDefault[]   = "STOCKS";
    char     topicStringDefault[] = "IBM/PRICE";
    char     publicationDefault[] = "130";
    MQCHAR48 qmName = "";

    MQHCONN  Hconn = MQHC_UNUSABLE_HCONN; /* connection handle           */
    MQHOBJ   Hobj     = MQHO_NONE;   /* object handle sub queue          */
    MQLONG   CompCode = MQCC_OK;     /* completion code                  */
    MQLONG   Reason   = MQRC_NONE;   /* reason code                      */
    MQOD     td = {MQOD_DEFAULT};    /* Object descriptor                */
    MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor               */
    MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options              */
    MQCHAR   resTopicStr[151];       /* Returned value of topic string */
    char *   topicName   = topicNameDefault;
    char *   topicString = topicStringDefault;
    char *   publication = publicationDefault;
    memset   (resTopicStr, 0 , sizeof(resTopicStr));

    switch(argc){            /* Replace defaults with args if provided */
        default:
            publication = argv[3];
        case(3):
            topicString = argv[2];
        case(2):
            if (strcmp(argv[1],"/"))  /* "/" invalid = No topic object  */
                topicName = argv[1];
            else
                *topicName = '\0';
        case(1):
            printf("Provide parameters:  TopicObject TopicString Publication\n");
    }

    printf("Publish \"%s\" to topic \"%-.48s\" and topic string \"%s\"\n", publication, topicName,
topicString);
    do {
        MQCONN(qmName, &Hconn, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
        td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
        strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
        td.ObjectString.VSPtr = topicString;
        td.ObjectString.VSLength = (MQLONG)strlen(topicString);
        td.ResObjectString.VSPtr = resTopicStr;
        td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
        MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
        MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
        if (CompCode != MQCC_OK) break;
        MQDISC(&Hconn, &CompCode, &Reason);
    } while (0);
    if (CompCode == MQCC_OK)
        printf("Published \"%s\" to topic string \"%s\"\n",  publication, resTopicStr);
    printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figure 66. Publieur IBM MQ simple vers une rubrique de 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
```

Figure 67. Exemple de sortie du second diffuseur de publications

Il y a quelques points à noter à propos de cet exemple.

char topicNameDefault[] = "STOCKS";
Le nom de rubrique par défaut STOCKS définit une partie de la chaîne de rubrique. Vous pouvez remplacer ce nom de rubrique en le fournissant comme premier argument du programme, ou éliminer l'utilisation du nom de rubrique en fournissant / comme premier paramètre.

char topicString[101] = "IBM/PRICE";
IBM/PRICE est la chaîne de rubrique par défaut. Vous pouvez remplacer cette chaîne de rubrique en la fournissant en tant que deuxième argument du programme.

Le gestionnaire de files d'attente combine la chaîne de rubrique fournie par l'objet de rubrique STOCKS , "NYSE", avec la chaîne de rubrique fournie par le programme "IBM/PRICE" et insère un "/" entre les deux chaînes de rubrique. Le résultat est la chaîne de rubrique résolue "NYSE/IBM/PRICE". La chaîne de rubrique résultante est identique à celle définie dans l'objet de rubrique IBMSTOCKPRICE et a exactement le même effet.

L'objet de rubrique d'administration associé à la chaîne de rubrique résolue n'est pas nécessairement le même objet de rubrique que celui transmis à MQOPEN par le diffuseur de publications. IBM MQ utilise l'arborescence implicite dans la chaîne de rubrique résolue pour déterminer quel objet de rubrique d'administration définit les attributs associés à la publication.

Supposons qu'il existe deux objets de rubrique A et B, et que A définit la rubrique "a", et que B définit la rubrique "a/b" (Figure 68, à la page 836). Si le programme de publication fait référence à l'objet de rubrique A et fournit la chaîne de rubrique "b", en convertissant la rubrique en chaîne de rubrique "a/b", la publication hérite ses propriétés de l'objet de rubrique B car la rubrique correspond à la chaîne de rubrique "a/b" définie pour B.

if (strcmp(argv[1],"/"))
argv[1] est le topicNameéventuellement fourni. "/" n'est pas valide en tant que nom de rubrique ; ici, cela signifie qu'il n'y a pas de nom de rubrique et que la chaîne de rubrique est fournie entièrement par le programme. La sortie dans Figure 67, à la page 835 montre l'ensemble de la chaîne de rubrique fournie dynamiquement par le programme.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
Pour le cas par défaut, le topicName facultatif doit être créé à l'avance, à l'aide de l'explorateur IBM MQ ou de la commande MQSC suivante:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
La chaîne de rubrique est une zone MQCHARV dans le descripteur de rubrique

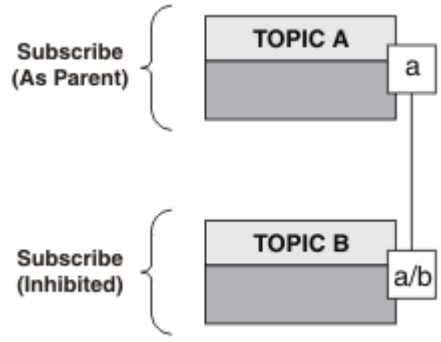


Figure 68. Associations d'objets de rubrique

Que démontre le deuxième exemple? Bien que le code soit très similaire au premier exemple-en fait, il n'y a que deux lignes de différence-le résultat est un programme sensiblement différent du premier. Le programmeur contrôle les destinations auxquelles les publications sont envoyées. En conjonction avec une entrée d'administrateur minimale utilisée pour concevoir des applications d'abonné, aucune

rubrique ou file d'attente n'a besoin d'être prédéfinie pour acheminer les publications des diffuseurs de publications vers les abonnés.

Dans le paradigme de la messagerie point-à-point, les files d'attente doivent être définies avant que les messages puissent circuler. Pour la publication / l'abonnement, ce n'est pas le cas, bien que IBM MQ implémente la publication / l'abonnement à l'aide de son système de mise en file d'attente sous-jacent ; les avantages de la distribution garantie, de la transactionnalité et du couplage souple associés à la messagerie et à la mise en file d'attente sont hérités par les applications de publication / abonnement.

Un concepteur doit décider si les programmes de diffuseur de publications et d'abonné doivent ou non connaître l'arborescence de rubriques sous-jacente, et si les programmes d'abonné doivent ou non connaître la mise en file d'attente. Etudier ensuite les exemples d'application d'abonné. Ils sont conçus pour être utilisés avec les exemples de diffuseur de publications, généralement la publication et l'abonnement à `NYSE/IBM/PRICE`.

Concepts associés

«Exemple 1: diffuseur de publications vers une rubrique fixe», à la page 831
Un programme IBM MQ pour illustrer la publication dans une rubrique définie par l'administrateur.

«Ecriture d'applications d'abonné», à la page 837
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application IBM MQ consommant des messages à partir d'une file d'attente, une application qui crée un abonnement et ne nécessite aucune connaissance de la mise en file d'attente, et enfin un exemple qui utilise à la fois la mise en file d'attente et les abonnements.

Ecriture d'applications d'abonné

Commencez par écrire des applications d'abonné en étudiant trois exemples: une application IBM MQ consommant des messages à partir d'une file d'attente, une application qui crée un abonnement et ne nécessite aucune connaissance de la mise en file d'attente, et enfin un exemple qui utilise à la fois la mise en file d'attente et les abonnements.

Dans Tableau 122, à la page 837 , les trois styles de consommateur ou d'abonné sont répertoriés, ainsi que les séquences d'appels de fonction IBM MQ qui les caractérisent.

1. Le premier style, MQ Publication Consumer, est identique à un programme MQ point à point qui n'exécute que MQGET. L'application ne sait pas qu'elle consomme des publications-elle lit simplement les messages d'une file d'attente. L'abonnement qui entraîne le routage des publications vers la file d'attente est créé de manière administrative à l'aide de IBM MQ Explorer ou d'une commande.
2. Le second style est le modèle préféré pour la plupart des applications d'abonné. L'application d'abonné crée l'abonnement, puis obtient les publications. La gestion des files d'attente est effectuée par le gestionnaire de files d'attente. Il s'agit d'un *abonné géré*.
3. Dans le troisième style, l'application d'abonné est chargée de spécifier la file d'attente qui sera utilisée pour stocker les publications, d'ouvrir et de fermer cette file d'attente et d'émettre des abonnements pour remplir la file d'attente avec des publications. Il s'agit d'un *abonné non géré*.

Une façon de comprendre ces styles consiste à étudier les exemples de programmes C répertoriés dans Tableau 122, à la page 837 pour chacun des styles. Les exemples sont conçus pour être exécutés conjointement avec l'exemple de diffuseur de publications disponible dans «Ecriture d'applications de publication», à la page 830.

Tableau 122. Modèles de programme IBM MQ point à point vs abonnement.

Etape	Consommateur de message MQ	«Exemple 1: consommateur de publication MQ», à la page 838	«Exemple 2: abonné MQ géré», à la page 841	«Exemple 3: Abonné MQ non géré», à la page 846
Connexion à un gestionnaire de files d'attente	MQCONN	MQCONN	MQCONN	MQCONN

Tableau 122. Modèles de programme IBM MQ point à point vs abonnement. (suite)				
Etape	**Consommateur de message MQ**	**«Exemple 1: consommateur de publication MQ», à la page 838**	**«Exemple 2: abonné MQ géré», à la page 841**	**«Exemple 3: Abonné MQ non géré», à la page 846**
Ouvrir la file d'attente	MQOPEN	MQOPEN		MQOPEN
S'abonner			MQSUB	MQSUB
Obtenir le (s) message (s)	MQGET	MQGET	MQGET	MQGET
Fermer la file d'attente	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Fermer l'abonnement			MQCLOSE	MQCLOSE
Se déconnecter du gestionnaire de files d'attente	MQDISC	MQDISC	MQDISC	MQDISC

L'utilisation de MQCLOSE est toujours facultative, soit pour libérer des ressources, soit pour transmettre des options MQCLOSE, soit uniquement pour la symétrie avec MQOPEN. Etant donné qu'il est peu probable que vous ayez à spécifier les options MQCLOSE lorsque la file d'attente d'abonnement est fermée dans le cas d'abonné MQ géré et que l'argument de symétrie n'est pas pertinent, la file d'attente d'abonnement n'est pas explicitement fermée dans l' exemple 2: abonné MQ géré.

Une autre façon de comprendre les modèles d'application de publication / abonnement consiste à examiner les interactions entre les différentes entités impliquées. Les diagrammes de séquence lifeline ou UML sont une bonne façon d'étudier les interactions. Trois exemples de ligne de vie sont décrits dans «Cycles de vie de publication / abonnement», à la page 856.

Exemple 1: consommateur de publication MQ
Le consommateur de publication MQ est un consommateur de message IBM MQ qui ne s'abonne pas aux rubriques lui-même.

Pour créer la file d'attente d'abonnement et de publication pour cet exemple, exécutez les commandes suivantes ou définissez les objets à l'aide de IBM MQ Explorer.

```
DEFINE QLOCAL(STOCKTICKER) REPLACE;
DEFINE SUB(IBMSTOCKPRICESUB) DEST(STOCKTICKER) TOPICOBJ(IBMSTOCKPRICE) REPLACE;
```

L'abonnement `IBMSTOCKPRICESUB` fait référence à l'objet de rubrique `IBMSTOCK` créé pour l'exemple de diffuseur de publications et la file d'attente locale `STOCKTICKER`. L'objet de rubrique `IBMSTOCK` définit la chaîne de rubrique utilisée dans l'abonnement, `NYSE/IBM/PRICE`. Notez que l'objet de rubrique et la file d'attente utilisée pour recevoir les publications doivent être définis avant la création de l'abonnement.

Le modèle de consommateur de publication MQ comporte un certain nombre de facettes intéressantes:

1. Multitraitement: partage du travail de lecture des publications. Les publications sont toutes placées dans la file d'attente unique associée à la rubrique d'abonnement. Plusieurs destinataires peuvent ouvrir la file d'attente à l'aide de `MQOO_INPUT_SHARED`.
2. Abonnements gérés de manière centralisée. Les applications ne construisent pas leurs propres rubriques d'abonnement ou abonnements ; l'administrateur est responsable de l'emplacement où les publications sont envoyées.

3. Concentration des abonnements: plusieurs abonnements différents peuvent être envoyés à une seule file d'attente.
4. Durabilité de l'abonnement: la file d'attente reçoit toutes les publications, que les consommateurs soient actifs ou non.
5. Migration et coexistence: le code consommateur fonctionne aussi bien pour un scénario de point à point que pour un scénario de publication / abonnement.

L'abonnement crée une relation entre la chaîne de rubrique `NYSE/IBM/PRICE` et la file d'attente `STOCKTICKER`. Les publications, y compris les publications actuellement conservées, sont transmises à `STOCKTICKER` dès la création de l'abonnement.

Un abonnement créé administrativement peut être géré ou non géré. Un abonnement géré prend effet dès qu'il a été créé, tout comme un abonnement non géré. Toutes les facettes de canevas ne sont pas disponibles pour un abonnement géré. Voir le «Exemple 3: Abonné MQ non géré», à la page 846

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

Les résultats sont affichés dans Figure 70, à la page 840.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>
int main(int argc, char **argv)
{
MQCHAR     publicationBuffer[101];
MQCHAR48 subscriptionQueueDefault = "STOCKTICKER";
MQCHAR48 qmName = "";            /* Use default queue manager  */

MQHCONN Hconn = MQHC_UNUSABLE_HCONN;    /* connection handle      */
MQHOBJ  Hobj = MQHO_NONE;         /* object handle sub queue   */
MQLONG  CompCode = MQCC_OK;        /* completion code       */
MQLONG  Reason = MQRC_NONE;        /* reason code         */
MQLONG  messlen = 0;
MQOD   od = {MQOD_DEFAULT};        /* Unmanaged subscription queue */
MQMD   md = {MQMD_DEFAULT};        /* Message Descriptor      */
MQGMO  gmo = {MQGMO_DEFAULT};       /* Get message options     */
char *  publication=publicationBuffer;
char *  subscriptionQueue = subscriptionQueueDefault;

switch(argc){           /* Replace defaults with args if provided */
default:
subscriptionQueue = argv[1]
case(1):
printf("Optional parameter: subscriptionQueue\n");
}

do {
MQCONN(qmName, &Hconn, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
strncpy(od.ObjectName, subscriptionQueue, MQ_Q_NAME_LENGTH);
MQOPEN(Hconn, &od, MQOO_INPUT_AS_Q_DEF | MQOO_FAIL_IF_QUIESCING , &Hobj, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
gmo.Options = MQGMO_WAIT | MQGMO_NO_SYNCPOINT | MQGMO_CONVERT;
gmo.WaitInterval = 10000;
printf("Waiting %d seconds for publications from %s\n", gmo.WaitInterval/1000,
subscriptionQueue);
do {
memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
md.Encoding    = MQENC_NATIVE;
md.CodedCharSetId = MQCCSI_Q_MGR;
memset(publication, 0, sizeof(publicationBuffer));
MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen,
&CompCode, &Reason);
if (Reason == MQRC_NONE)
printf("Received publication \"%s\"\n", publication);
}
while (CompCode == MQCC_OK);
if (CompCode != MQCC_OK && Reason != MQRC_NO_MSG_AVAILABLE) break;
MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
if (CompCode != MQCC_OK) break;
MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
printf("Completion code %d and Return code %d\n", CompCode, Reason);
}
```

Figure 69. Consommateur de publication 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
```

Figure 70. Sortie du consommateur de publication MQ

Il existe quelques conseils de programmation de langage IBM MQ *C* standard à prendre en compte:

memset(publication, 0, sizeof(publicationBuffer));
Assurez-vous que le message comporte une valeur NULL de fin pour faciliter le formatage à l'aide de `printf`. L'exemple de diffuseur de publications inclut la valeur NULL de fin dans la mémoire tampon de messages transmise à MQPUT en ajoutant 1 à `strlen(publication)`. La définition de la valeur null pour les mémoires tampon MQCHAR est un bon style de programmation pour les programmes IBM MQ C qui utilisent les mémoires tampon pour stocker des chaînes, ce qui garantit qu'une valeur null suit un tableau de caractères qui ne remplit pas complètement la mémoire tampon.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
Réservez une valeur null à la fin de la mémoire tampon de messages pour vous assurer que le message renvoyé a la valeur null de fin si `if (messlen == strlen(publication));` a la valeur true. Cette astuce complète la précédente et garantit qu'il existe au moins une valeur nulle dans `publicationBuffer` qui n'est pas écrasée par le contenu de `publication`.

Concepts associés

«Exemple 2: abonné MQ géré», à la page 841
L'abonné MQ géré est le modèle préféré pour la plupart des applications d'abonné. Un abonnement géré est un abonnement dans lequel IBM MQ gère l'abonnement et effectue l'enregistrement et le désenregistrement pour vous. L'exemple requiert *aucune* définition d'administration des files d'attente, des rubriques ou des abonnements.

«Exemple 3: Abonné MQ non géré», à la page 846
L'abonné non géré est une classe importante de l'application d'abonné. Avec elle, vous combinez les avantages de la publication / abonnement avec le *contrôle* de la mise en file d'attente et de la consommation des publications. Un abonnement non géré est l'endroit où l'application est responsable. pour spécifier la file d'attente dans laquelle les abonnements sont stockés. L'exemple illustre différentes manières de combiner des abonnements et des files d'attente.

«Ecriture d'applications de publication», à la page 830
Commencez par écrire des applications d'éditeur en étudiant deux exemples. La première est modélisée de la manière la plus proche possible sur une application point à point en plaçant des messages dans une file d'attente, et la seconde illustre la création dynamique de rubriques-un modèle plus courant pour les applications de diffuseur de publications.

Exemple 2: abonné MQ géré

L'abonné MQ géré est le modèle préféré pour la plupart des applications d'abonné. Un abonnement géré est un abonnement dans lequel IBM MQ gère l'abonnement et effectue l'enregistrement et le désenregistrement pour vous. L'exemple requiert *aucune* définition d'administration des files d'attente, des rubriques ou des abonnements.

Ce type d'abonné géré le plus simple utilise généralement un abonnement *non durable* . L'exemple se concentre sur un abonnement non durable. L'abonnement ne dure que pendant la durée de vie du descripteur d'abonnement à partir de MQSUB. Toutes les publications qui correspondent à la chaîne de rubrique pendant la durée de vie de l'abonnement sont envoyées à la file d'attente d'abonnement (et éventuellement à une publication conservée si l'indicateur MQSO_NEW_PUBLICATIONS_ONLY n'est pas défini ou défini par défaut, une publication antérieure correspondant à la chaîne de rubrique a été conservée et la publication a été persistante ou le gestionnaire de files d'attente ne s'est pas arrêté depuis la création de la publication).

Vous pouvez également utiliser un abonnement *durable* avec ce modèle. Généralement, si un abonnement durable géré est utilisé, il est effectué pour des raisons de fiabilité, plutôt que pour établir un abonnement qui, sans qu'aucune erreur ne se produise, survienne à l'abonné. Pour plus d'informations sur les différents cycles de vie associés à des abonnements gérés, non gérés, durables et non durables, voir la section relative aux rubriques connexes.

Les abonnements durables sont souvent associés à des publications persistantes, et les abonnements non durables à des publications non persistantes, mais il n'y a pas de relation nécessaire entre la durabilité des abonnements et la persistance des publications. Les quatre combinaisons de persistance et de durabilité sont possibles.

Pour le cas non durable géré pris en compte, le gestionnaire de files d'attente crée une file d'attente d'abonnement qui est purgée et supprimée lorsque la file d'attente est fermée. Les publications sont supprimées de la file d'attente lorsque l'abonnement non durable est fermé.

Les facettes précieuses du modèle non durable géré illustré par ce code sont les suivantes:

1. Abonnement à la demande: la chaîne de rubrique de l'abonnement est dynamique. Il est fourni par l'application lors de son exécution.
2. File d'attente d'auto-gestion: la file d'attente d'abonnement est auto-définie et gérée.
3. Gestion automatique du cycle de vie des abonnements: les abonnements *non durables* n'existent que pendant la durée de l'application d'abonné.
   - Si vous définissez un abonnement géré *durable* , il se traduit par une file d'attente d'abonnement permanente et les publications continuent d'y être stockées sans qu'aucun programme d'abonné ne soit actif. Le gestionnaire de files d'attente supprime la file d'attente (et efface toutes les publications non extraites) uniquement après que l'application ou l'administrateur a choisi de supprimer l'abonnement. L'abonnement peut être supprimé à l'aide d'une commande d'administration ou en fermant l'abonnement avec l'option `MQCO_REMOVE_SUB` .
   - Envisagez de définir `SubExpiry` pour les abonnements durables afin que les publications cessent d'être envoyées à la file d'attente et que l'abonné puisse consommer toutes les publications restantes avant de supprimer l'abonnement et de faire en sorte que le gestionnaire de files d'attente supprime la file d'attente et les publications restantes qu'elle contient.
4. Déploiement flexible des chaînes de rubrique: la gestion des rubriques d'abonnement est simplifiée en définissant la partie racine de l'abonnement à l'aide d'une rubrique définie par l'administrateur. La partie racine de l'arborescence de rubriques est ensuite masquée dans l'application. En masquant la partie racine, une application peut être déployée sans que l'application crée par inadvertance une arborescence de rubriques qui chevauche une autre arborescence de rubriques créée par une autre instance ou une autre application.
5. Rubriques administrées: en utilisant une chaîne de rubrique dans laquelle la première partie correspond à un objet de rubrique défini par l'administrateur, les publications sont gérées en fonction des attributs de l'objet de rubrique.
   - Par exemple, si la première partie de la chaîne de rubrique correspond à la chaîne de rubrique associée à un objet de rubrique en cluster, l'abonnement peut recevoir des publications d'autres membres du cluster
   - La mise en correspondance sélective des objets de rubrique définis de manière administrative et des abonnements définis à l'aide d'un programme vous permet de combiner les avantages des deux. L'administrateur fournit des attributs pour les rubriques et le programmeur définit dynamiquement des sous-rubriques sans se soucier de la gestion des rubriques.
   - Il s'agit de la chaîne de rubrique résultante qui est utilisée pour correspondre à l'objet de rubrique qui fournit les attributs associés à la rubrique, et pas nécessairement l'objet de rubrique nommé dans `sd.Objectname`, bien qu'il s'agisse généralement d'un seul et même objet. Voir «Exemple 2: diffuseur de publications vers une rubrique de variable», à la page 834.

En rendant l'abonnement durable dans l'exemple, les publications continuent d'être envoyées à la file d'attente d'abonnement une fois que l'abonné a fermé l'abonnement avec l'option `MQCO_KEEP_SUB` . La file d'attente continue de recevoir des publications lorsque l'abonné n'est pas actif. Vous pouvez remplacer ce comportement en créant l'abonnement avec l'option `MQSO_PUBLICATIONS_ON_REQUEST` et en utilisant `MQSUBRQ` pour demander la publication conservée.

L'abonnement peut être repris ultérieurement en ouvrant l'abonnement avec l'option `MQCO_RESUME` .

Vous pouvez utiliser l'identificateur de file d'attente, `Hobj`, renvoyé par `MQSUB` de différentes manières. Le descripteur de file d'attente est utilisé dans l'exemple pour interroger le nom de la file d'attente d'abonnement. Les files d'attente gérées sont ouvertes à l'aide des files d'attente modèles par défaut `SYSTEM.NDURABLE.MODEL.QUEUE` ou `SYSTEM.DURABLE.MODEL.QUEUE`. Vous pouvez remplacer les valeurs par défaut en fournissant vos propres files d'attente modèles durables et non durables, rubrique par rubrique, en tant que propriétés de l'objet de rubrique associé à l'abonnement.

Quels que soient les attributs hérités des files d'attente modèles, vous ne pouvez pas réutiliser un descripteur de file d'attente gérée pour créer un abonnement supplémentaire. Vous ne pouvez pas non plus obtenir un autre descripteur pour la file d'attente gérée en ouvrant la file d'attente gérée une seconde fois à l'aide du nom de la file d'attente renvoyée. La file d'attente se comporte comme si elle avait été ouverte en entrée exclusive.

Les files d'attente non gérées sont plus flexibles que les files d'attente gérées. Vous pouvez, par exemple, partager des files d'attente non gérées ou définir plusieurs abonnements sur une seule file d'attente. L'exemple suivant montre comment combiner des abonnements avec une file d'attente d'abonnement non gérée.

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

Les résultats sont affichés dans Figure 73, à la page 845.

```
#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);
}
```

Figure 71. Abonné MQ géré-Partie 1: déclarations et gestion des paramètres.

Il y a des commentaires supplémentaires à faire sur les déclarations de cet exemple.

MQHOBJ Hobj = MQHO_NONE;
Vous ne pouvez pas ouvrir explicitement une file d'attente d'abonnement géré non durable pour recevoir des publications, mais vous devez allouer de la mémoire pour le descripteur d'objet renvoyé par le gestionnaire de files d'attente lorsqu'il ouvre la file d'attente pour vous. Il est important d'initialiser le descripteur dans MQHO_OBJECT. Indique au gestionnaire de files d'attente qu'il doit renvoyer un descripteur de file d'attente à la file d'attente d'abonnement.

MQSD sd = {MQSD_DEFAULT};
Le nouveau descripteur d'abonnement, utilisé dans MQSUB.

MQCHAR48 qName;
Bien que l'exemple ne nécessite pas de connaissance de la file d'attente d'abonnement, l'exemple demande le nom de la file d'attente d'abonnement-la liaison MQINQ est un peu gênante dans le langage C, de sorte que vous pouvez trouver cette partie de l'exemple utile à étudier.

```
    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;
}
```

Figure 72. Abonné MQ géré-Partie 2: corps du code.

```
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
```

Figure 73. Abonné MQ

Des commentaires supplémentaires doivent être faits sur le code dans cet exemple.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Si topicName est null ou vide *(valeur par défaut)*, le nom de rubrique n'est pas utilisé pour calculer la chaîne de rubrique résolue.

sd.ObjectString.VSPtr = topicString;
Plutôt que d'utiliser uniquement un objet de rubrique prédéfini, dans cet exemple, le programmeur fournit un objet de rubrique et une chaîne de rubrique, qui sont combinés par MQSUB. Notez que la chaîne de rubrique est une structure MQCHARV .

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Alternative à la définition de la longueur d'une zone MQCHARV .

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Après avoir défini la chaîne de rubrique, les indicateurs sd.Options doivent faire l'objet d'une attention particulière. Il existe de nombreuses options, l'exemple ne spécifie que celles qui sont les plus couramment utilisées. Les autres options utilisent les valeurs par défaut.

1. Comme l'abonnement est *non durable*, c'est-à-dire qu'il a une durée de vie de l'abonnement ouvert dans l'application, définissez l'indicateur MQSO_CREATE . Vous pouvez également définir l'indicateur *(par défaut)* MQSO_NON_DURABLE pour la lisibilité.
2. En complément de MQSO_CREATE , MQSO_RESUME. Les deux indicateurs peuvent être définis ensemble ; le gestionnaire de files d'attente crée un nouvel abonnement ou reprend un abonnement existant, selon le cas. Toutefois, si vous spécifiez MQSO_RESUME , vous devez également initialiser la structure MQCHARV pour sd.SubName, même s'il n'y a pas d'abonnement à reprendre. L'échec de l'initialisation de SubName génère le code retour 2440: MQRC_SUB_NAME_ERROR à partir de MQSUB.

   Remarque : MQSO_RESUME est toujours ignoré pour un abonnement géré non durable, mais le fait de le spécifier sans initialiser la structure MQCHARV pour sd.SubName provoque l'erreur.
3. En outre, un troisième indicateur affecte la manière dont l'abonnement est ouvert, MQSO_ALTER. Compte tenu des droits d'accès corrects, les propriétés d'un abonnement repris sont modifiées pour correspondre à d'autres attributs spécifiés dans MQSUB.

   Remarque : Au moins l'un des indicateurs MQSO_CREATE, MQSO_RESUME et MQSO_ALTER doit être spécifié. Voir Options (MQLONG). Il existe des exemples d'utilisation des trois indicateurs dans «Exemple 3: Abonné MQ non géré», à la page 846.
4. Définissez MQSO_MANAGED pour que le gestionnaire de files d'attente gère automatiquement l'abonnement pour vous.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Si vous le souhaitez, omettez de définir la longueur de MQCHARV pour les chaînes à terminaison nulle et utilisez l'indicateur de terminaison nulle à la place.

sd.ResObjectString.VSPtr = resTopicStr;
La chaîne de rubrique résultante est répercutée dans `printf` en premier dans le programme. Configurez `MQCHARV ResObjectString` pour IBM MQ afin de renvoyer la chaîne résolue au programme.

Remarque : `resTopicStringBuffer` est initialisé avec des valeurs nulles dans `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Les chaînes de rubrique renvoyées ne se terminent pas par une valeur NULL de fin.

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
Définissez la taille de la mémoire tampon du `sd.ResObjectString` sur une valeur inférieure à sa taille réelle. Cela permet d'éviter le remplacement du caractère de fin null fourni, au cas où la chaîne de rubrique résolue remplisse la totalité de la mémoire tampon.

Remarque : Aucune erreur n'est renvoyée si la chaîne de rubrique est plus longue que `sizeof(resTopicStrBuffer)-1`. Même si `VSLength > VSBufSiz` , la longueur renvoyée dans `sd.ResObjectString.VSLength` correspond à la longueur de la chaîne complète et pas nécessairement à la longueur de la chaîne renvoyée. Testez `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` pour confirmer que la chaîne de rubrique est terminée.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
La fonction `MQSUB` crée un abonnement. S'il n'est pas durable, vous n'êtes probablement pas intéressé par son nom, mais vous pouvez inspecter son statut dans IBM MQ Explorer. Vous pouvez fournir le paramètre `sd.SubName` en tant qu'entrée, afin de savoir quel nom rechercher ; vous devez évidemment éviter les conflits de nom avec d'autres abonnements.

MQCLOSE(Hconn, &Hsub, MQCO_REMOVE_SUB, &CompCode, &Reason);
La fermeture de l'abonnement et de la file d'attente d'abonnement est facultative. Dans l'exemple, l'abonnement est fermé, mais pas la file d'attente. Dans ce cas, l'option `MQCLOSE MQCO_REMOVE_SUB` est la valeur par défaut car l'abonnement est non durable. L'utilisation de `MQCO_KEEP_SUB` est une erreur.

Remarque : la *file d'attente* d'abonnement n'est pas fermée par `MQSUBet` son descripteur, `Hobj`, reste valide jusqu'à ce que la file d'attente soit fermée par `MQCLOSE` ou `MQDISC`. Si l'application s'arrête prématurément, la file d'attente et l'abonnement sont nettoyés par le gestionnaire de files d'attente quelque temps après l'arrêt de l'application.

Concepts associés

«Exemple 1: consommateur de publication MQ», à la page 838
Le consommateur de publication MQ est un consommateur de message IBM MQ qui ne s'abonne pas aux rubriques lui-même.

«Exemple 3: Abonné MQ non géré», à la page 846
L'abonné non géré est une classe importante de l'application d'abonné. Avec elle, vous combinez les avantages de la publication / abonnement avec le *contrôle* de la mise en file d'attente et de la consommation des publications. Un abonnement non géré est l'endroit où l'application est responsable. pour spécifier la file d'attente dans laquelle les abonnements sont stockés. L'exemple illustre différentes manières de combiner des abonnements et des files d'attente.

«Ecriture d'applications de publication», à la page 830
Commencez par écrire des applications d'éditeur en étudiant deux exemples. La première est modélisée de la manière la plus proche possible sur une application point à point en plaçant des messages dans une file d'attente, et la seconde illustre la création dynamique de rubriques-un modèle plus courant pour les applications de diffuseur de publications.

Exemple 3: Abonné MQ non géré
L'abonné non géré est une classe importante de l'application d'abonné. Avec elle, vous combinez les avantages de la publication / abonnement avec le *contrôle* de la mise en file d'attente et de la consommation des publications. Un abonnement non géré est l'endroit où l'application est responsable. pour spécifier la file d'attente dans laquelle les abonnements sont stockés. L'exemple illustre différentes manières de combiner des abonnements et des files d'attente.

Le canevas non géré est plus souvent associé à des abonnements *durables* qu'à des abonnements *non durables*. Généralement, le cycle de vie d'un abonnement créé par un abonné non géré est indépendant du cycle de vie de l'application d'abonnement elle-même. En rendant l'abonnement durable, l'abonnement reçoit des publications même lorsqu'aucune application d'abonnement n'est active.

Vous pouvez créer des abonnements *gérés* durables pour obtenir le même résultat, mais certaines applications requièrent plus de flexibilité et de contrôle sur les files d'attente et les messages qu'avec un abonnement géré. Pour un abonnement géré durable, le gestionnaire de files d'attente crée une file d'attente permanente pour les publications qui correspondent à la rubrique d'abonnement. Elle supprime la file d'attente et les publications associées lorsque l'abonnement est supprimé.

Généralement, les abonnements *gérés* durables sont utilisés si le cycle de vie de l'application et de l'abonnement est essentiellement le même, mais difficile à garantir. En rendant l'abonnement durable et en demandant au diffuseur de publications de créer des publications persistantes, il n'y a pas de messages perdus si le gestionnaire de files d'attente ou l'abonné s'arrête prématurément et doit être récupéré.

Pour les applications non JMS ou les applications JMS qui n'utilisent pas d'abonnement partagé, le gestionnaire de files d'attente ouvre implicitement la file d'attente d'abonnement géré durable pour un abonné de sorte que le traitement partagé de la file d'attente ne soit pas possible. En outre, à moins que votre application n'utilise des abonnements partagés JMS, il n'est pas possible de créer plus d'un abonnement pour chaque file d'attente gérée et vous risquez de trouver les files d'attente plus difficiles à gérer car vous avez moins de contrôle sur les noms des files d'attente. Pour ces raisons, déterminez si l'abonné *non géré* MQ est mieux adapté aux applications nécessitant des abonnements durables que l'abonné *géré* MQ .

Le code dans Figure 76, à la page 853 illustre un modèle d'abonnement durable non géré. A des fins d'illustration, le code crée également des abonnements non gérés et non durables. Cet exemple illustre les facettes de canevas suivantes:

- Abonnements à la demande: les chaînes de rubrique d'abonnement sont dynamiques. Ils sont fournis par l'application lors de son exécution.
- Gestion de rubrique d'abonnement simplifiée: la gestion de rubrique d'abonnement est simplifiée en définissant la partie racine de la chaîne de rubrique d'abonnement à l'aide d'une rubrique définie par l'administrateur. La partie racine de l'arborescence de rubriques est masqué dans l'application. En masquant la partie racine, un abonné peut être déployé dans différentes arborescences de rubriques.
- Gestion flexible des abonnements: vous pouvez définir un abonnement de manière administrative ou le créer à la demande dans un programme d'abonné. Il n'y a pas de différence entre les abonnements créés à l'aide d'un programme et les abonnements créés à l'aide d'un programme, à l'exception d'un attribut qui indique comment l'abonnement a été créé. Il existe un troisième type d'abonnement qui est créé automatiquement par le gestionnaire de files d'attente pour la distribution des abonnements. Tous les abonnements sont affichés dans l'explorateur IBM MQ .
- Association flexible des abonnements avec des files d'attente: une file d'attente locale prédéfinie est associée à un abonnement par la fonction MQSUB . Il existe différentes façons d'utiliser MQSUB pour associer des abonnements à des files d'attente:
  - Associez un abonnement à une file d'attente ne comportant *aucun* abonnement existant, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Associez un *nouvel* abonnement à une file d'attente comportant des abonnements existants, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Déplacez un abonnement existant dans une autre file d'attente, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Reprenez un abonnement existant associé à une file d'attente existante, `MQSO_RESUME + (Hobj = MQHO_NONE)` ou `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - En combinant `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` dans différentes combinaisons, vous pouvez prendre en compte les différents états d'entrée de l'abonnement et de la file d'attente sans avoir à coder plusieurs versions de MQSUB avec des valeurs `sd.Options` différentes.

– Sinon, en codant un choix spécifique de MQSO_CREATE | MQSO_RESUME | MQSO_ALTER , le gestionnaire de files d'attente renvoie une erreur (Tableau 123, à la page 849) si les états de l'abonnement et de la file d'attente fournis en entrée de MQSUB sont incohérents avec la valeur de sd.Options. Le Figure 82, à la page 856 montre les résultats de l'émission de MQSUB pour l'abonnement X avec différents paramètres individuels de l'indicateur sd.Options et la transmission de trois descripteurs d'objet différents.

Explorez les différentes entrées de l'exemple de programme dans Figure 75, à la page 852 pour vous familiariser avec ces différents types d'erreurs. Une erreur courante, RC = 2440, qui n'est pas incluse dans les cas répertoriés dans le tableau, est une erreur de nom d'abonnement. il est généralement dû à la transmission d'un nom d'abonnement null ou non valide avec MQSO_RESUME ou MQSO_ALTER.

- Multitraitement: vous pouvez partager entre de nombreux consommateurs le travail de lecture des publications. Les publications sont toutes placées dans la file d'attente unique associée à la rubrique d'abonnement. Les consommateurs ont le choix d'ouvrir la file d'attente directement à l'aide de MQOPEN ou de reprendre l'abonnement à l'aide de MQSUB.
- Concentration d'abonnements: plusieurs abonnements peuvent être créés dans la même file d'attente. Soyez prudent avec cette fonctionnalité car elle peut entraîner un chevauchement des abonnements et la réception de la même publication plusieurs fois. L'option MQSO_GROUP_SUB élimine les publications en double causées par des abonnements qui se chevauchent.
- Séparation de l'abonné et du consommateur: outre les trois modèles de consommateur illustrés dans les exemples, un autre modèle consiste à séparer le consommateur de l'abonné. Il s'agit d'une variante de l'abonné MQ non géré, mais plutôt que d'émettre les MQOPEN et les MQSUB dans le même programme, un programme s'abonne aux publications et un autre programme les consomme. Par exemple, l'abonné peut faire partie d'un cluster de publication / abonnement et le consommateur peut être connecté à un gestionnaire de files d'attente en dehors du cluster de gestionnaires de files d'attente. Le consommateur reçoit les publications via la mise en file d'attente répartie standard en définissant la file d'attente d'abonnement en tant que définition de file d'attente éloignée.

Il est important de comprendre le comportement de MQSO_CREATE | MQSO_RESUME | MQSO_ALTER , en particulier si vous prévoyez de simplifier votre code en utilisant des combinaisons de ces options. Consultez le tableau Tableau 123, à la page 849 qui présente les résultats de la transmission de différents descripteurs de file d'attente à MQSUB et les résultats de l'exécution de l'exemple de programme présenté dans Figure 77, à la page 854 à Figure 82, à la page 856.

Le scénario utilisé pour construire la table comporte un abonnement X et deux files d'attente, A et B. Le paramètre de nom d'abonnement sd.SubName est défini sur X, le nom d'un abonnement associé à la file d'attente A. Aucun abonnement n'est associé à la file d'attente B .

Dans Tableau 123, à la page 849, MQSUB est transmis à l'abonnement X et le descripteur de file d'attente à la file d'attente A. Les résultats des options d'abonnement sont les suivants:

- MQSO_CREATE échoue car le descripteur de file d'attente correspond à la file d'attente A qui possède déjà un abonnement à X. Comparez ce comportement à l'appel réussi. Cet appel aboutit car la file d'attente B n'est pas associée à un abonnement à X .
- MQSO_RESUME aboutit car l'identificateur de file d'attente correspond à la file d'attente A qui possède déjà un abonnement à X. En revanche, l'appel échoue lorsque l'abonnement X n'existe pas dans la file d'attente A.
- MQSO_ALTER se comporte de la même manière que MQSO_RESUME en ce qui concerne l'ouverture de l'abonnement et de la file d'attente. Toutefois, si les attributs contenus dans le descripteur d'abonnement transmis à MQSUB diffèrent des attributs de l'abonnement, MQSO_RESUME échoue, tandis que MQSO_ALTER réussit tant que l'instance de programme a le droit de modifier les attributs. Notez que vous ne pouvez jamais modifier la chaîne de rubrique dans un abonnement ; au lieu de renvoyer une erreur, MQSUB ignore les valeurs de nom de rubrique et de chaîne de rubrique dans le descripteur d'abonnement et utilise les valeurs de l'abonnement existant.

Ensuite, regardez Tableau 123, à la page 849 où MQSUB est transmis à l'abonnement X et le descripteur de file d'attente à la file d'attente B. Les résultats des options d'abonnement sont les suivants:

- MQSO_CREATE réussit et crée l'abonnement X dans la file d'attente B car il s'agit d'un nouvel abonnement dans la file d'attente B.
- MQSO_RESUME échoue. MQSUB recherche l'abonnement X dans la file d'attente B et ne le trouve pas, mais au lieu de renvoyer *RC = 2428-l'abonnement X n'existe pas*, il renvoie *RC = 2019-La file d'attente d'abonnement ne correspond pas à l'identificateur d'objet de file d'attente*. Le comportement de la troisième option MQSO_ALTER suggère la raison de cette erreur inattendue. MQSUB attend que le descripteur de file d'attente pointe vers une file d'attente avec un abonnement. Il le vérifie d'abord avant de vérifier si l'abonnement nommé dans sd.SubName existe.
- MQSO_ALTER réussit et déplace l'abonnement de la file d'attente A vers la file d'attente B.

Un cas qui ne s'affiche pas dans le tableau est celui où le nom d'abonnement de l'abonnement dans la file d'attente A ne correspond pas au nom d'abonnement dans sd.SubName. Cet appel échoue avec un *RC = 2428-l'abonnement X n'existe pas dans la file d'attente A*.

Tableau 123. Erreurs de MQSUB avec des descripteurs de file d'attente et des combinaisons d'abonnements différents

Descripteurs de file d'attente	**File d'attente A** **Abonnement X** **File d'attente B** **Aucun abonnement**	**File d'attente A** **Aucun abonnement** **File d'attente B** **Aucun abonnement**
Hobj pour la *file d'attente A* transmise à MQSUB	**MQSO_CREATE** RC = 2432-L'abonnement X existe déjà dans la file d'attente A **MQSO_RESUME** Reprend l'abonnement X dans la file d'attente A **MQSO_ALTER** Reprend l'abonnement X sur la file d'attente A et effectue les modifications autorisées	**MQSO_CREATE** Crée l'abonnement X dans la file d'attente A **MQSO_RESUME** RC = 2428-L'abonnement X n'existe pas dans la file d'attente A **MQSO_ALTER** RC = 2428-L'abonnement X n'existe pas dans la file d'attente A
Hobj pour la *file d'attente B* transmise à MQSUB	**MQSO_CREATE** Crée un abonnement X sur la file d'attente B **MQSO_RESUME** RC = 2019-La file d'attente d'abonnement ne correspond pas à l'identificateur d'objet de file d'attente **MQSO_ALTER** Déplacer l'abonnement X de la file d'attente A vers la file d'attente B	**MQSO_CREATE** Crée un abonnement X sur la file d'attente B **MQSO_RESUME** RC = 2428-L'abonnement X n'existe pas dans la file d'attente B **MQSO_ALTER** RC = 2428-L'abonnement X n'existe pas dans la file d'attente B

Tableau 123. Erreurs de MQSUB avec des descripteurs de file d'attente et des combinaisons d'abonnements différents (suite)

Descripteurs de file d'attente	**File d'attente A** **Abonnement X** **File d'attente B** **Aucun abonnement**	**File d'attente A** **Aucun abonnement** **File d'attente B** **Aucun abonnement**
MQHO_NONE transmis à MQSUB	**MQSO_CREATE** RC = 2019-Descripteur d'objet incorrect: définissez l'indicateur MQSO_MANAGED pour créer un abonnement géré et une file d'attente gérée **MQSO_RESUME** Reprend l'abonnement X dans la file d'attente A et renvoie Hobj à la file d'attente A **MQSO_ALTER** Reprend l'abonnement X dans la file d'attente A, renvoie Hobj à la file d'attente A et effectue les modifications autorisées	**MQSO_CREATE** RC = 2019-Descripteur d'objet incorrect: définissez l'indicateur MQSO_MANAGED pour créer un abonnement géré et une file d'attente gérée **MQSO_RESUME** RC = 2428-Pas d'abonnement X **MQSO_ALTER** RC = 2019-Descripteur d'objet incorrect: aucune file d'attente A ou B

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

```
#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));
```

Figure 74. Abonné MQ non géré-partie 1: déclarations.

```
    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);
}
```

Figure 75. Abonné MQ non géré-partie 2: gestion des paramètres.

Les commentaires supplémentaires sur le traitement des paramètres dans cet exemple sont les suivants:

switch((argv[5][0]))
Vous avez le choix d'entrer A lter | C reate | R esume dans le paramètre 5 pour tester l'effet du remplacement d'une partie du paramètre d'option MQSUB utilisé par défaut dans l'exemple. Le paramètre par défaut utilisé par l'exemple est MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Remarque : Définir MQSO_ALTER ou MQSO_RESUME sans définir MQSO_DURABLE est une erreur, et sd.SubName doit être défini et faire référence à un abonnement qui peut être repris ou modifié.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Si la file d'attente d'abonnement par défaut, STOCKTICKER , est remplacée par une chaîne nulle, tant que MQSO_CREATE est défini, l'exemple définit l'indicateur MQSO_MANAGED et crée une file d'attente d'abonnement dynamique. Si Alter or Resume est défini dans le cinquième paramètre, le comportement de l'exemple dépendra de la valeur de subscriptionName.

***subscriptionName = '\0';**
sdOptions = sdOptions - MQSO_DURABLE;

Si l'abonnement par défaut, IBMSTOCKPRICESUB, est remplacé par une chaîne nulle, l'exemple supprime l'indicateur MQSO_DURABLE . Si vous exécutez l'exemple en fournissant les valeurs par défaut pour les autres paramètres, un abonnement temporaire supplémentaire destiné à STOCKTICKER est créé et reçoit des publications en double. La prochaine fois que vous exécuterez l'exemple, sans aucun paramètre, vous ne recevrez qu'une seule publication à nouveau.

```
    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;
}
```

Figure 76. Abonné MQ non géré-partie 3: corps du code.

Les commentaires supplémentaires sur le code dans cet exemple sont les suivants:

if (strlen(subscriptionQueue))
S'il n'existe pas de nom de file d'attente d'abonnement, l'exemple utilise MQHO_NONE comme valeur de Hobj.

MQOPEN(...);
La file d'attente d'abonnement est ouverte et l'identificateur de file d'attente est sauvegardé dans Hobj.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
L'abonnement est ouvert à l'aide du Hobj transmis à partir de MQOPEN (ou de MQHO_NONE s'il n'existe pas de nom de file d'attente d'abonnement). Une file d'attente non gérée peut être reprise sans l'ouvrir explicitement avec un MQOPEN.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
L'abonnement est fermé à l'aide du descripteur d'abonnement. Selon que l'abonnement est durable ou non, il est fermé avec un MQCO_KEEP_SUB ou un MQCO_REMOVE_SUBimplicite. Vous pouvez fermer un abonnement durable avec MQCO_REMOVE_SUB, mais vous ne pouvez pas fermer un abonnement non durable avec MQCO_KEEP_SUB. L'action de MQCO_REMOVE_SUB consiste à supprimer l'abonnement qui arrête toute autre publication envoyée à la file d'attente d'abonnement.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
Aucune action spéciale n'est effectuée si l'abonnement n'est pas géré. Si la file d'attente est gérée et que l'abonnement est fermé avec un MQCO_REMOVE_SUBexplicite ou implicite, toutes les publications sont purgées de la file d'attente et la file d'attente est supprimée à ce stade.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Assurez-vous que les messages reçus sont ceux de notre abonnement.

Les résultats de l'exemple illustrent les aspects de la publication / l'abonnement:

Dans Figure 77, à la page 854 , l'exemple commence par la publication de 130 sur la rubrique 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
```

Figure 77. Publier 130 dans NYSE/IBM/PRICE

Dans l'exécution Figure 78, à la page 854 de l'exemple utilisant les paramètres par défaut, la publication conservée 130est reçue. L'objet de rubrique et la chaîne de rubrique fournis sont ignorés, comme indiqué dans la Figure 82, à la page 856. L'objet de rubrique et la chaîne de rubrique sont toujours extraits de l'objet d'abonnement, lorsqu'ils sont fournis, et la chaîne de rubrique est non modifiable. Le comportement réel de l'exemple dépend du choix ou de la combinaison de MQSO_CREATE, MQSO_RESUMEet MQSO_ALTER. Dans cet exemple, MQSO_RESUME est l'option sélectionnée.

```
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
```

Figure 78. Recevoir la publication conservée

Dans (Figure 79, à la page 855) aucune publication n'est reçue, car l'abonnement durable a déjà reçu la publication conservée. Dans cet exemple, l'abonnement reprend en fournissant uniquement le nom de l'abonnement sans le nom de la file d'attente. Si le nom de la file d'attente est indiqué, la file d'attente est ouverte en premier et le descripteur est transmis à MQSUB.

Remarque : L'erreur 2038 de MQINQ est due au MQOPEN implicite de STOCKTICKER par MQSUB sans inclure l'option MQOO_INQUIRE . Evitez le code retour 2038 de MQINQ en ouvrant la file d'attente de manière explicite.

```
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
```

Figure 79. Reprendre l'abonnement

Dans Figure 80, à la page 855, l'exemple crée un abonnement non durable non géré en utilisant comme destination la destination STOCKTICKER. Etant donné qu'il s'agit d'un nouvel abonnement, il reçoit la publication conservée.

```
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
```

Figure 80. Recevoir la publication conservée avec un nouvel abonnement non durable non géré

Dans Figure 81, à la page 855, pour illustrer le chevauchement des abonnements, une autre publication est envoyée, modifiant la publication conservée. Ensuite, un nouvel abonnement non durable et non géré est créé en ne fournissant pas de nom d'abonnement. La publication conservée est reçue deux fois, une fois pour le nouvel abonnement et une fois pour l'abonnement IBMSTOCKPRICESUB durable qui est toujours actif dans la file d'attente STOCKTICKER . L'exemple illustre que c'est la file d'attente qui a des abonnements et non l'application. Bien qu'elle ne se réfère pas à l'abonnement IBMSTOCKPRICESUB dans cet appel de l'application, l'application reçoit la publication deux fois: une fois à partir de l'abonnement durable qui a été créé de manière administrative et une fois à partir de l'abonnement non durable créé par l'application elle-même.

```
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
```

Figure 81. Chevauchement d'abonnements

Dans Figure 82, à la page 856 , l'exemple montre que la fourniture d'une nouvelle chaîne de rubrique et d'un abonnement existant n'entraîne pas de modification de l'abonnement.

1. Dans le premier cas, Resume reprend l'abonnement existant, comme prévu, et ignore la chaîne de rubrique modifiée.
2. Dans le second cas, Alter génère une erreur, `RC = 2510, Topic not alterable`.
3. Dans le troisième exemple, Create génère une erreur `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
```

Figure 82. Les rubriques d'abonnement ne peuvent pas être modifiées

Concepts associés

«Exemple 1: consommateur de publication MQ», à la page 838
Le consommateur de publication MQ est un consommateur de message IBM MQ qui ne s'abonne pas aux rubriques lui-même.

«Exemple 2: abonné MQ géré», à la page 841
L'abonné MQ géré est le modèle préféré pour la plupart des applications d'abonné. Un abonnement géré est un abonnement dans lequel IBM MQ gère l'abonnement et effectue l'enregistrement et le désenregistrement pour vous. L'exemple requiert *aucune* définition d'administration des files d'attente, des rubriques ou des abonnements.

«Ecriture d'applications de publication», à la page 830
Commencez par écrire des applications d'éditeur en étudiant deux exemples. La première est modélisée de la manière la plus proche possible sur une application point à point en plaçant des messages dans une file d'attente, et la seconde illustre la création dynamique de rubriques-un modèle plus courant pour les applications de diffuseur de publications.

Cycles de vie de publication / abonnement

Tenez compte des cycles de vie des rubriques, des abonnements, des abonnés, des publications, des diffuseurs et des files d'attente dans la conception des applications de publication / abonnement.

Le cycle de vie d'un objet, tel qu'un abonnement, commence par sa création et se termine par sa suppression. Il peut également inclure d'autres états et changements qu'il subit, tels que la mise en suspens temporaire, la présence de rubriques parent et enfant, l'expiration et la suppression.

Traditionnellement, les objets IBM MQ tels que les files d'attente sont créés par voie administrative ou par des programmes d'administration utilisant le format PCF (Programmable Command Format). La fonction de publication / abonnement est différente lorsqu'elle fournit les instructions d'API MQSUB et MQCLOSE permettant de créer et de supprimer des abonnements. Le concept des abonnements gérés permet non seulement de créer et de supprimer des files d'attente, mais également de nettoyer les messages non consommés, et d'avoir des associations entre les objets de rubrique créés de manière administrative et les chaînes de rubrique créées de manière programmatique ou administrative.

Cette richesse fonctionnelle répond à un large éventail d'exigences de publication / abonnement et simplifie également la conception de certains modèles communs d'application de publication / abonnement. Les abonnements gérés, par exemple, simplifient à la fois la programmation et l'administration d'un abonnement destiné à durer uniquement tant que le programme qui l'a créé. Les abonnements non gérés simplifient la programmation lorsqu'il existe une connexion plus souple entre l'abonnement et la consommation de publications. Les abonnements créés de manière centralisée sont utiles lorsque le modèle est celui de l'acheminement du trafic de publication vers les consommateurs sur la base d'un modèle de contrôle centralisé, par exemple l'envoi d'informations de vol à des portes automatisées, alors que des abonnements créés de manière programmatique peuvent être utilisés si le personnel de la porte est responsable de l'abonnement aux enregistrements de passagers pour ce vol, en entrant un numéro de vol à une porte.

Dans ce dernier exemple, un abonnement durable géré peut être approprié: géré, car les abonnements sont créés très souvent et ont un noeud final clair lors de la fermeture de la porte et l'abonnement peut être supprimé à l'aide d'un programme ; durable, pour éviter de perdre un enregistrement de passager en raison de la panne du programme d'abonné de la porte pour une raison ou une autre[8]. Pour lancer la publication des dossiers des passagers à la porte d'embarquement, une conception possible serait que

l'application de la porte d'embarquement souscrirait aux dossiers des passagers à l'aide du numéro de la porte d'embarquement et publierait l'événement d'ouverture de la porte d'embarquement à l'aide du numéro de la porte d'embarquement. L'éditeur répond à l'événement d'ouverture de la porte en publiant les dossiers des passagers-qui pourraient alors également être rendus à d'autres parties intéressées, telles que la facturation, pour enregistrer le vol en cours, et aux services à la clientèle, pour envoyer des notifications par SMS aux téléphones mobiles des passagers du numéro de la porte.

L'abonnement géré de manière centralisée peut utiliser un modèle non géré durable, qui achemine les listes de passagers vers le portail à l'aide d'une file d'attente prédéfinie pour chaque portail.

Les trois exemples de cycles de vie de publication / abonnement suivants illustrent la façon dont les abonnés non durables gérés, durables gérés et durables non gérés interagissent avec les abonnements, les rubriques, les files d'attente, les diffuseurs de publications et le gestionnaire de files d'attente, ainsi que la façon dont les responsabilités peuvent être réparties entre l'administration et les programmes d'abonné.

Abonné non durable géré

La Figure 83, à la page 858 montre une application qui crée un abonnement non durable géré, qui obtient deux messages publiés dans la rubrique identifiée dans l'abonnement et qui s'arrête. Les interactions libellées dans une police grise en italique avec des flèches en pointillés sont implicites.

Il y a quelques points à noter.

1. L'application crée un abonnement sur une rubrique qui a déjà été publiée deux fois. Lorsque l'abonné reçoit sa première publication, il reçoit la *deuxième* publication qui est la publication actuellement conservée.
2. Le gestionnaire de files d'attente crée une file d'attente d'abonnement temporaire ainsi qu'un abonnement pour la rubrique.
3. L'abonnement a une expiration. Lorsque l'abonnement arrive à expiration, aucune autre publication de la rubrique n'est envoyée à cet abonnement, mais l'abonné continue d'obtenir des messages publiés avant l'expiration de l'abonnement. L'expiration de la publication n'est pas affectée par l'expiration de l'abonnement.
4. La quatrième publication n'est pas placée dans la file d'attente d'abonnement et par conséquent, le dernier `MQGET` ne renvoie pas de publication.
5. Bien que l'abonné ferme son abonnement, il ne ferme pas sa connexion à la file d'attente ou au gestionnaire de files d'attente.
6. Le gestionnaire de files d'attente est nettoyé peu après l'arrêt de l'application. L'abonnement étant géré et non durable, la file d'attente d'abonnement est supprimée.

[8] L'éditeur doit envoyer les dossiers passagers en tant que messages persistants pour éviter d'autres défaillances possibles, bien sûr.

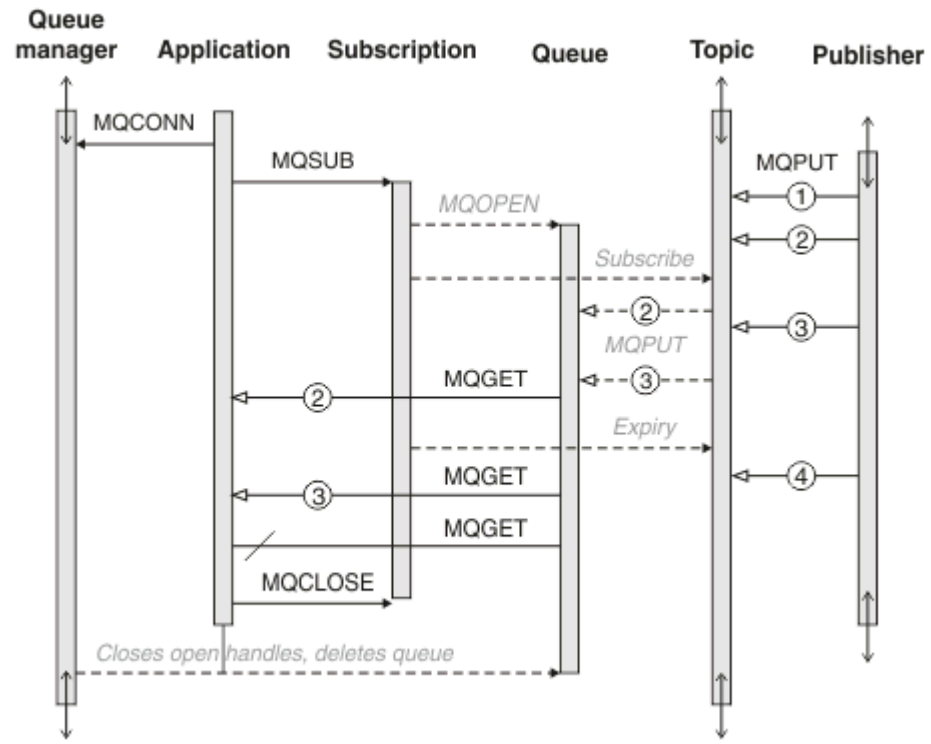


Figure 83. Lignes de vie d'abonné non durables gérées

Abonné durable géré

L'abonné durable géré reprend l'exemple précédent et affiche un abonnement géré survivant à l'arrêt et au redémarrage de l'application d'abonnement.

Il y a de nouveaux points à noter.

1. Dans cet exemple, contrairement à la dernière, la rubrique de publication n'existait pas avant d'être définie dans l'abonnement.
2. La première fois que l'abonné s'arrête, il ferme l'abonnement avec l'option MQCO_KEEP_SUB. Il s'agit du comportement par défaut pour la fermeture implicite d'un abonnement durable géré.
3. Lorsque l'abonné reprend l'abonnement, la file d'attente d'abonnement est rouverte.
4. La nouvelle publication 2, placée dans la file d'attente avant sa réouverture, est disponible pour MQGET, même après la suppression de l'abonnement.

   Même si l'abonnement est durable, l'abonné reçoit de manière fiable tous les messages envoyés par le diffuseur de publications uniquement si l'abonnement est *à la fois* durable et persistant. La persistance des messages dépend de la valeur de la zone Persistent dans le MQMD du message envoyé par le diffuseur de publications. Un abonné n'a aucun contrôle à ce sujet.
5. La fermeture de l'abonnement avec l'indicateur MQCO_REMOVE_SUB supprime l'abonnement, en arrêtant toute autre publication placée dans la file d'attente d'abonnement. Lorsque la file d'attente d'abonnement est fermée, le gestionnaire de files d'attente supprime la publication non lue 3, puis supprime la file d'attente. L'action équivaut à la suppression administrative de l'abonnement.

   Remarque : Ne supprimez pas la file d'attente manuellement ou émettez la commande MQCLOSE avec l'option MQCO_DELETEou MQCO_PURGE_DELETE. Les détails d'implémentation visibles d'un abonnement géré ne font pas partie de l'interface IBM MQ prise en charge. Le gestionnaire de files d'attente ne peut pas gérer un abonnement de manière fiable à moins qu'il ne dispose d'un contrôle complet.

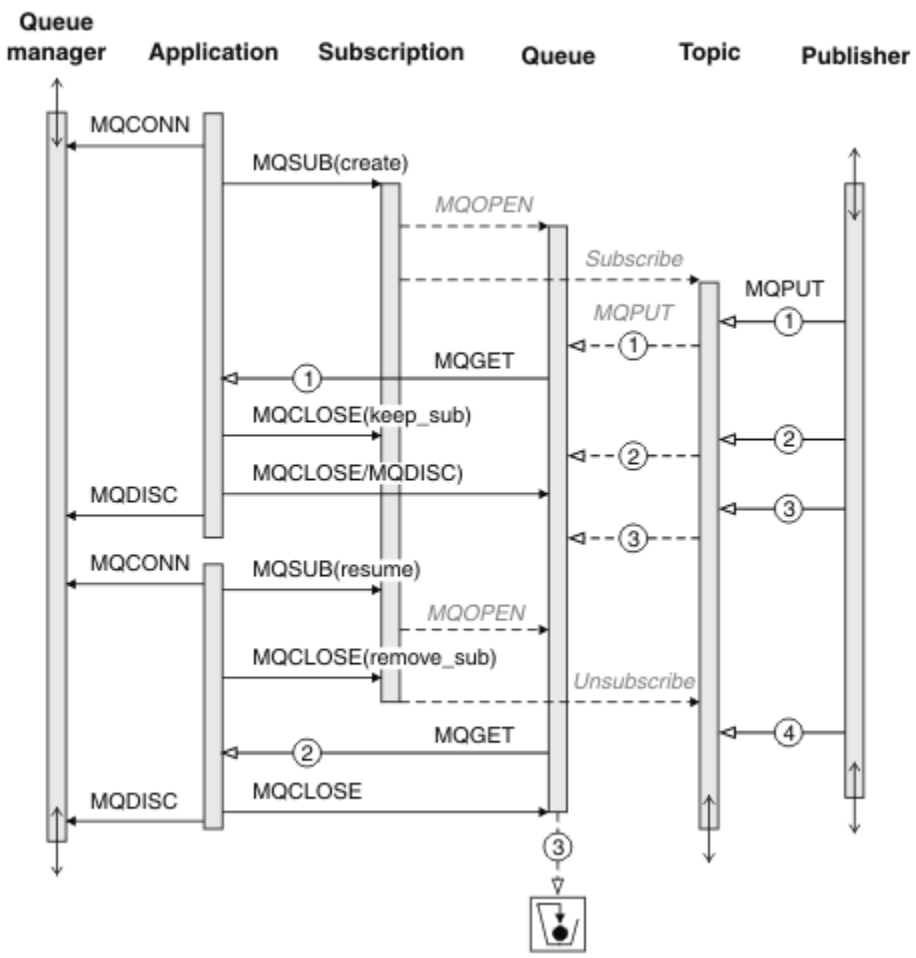


Figure 84. Lignes de vie d'abonné durable gérées

Abonné durable non géré

Un administrateur est ajouté dans le troisième exemple: l'abonné durable non géré. Il s'agit d'un bon exemple qui montre comment l'administrateur peut interagir avec une application de publication / abonnement.

Les points à noter sont répertoriés.

1. Le diffuseur de publications insère un message, 1, dans une rubrique qui est associée ultérieurement à l'objet de rubrique utilisé pour l'abonnement. L'objet de rubrique définit une chaîne de rubrique qui correspond à la rubrique qui a été publiée à l'aide de caractères génériques.
2. La rubrique comporte une publication conservée.
3. L'administrateur crée un objet de rubrique, une file d'attente et un abonnement. L'objet de rubrique et la file d'attente doivent être définis avant l'abonnement.
4. L'application ouvre la file d'attente associée à l'abonnement et transmet MQSUB le descripteur de la file d'attente. Il peut également simplement ouvrir l'abonnement, en lui transmettant le descripteur de file d'attente MQHO_NONE. L'inverse n'est pas vrai, il ne peut pas reprendre un abonnement en lui transmettant uniquement un descripteur de file d'attente sans nom d'abonnement-une file d'attente peut avoir plusieurs abonnements.
5. L'application ouvre l'abonnement à l'aide de l'option MQSO_RESUME même si c'est la première fois qu'elle ouvre l'abonnement. Il reprend un abonnement créé administrativement.

6. L'abonné reçoit la publication conservée, 1. La publication 2, bien que publiée avant toute publication reçue par l'abonné, a été publiée après le démarrage de l'abonnement et est la deuxième publication dans la file d'attente d'abonnement.

   Remarque : Si la publication conservée n'est pas publiée en tant que message persistant, elle est perdue après le redémarrage du gestionnaire de files d'attente.
7. Dans cet exemple, l'abonnement est durable. Il est possible pour un programme de créer un abonnement non durable non géré ; il doit être évident que ce n'est pas une tâche qu'un administrateur peut effectuer.
8. L'effet de l'option MQCO_REMOVE_SUB sur la fermeture de l'abonnement est de supprimer l'abonnement comme si l'administrateur l'avait supprimé. Cela arrête toute autre publication envoyée à la file d'attente, mais n'affecte pas les publications qui se trouvent déjà dans la file d'attente, même lorsque la file d'attente est fermée, contrairement à un abonnement durable *géré* .
9. L'administrateur supprime ensuite le message restant, 3, et supprime la file d'attente.

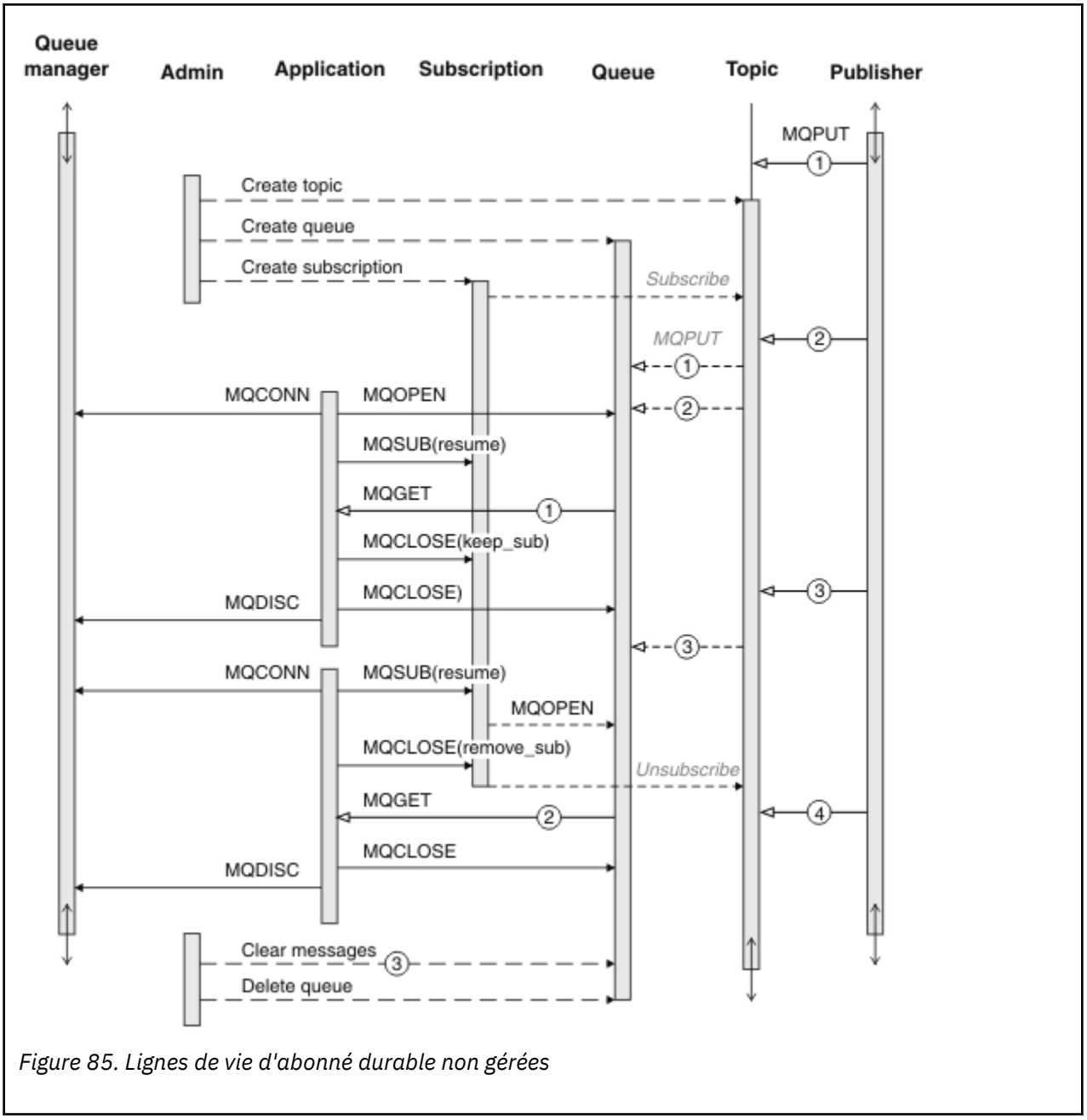


Figure 85. Lignes de vie d'abonné durable non gérées

Un modèle normal pour un abonnement non géré consiste à effectuer des tâches de nettoyage de file d'attente et d'abonnement par l'administrateur. En règle générale, on ne tente pas d'émuler le comportement d'un abonné géré et d'arranger les files d'attente et les abonnements à l'aide d'un programme dans le code d'application. Si vous avez besoin d'écrire une logique de gestion, vous devez vous demander si vous pouvez obtenir les mêmes résultats à l'aide d'un modèle géré. Il n'est pas facile d'écrire un code de gestion parfaitement synchronisé et fiable. Il est plus facile de nettoyer plus tard, soit manuellement, soit à l'aide d'un programme de gestion automatisé, lorsque vous pouvez être certain que les messages, les abonnements et les files d'attente peuvent être simplement supprimés, quel que soit leur état.

Propriétés des messages de publication / abonnement

Plusieurs propriétés de message se rapportent à la messagerie de publication / abonnement IBM MQ .

Jeton PubAccounting

Il s'agit de la valeur qui sera indiquée dans la zone AccountingToken du descripteur de message (MQMD) de tous les messages de publication correspondant à cet abonnement. AccountingToken fait partie du contexte d'identité du message. Pour plus d'informations sur le contexte de message, voir «Contexte de message», à la page 49. Pour plus d'informations sur la zone AccountingToken dans le MQMD, voir AccountingToken.

PubApplIdentityData

Il s'agit de la valeur qui sera dans la zone de données ApplIdentitydu descripteur de message (MQMD) de tous les messages de publication correspondant à cet abonnement. ApplIdentityLes données font partie du contexte d'identité du message. Pour plus d'informations sur le contexte de message, voir «Contexte de message», à la page 49. Pour plus d'informations sur la zone de données ApplIdentitydans le MQMD, voir DonnéesApplIdentity.

Si l'option MQSO_SET_IDENTITY_CONTEXT n'est pas spécifiée, les données ApplIdentityqui seront définies dans chaque message publié pour cet abonnement sont vides, comme informations de contexte par défaut.

Si l'option MQSO_SET_IDENTITY_CONTEXT est spécifiée, PubApplIdentityData est généré par l'utilisateur et cette zone est une zone d'entrée qui contient les données ApplIdentityà définir dans chaque publication pour cet abonnement.

PubPriority

Il s'agit de la valeur qui sera dans la zone Priorité du descripteur de message (MQMD) de tous les messages de publication correspondant à cet abonnement. Pour plus d'informations sur la zone Priorité dans le MQMD, voir Priorité.

La valeur doit être supérieure ou égale à zéro ; zéro est la priorité la plus basse. Les valeurs spéciales suivantes peuvent également être utilisées:

- MQPRI_PRIORITY_AS_Q_DEF-Lorsqu'une file d'attente d'abonnement est fournie dans la zone Hobj de l'appel MQSUB et qu'elle n'est pas un descripteur géré, la priorité du message est extraite de l'attribut DefProirity de cette file d'attente. Si la file d'attente ainsi identifiée est une file d'attente de cluster ou qu'il existe plusieurs définitions dans le chemin de résolution de nom de file d'attente, la priorité est déterminée lorsque le message de publication est inséré dans la file d'attente, comme décrit pour Priorité dans le MQMD. Si l'appel MQSUB utilise un descripteur géré, la priorité du message est extraite de l'attribut DefPriority de la file d'attente modèle associée à la rubrique à laquelle le message est abonné.
- MQPRI_PRIORITY_AS_PUBLISHED-La priorité du message est la priorité de la publication d'origine. Il s'agit de la valeur initiale de cette zone.

SubCorrelId

Avertissement : un identificateur de corrélation ne peut être transmis qu'entre des gestionnaires de files d'attente dans un cluster de publication / abonnement et non une hiérarchie.

Toutes les publications envoyées pour correspondre à cet abonnement contiendront cet identificateur de corrélation dans le descripteur de message. Si plusieurs abonnements utilisent la même file d'attente pour obtenir leurs publications, l'utilisation de MQGET par ID de corrélation permet d'obtenir uniquement les publications d'un abonnement spécifique. Cet identificateur de corrélation peut être généré par le gestionnaire de files d'attente ou par l'utilisateur.

Si l'option MQSO_SET_CORREL_ID n'est pas spécifiée, l'identificateur de corrélation est généré par le gestionnaire de files d'attente et cette zone est une zone de sortie qui contient l'identificateur de corrélation qui sera défini dans chaque message publié pour cet abonnement.

Si l'option MQSO_SET_CORREL_ID est spécifiée, l'identificateur de corrélation est généré par l'utilisateur et cette zone est une zone d'entrée qui contient l'identificateur de corrélation à définir dans chaque publication pour cet abonnement. Dans ce cas, si la zone contient MQCI_NONE, l'identificateur de corrélation qui sera défini dans chaque message publié pour cet abonnement sera l'identificateur de corrélation créé par l'insertion d'origine du message.

Si l'option MQSO_GROUP_SUB est spécifiée et que l'identificateur de corrélation spécifié est identique à un abonnement groupé existant utilisant la même file d'attente et une chaîne de rubrique se chevauchant, seul l'abonnement le plus significatif du groupe est fourni avec une copie de la publication.

SubUserData

Il s'agit des données utilisateur de l'abonnement. Les données fournies sur l'abonnement dans cette zone seront incluses en tant que propriété de message de données MQSubUserde chaque publication envoyée à cet abonnement.

Propriétés de publication

Le Tableau 124, à la page 862 répertorie les propriétés de publication fournies avec un message de publication.

Vous pouvez accéder à ces propriétés directement à partir du dossier **MQRFH2** ou les extraire à l'aide de MQINQMP. MQINQMP accepte le nom de la propriété ou le nom **MQRFH2** comme nom de la propriété à interroger.

Tableau 124. Propriétés de publication

Nom de la propriété	Nom MQRFH2	Tapez	Description
MQTopicString	mqps.Top	MQTYPE_STRING	Chaîne de rubrique
MQSubUserData	mqps.Sud	MQTYPE_STRING	Données utilisateur de l'abonné
MQIsRetained	mqps.Ret	MQTYPE_BOOLEAN	Publication conservée
MQPubOptions	mqps.Pub	MQTYPE_INT32	Options de publication
MQPubLevel	mqps.Pbl	MQTYPE_INT32	Niveau de publication
MQPubTime	mqpse.Pts	MQTYPE_STRING	Heure de publication
MQPubSeqNum	mqpse.Seq	MQTYPE_INT32	Numéro de séquence de la publication
MQPubStrIntData	mqpse.Sid	MQTYPE_STRING	Données de chaîne / entier ajoutées par le diffuseur de publications

Tableau 124. Propriétés de publication (suite)			
Nom de la propriété	**Nom MQRFH2**	**Tapez**	**Description**
`MQPubFormat`	`mqpse.Pfmt`	`MQTYPE_INT32`	Format du message: MQRFH1 MQRFH2 PCF

Ordre des messages

Pour une rubrique particulière, les messages sont publiés par le gestionnaire de files d'attente dans l'ordre dans lequel ils sont reçus des applications de publication (sujet à une réorganisation en fonction de la priorité des messages).

L'ordre des messages signifie normalement que chaque abonné reçoit des messages d'un gestionnaire de files d'attente particulier, sur une rubrique particulière, d'un diffuseur de publications particulier dans l'ordre dans lequel ils sont publiés par ce diffuseur de publications.

Toutefois, comme pour tous les messages IBM MQ , il est possible que les messages soient parfois distribués dans le désordre. Cela peut se produire dans les situations suivantes:

- Si un lien du réseau tombe en panne et que les messages suivants sont réacheminés via un autre lien
- Si une file d'attente est temporairement saturée, ou bloquée, de sorte qu'un message est inséré dans une file d'attente de rebut et donc retardé, alors que les messages suivants passent directement.
- Si l'administrateur supprime un gestionnaire de files d'attente alors que les diffuseurs de publications et les abonnés fonctionnent toujours, les messages en file d'attente sont placés dans la file d'attente des messages non livrés et les abonnements sont interrompus.

Si ces circonstances ne peuvent pas se produire, les publications sont toujours distribuées dans l'ordre.

Remarque : Il n'est pas possible d'utiliser des messages groupés ou segmentés avec la fonction de publication / abonnement.

Interception des publications

Vous pouvez intercepter une publication, la modifier, puis la republier avant qu'elle n'atteigne un autre abonné.

Vous pouvez être amené à intercepter une publication avant qu'elle n'atteigne un abonné afin d'effectuer l'une des actions suivantes:

- Joindre des informations supplémentaires au message
- Bloquer le message
- Transformer le message

Vous pouvez effectuer la même opération sur chaque message ou modifier l'opération en fonction de l'abonnement, du message ou de l'en-tête de message.

Référence associée

MQ_PUBLISH_EXIT-Exit de publication

Niveaux d'abonnement

Définissez le niveau d'abonnement d'un abonnement pour intercepter une publication avant qu'elle n'atteigne ses abonnés finaux. Un abonné interceptant s'abonne à un niveau d'abonnement supérieur et republie à un niveau de publication inférieur. Générez une chaîne d'interception des abonnés pour effectuer le traitement des messages sur une publication avant qu'elle ne soit distribuée aux abonnés finaux.

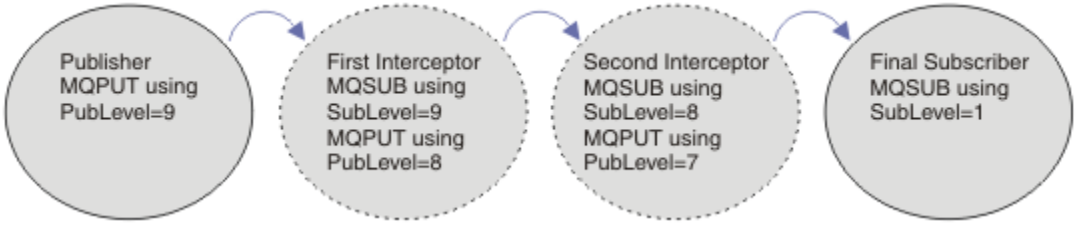


Figure 86. Séquence d'interception des abonnés

Pour intercepter une publication, utilisez l'attribut **MQSD** SubLevel . Une fois qu'un message a été intercepté, il peut être transformé puis republié à un niveau de publication inférieur en modifiant l'attribut **MQPMO** PubLevel . Le message est ensuite envoyé aux abonnés finaux, ou il est à nouveau intercepté par un abonné intermédiaire à un niveau d'abonnement inférieur.

L'abonné intercepteur transforme généralement un message avant de le republier. Une séquence d'interception d'abonnés forme un flux de messages. Vous pouvez également ne pas republier la publication interceptée: les abonnés à des niveaux d'abonnement inférieurs ne reçoivent pas le message.

Assurez-vous que l'intercepteur reçoit les publications avant les autres abonnés. Définissez le niveau d'abonnement de l'intercepteur sur une valeur supérieure à celle des autres abonnés. Par défaut, les abonnés ont un SubLevel de 1. La valeur la plus élevée est 9. Une publication doit commencer par un niveau PubLevel au moins aussi élevé que le niveau le plus élevé SubLevel. Publication initiale avec la valeur par défaut PubLevel de 9.

- Si vous disposez d'un abonné intercepteur sur une rubrique, définissez SubLevel sur 9.
- Pour plusieurs applications d'interception sur une rubrique, définissez un SubLevel inférieur pour chaque abonné d'interception successif.
- Vous pouvez implémenter un maximum d'applications d'interception 8 , avec des niveaux d'abonnement allant de 9 à 2 inclus. Le destinataire final du message a un SubLevel de 1.

L'intercepteur dont le niveau d'abonnement le plus élevé est égal ou inférieur à PubLevel de la publication reçoit la publication en premier. Configurez un seul abonné d'interception pour une rubrique à un niveau d'abonnement particulier. Le fait d'avoir plusieurs abonnés à un niveau d'abonnement particulier entraîne l'envoi de plusieurs copies de la publication à l'ensemble final d'applications d'abonnement.

Un abonné avec un SubLevel de 0 est utilisé comme fourre-tout. Il reçoit la publication si aucun abonné final n'obtient le message. Un abonné avec SubLevel de 0 peut être utilisé pour surveiller les publications qu'aucun autre abonné n'a reçues.

Programmation d'un abonné intercepteur

Utilisez les options d'abonnement décrites dans Tableau 125, à la page 864.

Tableau 125. Options d'abonnement pour l'interception des abonnés

Option d'abonnement	Remarques
MQSO_SET_CORREL_ID et SubCorrelId défini sur MQCI_NONE	Conservez l'ID CorrelId de la publication interceptée identique à la publication d'origine. **Remarque :** Vous ne pouvez pas transmettre l'identificateur de corrélation d'une publication dans une hiérarchie. Cette zone est utilisée par le gestionnaire de files d'attente.
PubPriority défini sur MQPRI_PRIORITY_AS_PUBLISHED	Conservez la priorité de la publication interceptée identique à celle de la publication d'origine.

Les options dans Tableau 125, à la page 864 doivent être utilisées par tous les abonnés à l'interception. Il en résulte que l'identificateur de corrélation et la priorité de message ne sont pas modifiés à partir du paramètre du diffuseur de publications d'origine.

Lorsque l'abonné intercepteur a traité la publication, il republie le message dans la même rubrique à un niveau `PubLevel` inférieur à `SubLevel` de son propre abonnement. Si l'abonné intercepteur a défini un `SubLevel` de 9, il republie le message avec un `PubLevel` de 8.

Pour republier correctement le message, plusieurs éléments d'information de la publication d'origine sont requis. Réutilisez le même **MQMD** que dans le message d'origine et définissez `MQPMO_PASS_ALL_CONTEXT` pour vous assurer que toutes les informations du **MQMD** sont transmises à l'abonné suivant. Copiez les valeurs des propriétés de message indiquées dans le Tableau 126, à la page 865 dans les zones correspondantes du message republié. L'abonné intercepteur peut modifier ces valeurs. Utilisez l'opérateur OR pour ajouter des valeurs supplémentaires à **MQPMO**. Zone `Options` , pour combiner les options d'insertion de message.

Vous devez ouvrir la file d'attente de publication de manière explicite au lieu d'utiliser une file d'attente de publication gérée. Vous ne pouvez pas définir `MQSO_SET_CORREL_ID` pour une file d'attente gérée. Vous ne pouvez pas non plus définir `MQOO_SAVE_ALL_CONTEXT` sur une file d'attente gérée. Voir les fragments de code répertoriés dans le «Exemples», à la page 865.

Tableau 126. Valeurs MQPUT pour les messages republiés

Republier le message à l'aide de MQPUT	**Informations dans le message de publication**
MQOD. `ObjectString`	Propriété de message `MQTopicString`
MQPMO. `Options`	Propriété de message `MQPubOptions`

L'abonné final a le choix de définir ses options d'abonnement différemment. Par exemple, il peut définir la priorité de publication explicitement plutôt que sur `MQPRI_PRIORITY_AS_PUBLISHED`. Les paramètres d'un abonné final affectent uniquement la publication de l'abonné d'interception final dans la chaîne.

Publications conservées

Une publication conservée doit être conservée une fois qu'elle a été interceptée, en copiant les options d'insertion de message d'origine dans le message republié.

L'option `MQPMO_RETAIN` est définie par le diffuseur de publications. Chaque abonné intercepteur doit transférer le `MQPubOptions` vers les options d'insertion de message du message republié, comme illustré dans la Tableau 126, à la page 865. La copie des options d'insertion de message conserve les options définies par le diffuseur de publications d'origine, y compris la possibilité de conserver la publication.

Lorsqu'une publication termine son passage le long de la chaîne d'interception des abonnés, et qu'elle est distribuée aux abonnés finaux, elle est finalement conservée. Les nouveaux abonnés, à l'adresse `SubLevel` 1, qui demandent la publication conservée, la reçoivent sans autre interception. Les abonnés dont le niveau `SubLevel` est supérieur à 1 ne reçoivent pas la publication conservée. Par conséquent, la publication conservée n'est pas modifiée par la chaîne d'interception des abonnés une deuxième fois.

Exemples

Les exemples sont des fragments de code qui peuvent être combinés pour générer un abonné d'interception. Le code est écrit pour être bref, plutôt que de la qualité de la production.

Les directives de préprocesseur dans Figure 87, à la page 866 définissent les deux propriétés à extraire des messages de publication requis par l'appel `MQINQMP` MQI.

```
#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
```

Figure 87. Directives de préprocesseur

Le Figure 88, à la page 866 répertorie les déclarations utilisées dans les fragments de code. A l'exception des termes mis en évidence, les déclarations sont standard pour une application IBM MQ .

Les options Put et Get mises en évidence sont initialisées pour transmettre tout le contexte. Les MQTOPICSTRING et MQPUBOPTIONS mis en évidence sont des initialiseurs MQCHARV pour les noms de propriété définis dans les directives de préprocesseur. Les noms sont transmis à 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;
```

Figure 88. Déclarations

Les initialisations qui ne sont pas faciles à effectuer dans des déclarations sont présentées dans Figure 89, à la page 867. Les valeurs mises en évidence nécessitent une explication.

SYSTEM.NDURABLE.MODEL.QUEUE
: Dans cet exemple, au lieu d'utiliser MQSUB pour ouvrir un abonnement non durable géré, la file d'attente modèle, SYSTEM.NDURABLE.MODEL.QUEUE, est utilisée pour créer une file d'attente dynamique temporaire. Son descripteur est transmis à MQSUB. En ouvrant directement la file

d'attente, vous pouvez sauvegarder tous les contextes de message et définir l'option d'abonnement, MQSO_SET_CORREL_ID.

MQGMO_CURRENT_VERSION
Il est important d'utiliser la version actuelle de la plupart des structures IBM MQ . Les zones telles que gmo.MsgHandle ne sont disponibles que dans la dernière version des structures de contrôle.

MQGMO_PROPERTIES_IN_HANDLE
La chaîne de rubrique et les options de message d'insertion définies dans la publication d'origine doivent être extraites par l'abonné intercepteur à l'aide des propriétés de message. Vous pouvez également lire directement la structure **MQRFH2** dans le message.

MQSO_SET_CORREL_ID
Utilisez MQSO_SET_CORREL_ID en combinaison avec,

```
memcpy(sd.SubCorrelId, MQCI_NONE, sizeof(sd.SubCorrelId));
```

Ces options ont pour effet de transmettre l'identificateur de corrélation. L'identificateur de corrélation défini par le diffuseur de publications d'origine est placé dans la zone d'identificateur de corrélation de la publication qui est reçue par l'abonné intercepteur. Chaque abonné intercepteur transmet le même identificateur de corrélation. L'abonné final a alors la possibilité de recevoir le même identifiant de corrélation.

Remarque : Si la publication est transmise via une hiérarchie de publication / abonnement, l'identificateur de corrélation n'est jamais conservé.

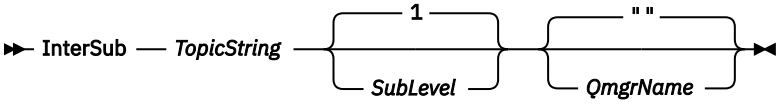
MQPRI_PRIORITY_AS_PUBLISHED
La publication est placée dans la file d'attente de publication avec la même priorité de message que celle avec laquelle elle a été publiée.

```
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;
```

Figure 89. Initialisations

Figure 90, à la page 868 montre le fragment de code permettant de lire les paramètres de ligne de commande, de terminer l'initialisation et de créer l'abonnement d'interception.

Exécutez le programme à l'aide de la commande suivante:



Pour rendre le traitement des erreurs aussi discret que possible, le code anomalie de chaque appel MQI est stocké dans un élément de tableau différent. Après chaque appel, le code achèvement est testé et si la valeur est MQCC_FAIL, le contrôle quitte le bloc de code `do { } while(0)` .

Les deux lignes de code remarquables sont:

pmo.PubLevel = sd.SubLevel - 1;
Définit le niveau de publication du message republié à un niveau inférieur à celui de l'abonnement de l'abonné intercepteur.

gmo.MsgHandle = Hmsg;
Fournit un descripteur de message permettant à MQGET de renvoyer les propriétés de message.

```
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;
```

Figure 90. Préparation de l'interception des publications

Le fragment de code principal, Figure 91, à la page 869, extrait les messages de la file d'attente de publication. Il interroge les propriétés de message et publie à nouveau les messages à l'aide de la chaîne de rubrique et du **MQPMO**d'origine. Propriétés `option` de la publication.

Dans cet exemple, aucune transformation n'est effectuée sur la publication. La chaîne de rubrique de la publication republiée correspond toujours à la chaîne de rubrique sur laquelle l'abonné intercepteur s'est abonné. Si l'abonné intercepteur est responsable de l'interception de plusieurs abonnements envoyés à la même file d'attente de publication, il peut être nécessaire d'interroger la chaîne de rubrique pour distinguer les publications qui correspondent à des abonnements différents.

Les appels à MQINQMP sont mis en évidence. La chaîne de rubrique et les propriétés des options de message d'insertion de publication sont écrites directement dans les structures de contrôle de sortie. La seule raison de la modification de la zone de longueur MQCHARV de `putOD.ObjectString` d'une longueur explicite à une chaîne terminée par une valeur nulle est d'utiliser `printf` pour générer la chaîne.

```
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;
}
```

Figure 91. Intercepter la publication et la republier

Le fragment de code final est illustré dans la Figure 92, à la page 869.

```
    } 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);
}
```

Figure 92. Completion

Interception des publications et publication / abonnement distribué
Suivez un modèle simple lorsque vous déployez des abonnés intercepteurs ou des exits de publication dans une topologie de publication / abonnement distribuée. Déployez l'interception des abonnés sur les mêmes gestionnaires de files d'attente que les diffuseurs et les exits de publication sur les mêmes gestionnaires de files d'attente que les abonnés finaux.

La Figure 93, à la page 870 montre deux gestionnaires de files d'attente connectés dans un cluster de publication / abonnement. Un diffuseur de publications crée une publication dans une rubrique de cluster au niveau de la publication 9. Les flèches numérotées indiquent la séquence des étapes effectuées par la publication lorsqu'elle est transmise aux abonnés à la rubrique de cluster. La publication est interceptée par l'abonné avec `Sublevel` 9 et republiée avec `Publevel` 8. Il est de nouveau intercepté par un abonné au `Sous-niveau` 8. L'abonné republie sur `Publevel` 7. L'abonné proxy fourni par le gestionnaire de files d'attente achemine la publication vers le gestionnaire de files d'attente B, où un exit de publication a été déployé en plus d'un abonné final. La publication est traitée par l'exit de publication avant d'être finalement reçue par l'abonné final au `Sous-niveau` 1. Les abonnés à l'interception et l'exit de publication sont affichés avec des contours rompus.

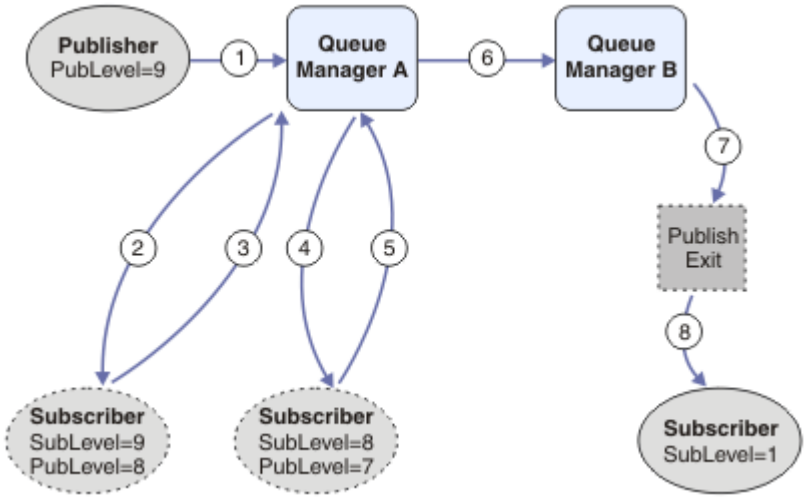


Figure 93. Exit d'interception et de publication dans un cluster

L'objectif du modèle simple est que chaque abonné recevant une publication reçoive la publication identique. La publication passe par la même séquence de transformations quel que soit l'endroit où l'abonné est connecté. Vous souhaitez probablement éviter que la séquence des transformations varie en fonction de l'endroit où les diffuseurs de publications ou les abonnés finaux sont connectés. Une exception raisonnable serait d'adapter la publication finalement livrée à chaque abonné. Utilisez l'exit de publication pour personnaliser la publication en fonction de la file d'attente dans laquelle la publication est finalement distribuée.

Vous devez déterminer avec soin où déployer les exits d'interception des abonnés et de publication dans une topologie de publication / abonnement distribuée. Le modèle simple déploie les abonnés intercepteurs sur le même gestionnaire de files d'attente que les diffuseurs et les exits de publication sur les mêmes gestionnaires de files d'attente que les abonnés finaux.

Anti-pattern

Figure 94, à la page 871 montre comment les choses peuvent se passer, si vous ne suivez pas un modèle simple. Pour compliquer le déploiement, un abonné final est ajouté au gestionnaire de files d'attente A et deux abonnés d'interception supplémentaires sont ajoutés au gestionnaire de files d'attente B.

La publication est réacheminée vers le gestionnaire de files d'attente B à l'adresse `PubLevel` 7, où elle est interceptée par un abonné à l'adresse `SubLevel` 5 avant d'être consommée par l'abonné final à l'adresse `SubLevel` 1. L'exit de publication intercepte la publication avant qu'elle ne soit transmise à la fois au consommateur d'interception et au consommateur final dans le gestionnaire de files d'attente B. La publication atteint l'abonné final sur le gestionnaire de files d'attente A sans être traitée par l'exit de publication.

Dans une topologie de publication / abonnement, les abonnés proxy s'abonnent à `SubLevel` 1et passent sur le `PubLevel` défini par le dernier abonné intercepteur. Dans Figure 94, à la page 871, le résultat est que la publication n'est pas interceptée par l'abonné à l'aide de `SubLevel` 9 au niveau du gestionnaire de files d'attente B.

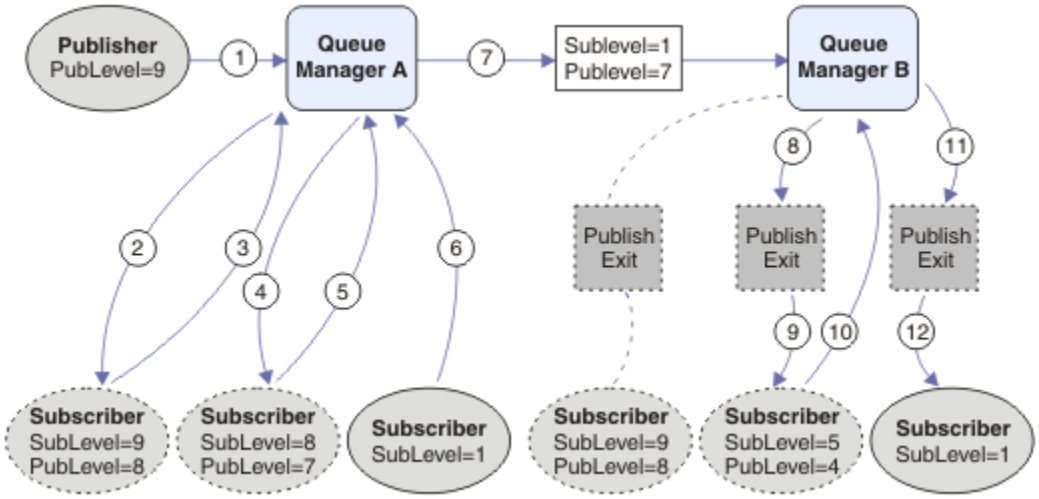


Figure 94. Déploiement complexe de l'interception des abonnés

Options de publication
Plusieurs options sont disponibles pour contrôler le mode de publication des messages.

Rétention des informations de réponse des abonnés
Si vous ne souhaitez pas que les abonnés puissent répondre aux publications qu'ils reçoivent, il est possible de retenir des informations dans les zones ReplyToQ et ReplyToQmgr de MQMD à l'aide de l'option d'insertion de message MQPMO_SUPPRESS_REPLYTO. Si cette option est utilisée, le gestionnaire de files d'attente supprime ces informations du MQMD lorsqu'il reçoit la publication avant de la transmettre à des abonnés.

Cette option ne peut pas être utilisée en combinaison avec une option de rapport qui nécessite une file d'attente ReplyTo, si l'appel échoue avec MQRC_MISSING_REPLY_TO_Q.

Niveau de publication
L'utilisation des niveaux de publication permet de contrôler les abonnés qui reçoivent la publication. Le niveau de publication indique le niveau d'abonnement ciblé par la publication. Seuls les abonnements dont le niveau d'abonnement le plus élevé est inférieur ou égal au niveau de publication de la publication recevront la publication. Cette valeur doit être comprise entre zéro et neuf ; zéro est le niveau de publication le plus bas. La valeur initiale de cette zone est 9. L'une des utilisations des niveaux de publication et d'abonnement consiste à intercepter les publications.

Vérifier si une publication n'est pas distribuée à des abonnés

Pour vérifier si une publication n'a pas été distribuée à des abonnés, utilisez l'option put-message MQPMO_WARN_IF_NO_SUBS_MET en correspondance avec l'appel MQPUT. Si un code achèvement de MQCC_WARNING et un code anomalie MQRC_NO_SUBS_MET en correspondance sont renvoyés par l'opération d'insertion, la publication n'a été distribuée à aucun abonnement. Si l'option MQPMO_RETAIN est spécifiée lors de l'opération d'insertion, le message est conservé et distribué à tout abonnement correspondant défini ultérieurement. Dans un système de publication / abonnement distribué, le code anomalie MQRC_NO_SUBS_APPARIÉ est renvoyé uniquement si aucun abonnement de proxy n'est enregistré pour la rubrique sur le gestionnaire de files d'attente.

Options d'abonnement
Plusieurs options sont disponibles pour contrôler la façon dont les abonnements aux messages sont gérés.

Persistance des messages

Les gestionnaires de files d'attente conservent la persistance des publications qu'ils transmettent aux abonnés, telle qu'elle est définie par le diffuseur de publications. Le diffuseur de publications définit la persistance comme étant l'une des options suivantes:

0
Non persistant

1
Persistante

2
Persistance en tant que définition de file d'attente/rubrique

Pour la publication / l'abonnement, le diffuseur de publications résout l'objet de rubrique et **`topicString`** un objet de rubrique résolu. Si le diffuseur de publications spécifie Persistance comme définition de file d'attente/rubrique, la persistance par défaut de l'objet de rubrique résolu est définie pour la publication.

Publications conservées

Pour contrôler la réception des publications conservées, les abonnés peuvent utiliser deux options d'abonnement:

Publication sur demande uniquement, MQSO_PUBLICATIONS_ON_REQUEST

Si vous souhaitez qu'un abonné ait le contrôle du moment où il reçoit les publications, vous pouvez utiliser l'option d'abonnement MQSO_PUBLICATIONS_ON_REQUEST. Un abonné peut ensuite contrôler à quel moment il reçoit des publications à l'aide de l'appel MQSUBRQ (en spécifiant le descripteur Hsub renvoyé par l'appel MQSUB d'origine) pour demander qu'une publication conservée d'une rubrique lui soit envoyée. Les abonnés utilisant l'option d'abonnement MQSO_PUBLICATIONS_ON_REQUEST ne reçoivent pas de publications non conservées.

Si vous spécifiez MQSO_PUBLICATIONS_ON_REQUEST, vous devez utiliser MQSUBRQ pour extraire une publication. Si vous n'utilisez pas MQSO_PUBLICATIONS_ON_REQUEST, des messages sont générés lors de leur publication.

Si un abonné utilise l'appel MQSUBRQ et des caractères génériques dans la rubrique de l'abonnement, l'abonnement peut correspondre à plusieurs rubriques ou noeuds d'une arborescence de rubriques, dont tous les messages conservés (le cas échéant) seront envoyés à l'abonné.

Cette option peut être particulièrement utile lorsqu'elle est utilisée avec des abonnements durables car un gestionnaire de files d'attente continue à envoyer des publications à un abonné s'il s'est abonné de manière durable même si cette application d'abonné n'est pas en cours d'exécution. Cela peut entraîner une accumulation de messages dans la file d'attente de l'abonné. Cette génération peut être évitée si l'abonné s'enregistre à l'aide de l'option MQSO_PUBLICATIONS_ON_REQUEST. Vous pouvez également utiliser des abonnements non durables, le cas échéant, pour votre application afin d'éviter une accumulation de messages indésirables.

Si un abonnement est durable et qu'un diffuseur de publications utilise des publications conservées, l'application d'abonné peut utiliser l'appel MQSUBRQ pour actualiser ses informations d'état après un redémarrage. L'abonné doit ensuite actualiser périodiquement son état à l'aide de l'appel MQSUBRQ.

Aucune publication ne sera envoyée suite à l'appel MQSUB utilisant cette option. Un abonnement durable qui a été repris après la déconnexion utilisera l'option MQSO_PUBLICATIONS_ON_REQUEST si l'abonnement d'origine a été configuré pour utiliser cette option.

Nouvelles publications uniquement, MQSO_NEW_PUBLICATIONS_ONLY

Si une publication conservée existe sur une rubrique, les abonnés qui s'abonnent après la publication recevront une copie de cette publication. Si un abonné ne souhaite pas recevoir de publications antérieures à l'abonnement en cours, il peut utiliser l'option d'abonnement MQSO_NEW_PUBLICATIONS_ONLY.

Groupement des abonnements

Pensez à regrouper les abonnements si vous avez configuré une file d'attente pour recevoir des publications et que plusieurs abonnements se chevauchant alimentent les publications dans la même file d'attente. Cette situation est similaire à l'exemple de la rubrique Chevauchement d'abonnements.

Vous pouvez éviter de recevoir des publications en double en définissant l'option MQSO_GROUP_SUB lorsque vous vous abonnez à une rubrique. Par conséquent, lorsque plusieurs abonnements du groupe correspondent à la rubrique d'une publication, un seul abonnement est responsable de la mise en file d'attente de la publication. Les autres abonnements correspondant à la rubrique de publication sont ignorés.

L'abonnement chargé de placer la publication dans la file d'attente est choisi en fonction du fait qu'il possède la chaîne de rubrique correspondante la plus longue, avant de rencontrer des caractères génériques. Il peut être considéré comme l'abonnement correspondant le plus proche. Ses propriétés sont propagées à la publication, notamment si elle possède la propriété MQSO_NOT_OWN_PUBS . Si c'est le cas, aucune publication n'est distribuée dans la file d'attente, même si d'autres abonnements correspondants ne possèdent pas la propriété MQSO_NOT_OWN_PUBS .

Vous ne pouvez pas placer tous vos abonnements dans un seul groupe pour éliminer les publications en double. Les abonnements groupés doivent remplir les conditions suivantes:

1. Aucun des abonnements n'est géré.
2. Un groupe d'abonnements distribue des publications dans la même file d'attente.
3. Chaque abonnement doit être au même niveau d'abonnement.
4. Le message de publication de chaque abonnement du groupe possède le même identificateur de corrélation.

   Pour vous assurer que chaque abonnement génère un message de publication avec le même identificateur de corrélation, définissez MQSO_SET_CORREL_ID pour créer votre propre identificateur de corrélation dans la publication et définissez la même valeur dans la zone **SubCorrelId** de chaque abonnement. Ne définissez pas **SubCorrelId** sur la valeur MQCI_NONE.

Pour plus d'informations, voir ../refdev/q100080_.dita#q100080_/mqso_group_sub .

Interrogation et définition des attributs d'objet

Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

Elles affectent la manière dont un gestionnaire de files d'attente traite un objet. Les attributs de chaque type d'objet IBM MQ sont décrits en détail dans Attributs d'objets.

Certains attributs sont définis lorsque l'objet est défini et peuvent être modifiés uniquement à l'aide des commandes IBM MQ . Par exemple, la priorité par défaut des messages placés dans une file d'attente est un attribut de ce type. D'autres attributs sont affectés par le fonctionnement du gestionnaire de files d'attente et peuvent changer au fil du temps ; par exemple, la longueur en cours d'une file d'attente.

Vous pouvez vous renseigner sur les valeurs en cours de la plupart des attributs à l'aide de l'appel MQINQ. L'interface MQI fournit également un appel MQSET avec lequel vous pouvez modifier certains attributs de file d'attente. Vous ne pouvez pas utiliser les appels MQI pour modifier les attributs d'un autre type d'objet. A la place, vous devez utiliser l'une des ressources suivantes:

- ALW La fonction MQSC, décrite dans la rubrique Commandes MQSC.
- IBM i Les commandes CL CHGMQMx, qui sont décrites dans la rubrique Référence des commandes CL pour IBM i, ou la fonction MQSC.
- z/OS Les commandes de l'opérateur ALTER ou les commandes DEFINE avec l'option REPLACE, qui sont décrites dans Commandes MQSC.

Remarque : Les noms des attributs des objets sont affichés dans cette documentation sous la forme que vous utilisez avec les appels MQINQ et MQSET. Lorsque vous utilisez des commandes IBM MQ pour

définir, modifier ou afficher les attributs, vous devez les identifier à l'aide des mots clés affichés dans les descriptions des commandes dans les liens des rubriques.

Les appels MQINQ et MQSET utilisent des tableaux de sélecteurs pour identifier les attributs que vous souhaitez consulter ou définir. Il existe un sélecteur pour chaque attribut que vous pouvez utiliser. Le nom du sélecteur comporte un préfixe, déterminé par la nature de l'attribut:

Tableau 127. Préfixes des noms de sélecteur

Préfixe	**Description**
MQCA_	Ces sélecteurs font référence à des attributs qui contiennent des données de type caractères (par exemple, le nom d'une file d'attente).
MQIA_	Ces sélecteurs font référence à des attributs qui contiennent des valeurs numériques (telles que *CurrentQueueDepth*, le nombre de messages dans une file d'attente) ou une valeur constante (telle que *SyncPoint*, si le gestionnaire de files d'attente prend en charge les points de synchronisation).

Avant d'utiliser les appels MQINQ ou MQSET, votre application doit être connectée au gestionnaire de files d'attente et vous devez utiliser l'appel MQOPEN pour ouvrir l'objet afin de définir ou d'interroger des attributs. Ces opérations sont décrites dans «Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753 et «Ouverture et fermeture d'objets», à la page 760.

Utilisez les liens suivants pour en savoir plus sur l'interrogation et la définition des attributs d'objet:

- «Interrogation des attributs d'un objet», à la page 875
- «Certains cas où l'appel MQINQ échoue», à la page 875
- «Définition des attributs de file d'attente», à la page 876

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Interrogation des attributs d'un objet

Utilisez l'appel MQINQ pour vous renseigner sur les attributs de n'importe quel type de IBM MQ.

Comme entrée pour cet appel, vous devez fournir:

- Un descripteur de connexion.
- Un descripteur d'objet.
- Nombre de sélecteurs.
- Un tableau de sélecteurs d'attribut, chaque sélecteur ayant la forme MQCA_ * ou MQIA_ *. Chaque sélecteur représente un attribut avec une valeur que vous souhaitez interroger, et chaque sélecteur doit être valide pour le type d'objet représenté par le descripteur d'objet. Vous pouvez spécifier des sélecteurs dans n'importe quel ordre.
- Nombre d'attributs de type entier que vous interrogez. Indiquez zéro si vous ne vous interrogez pas sur les attributs de type entier.
- Longueur de la mémoire tampon des attributs de caractères dans *CharAttrLength*. Il doit s'agir au moins de la somme des longueurs requises pour contenir chaque chaîne d'attribut de caractère. Indiquez zéro si vous n'interrogez pas les attributs de caractères.

La sortie de MQINQ est la suivante:

- Ensemble de valeurs d'attribut entières copiées dans le tableau. Le nombre de valeurs est déterminé par *IntAttrCount*. Si *IntAttrCount* ou *SelectorCount* est égal à zéro, ce paramètre n'est pas utilisé.
- Mémoire tampon dans laquelle les attributs de caractères sont renvoyés. La longueur de la mémoire tampon est indiquée par le paramètre **CharAttrLength** . Si *CharAttrLength* ou *SelectorCount* est égal à zéro, ce paramètre n'est pas utilisé.
- Code achèvement. Si le code achèvement indique un avertissement, cela signifie que l'appel n'a été effectué que partiellement. Dans ce cas, examinez le code anomalie.
- Code anomalie. Il existe trois situations d'achèvement partiel:
  - Le sélecteur ne s'applique pas au type de file d'attente
  - Il n'y a pas assez d'espace autorisé pour les attributs de type entier
  - Il n'y a pas assez d'espace autorisé pour les attributs de caractères

  Si plusieurs de ces situations se produisent, la première qui s'applique est renvoyée.

Si vous ouvrez une file d'attente pour la sortie ou l'interrogation et qu'elle est résolue en file d'attente de cluster non locale, vous ne pouvez interroger que le nom de la file d'attente, le type de file d'attente et les attributs communs. Les valeurs des attributs communs sont celles de la file d'attente choisie si MQOO_BIND_ON_OPEN a été utilisé. Les valeurs sont celles d'une file d'attente de cluster arbitraire si MQOO_BIND_NOT_FIXED ou MQOOO_BIND_ON_GROUP a été utilisé ou si MQOO_BIND_AS_Q_DEF a été utilisé et que l'attribut de file d'attente **DefBind** était MQBND_BIND_NOT_FIXED. Pour plus d'informations, voir «MQOPEN et clusters», à la page 910 et MQOPEN .

Remarque : Les valeurs renvoyées par l'appel sont un instantané des attributs sélectionnés. Les attributs peuvent changer avant que votre programme n'agisse sur les valeurs renvoyées.

Il existe une description de l'appel MQINQ dans MQINQ.

Certains cas où l'appel MQINQ échoue

Si vous ouvrez un alias pour en savoir plus sur ses attributs, les attributs de la file d'attente alias (l'objet IBM MQ utilisé pour accéder à une autre file d'attente) vous sont renvoyés, et non ceux de la file d'attente de base.

Toutefois, la définition de la file d'attente de base dans laquelle l'alias est résolu est également ouverte par le gestionnaire de files d'attente et si un autre programme modifie l'utilisation de la file d'attente de

base dans l'intervalle entre vos appels MQOPEN et MQINQ, votre appel MQINQ échoue et renvoie le code anomalie MQRC_OBJECT_CHANGED. L'appel échoue également si les attributs de l'objet file d'attente alias sont modifiés.

De même, lorsque vous ouvrez une file d'attente éloignée pour en savoir plus sur ses attributs, les attributs de la définition locale de la file d'attente éloignée vous sont renvoyés uniquement.

Si vous spécifiez un ou plusieurs sélecteurs qui ne sont pas valides pour le type d'attributs de file d'attente que vous interrogez, l'appel MQINQ se termine avec un avertissement et définit la sortie comme suit:

- Pour les attributs de type entier, les éléments correspondants de *IntAttrs* sont définis sur MQIAV_NOT_APPLICABLE.
- Pour les attributs de caractères, les parties correspondantes de la chaîne *CharAttrs* sont définies sur des astérisques.

Si vous spécifiez un ou plusieurs sélecteurs qui ne sont pas valides pour le type d'attributs d'objet que vous interrogez, l'appel MQINQ échoue et renvoie le code anomalie MQRC_SELECTOR_ERROR.

Vous ne pouvez pas appeler MQINQ pour consulter une file d'attente modèle ; utilisez la fonction MQSC ou les commandes disponibles sur votre plateforme.

Définition des attributs de file d'attente

Utilisez ces informations pour apprendre à définir des attributs de file d'attente à l'aide de l'appel MQSET.

Vous pouvez définir uniquement les attributs de file d'attente suivants à l'aide de l'appel MQSET:

- *InhibitGet* (mais pas pour les files d'attente éloignées)
- Multi *DistList*
- *InhibitPut*
- *TriggerControl*
- *TriggerType*
- *TriggerDepth*
- *TriggerMsgPriority*
- *TriggerData*

L'appel MQSET possède les mêmes paramètres que l'appel MQINQ. Toutefois, pour MQSET, tous les paramètres à l'exception du code achèvement et du code anomalie sont des paramètres d'entrée. Il n'y a pas de situations d'exécution partielle.

Remarque : Vous ne pouvez pas utiliser l'interface MQI pour définir les attributs des objets IBM MQ autres que les files d'attente définies localement.

Pour plus de détails sur l'appel MQSET, voir MQSET.

Validation et annulation d'unités de travail

Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

Les termes suivants sont utilisés dans cette rubrique:

- Valider
- Renvoyer
- Coordination des points de synchronisation
- Point de synchronisation
- Unité d'oeuvre
- validation en une phase
- Validation en deux phases

Si vous connaissez bien ces termes de traitement des transactions, vous pouvez passer à «Remarques sur les points de synchronisation dans les applications IBM MQ», à la page 878.

Valider et revenir en arrière

Lorsqu'un programme place un message dans une file d'attente au sein d'une unité d'oeuvre, ce message est rendu visible par d'autres programmes uniquement lorsque le programme valide l'unité d'oeuvre. Pour qu'une unité d'oeuvre soit validée, toutes les mises à jour doivent aboutir afin de préserver l'intégrité des données. Si le programme détecte une erreur et décide que l'opération d'insertion n'est pas permanente, il peut rétablir l'unité d'oeuvre. Lorsqu'un programme effectue une annulation, IBM MQ restaure la file d'attente en supprimant les messages qui ont été placés dans la file d'attente par cette unité d'oeuvre. La manière dont le programme effectue les opérations de validation et d'exécution dépend de l'environnement dans lequel le programme est exécuté.

De même, lorsqu'un programme extrait un message d'une file d'attente d'une unité d'oeuvre, ce message reste dans la file d'attente jusqu'à ce que le programme valide l'unité d'oeuvre, mais le message n'est pas disponible pour être extrait par d'autres programmes. Le message est définitivement supprimé de la file d'attente lorsque le programme valide l'unité de travail. Si le programme annule l'unité d'oeuvre, IBM MQ restaure la file d'attente en rendant les messages disponibles pour être extraits par d'autres programmes.

Coordination des points de synchronisation, point de synchronisation, unité de travail

La *coordination des points de synchronisation* est le processus par lequel les unités de travail sont validées ou annulées avec l'intégrité des données.

La décision de valider ou d'éliminer les modifications est prise, dans le cas le plus simple, à la fin d'une transaction. Toutefois, il peut être plus utile pour une application de synchroniser les modifications de données à d'autres points logiques au sein d'une transaction. Ces points logiques sont appelés *points de synchronisation* (ou *points de synchronisation*) et la période de traitement d'un ensemble de mises à jour entre deux points de synchronisation est appelée *unité de travail*. Plusieurs appels MQGET et MQPUT peuvent faire partie d'une seule unité d'oeuvre.

Le nombre maximal de messages dans une unité de travail peut être contrôlé par l'attribut MAXUMSGS de la commande ALTER QMGR .

validation en une phase

Un processus de *validation en une seule phase* est un processus dans lequel un programme peut valider des mises à jour dans une file d'attente sans coordonner ses modifications avec d'autres gestionnaires de ressources.

Validation en deux phases

Un processus de *validation en deux phases* est un processus dans lequel les mises à jour apportées par un programme aux files d'attente IBM MQ peuvent être coordonnées avec les mises à jour apportées à d'autres ressources (par exemple, des bases de données sous le contrôle de Db2). Dans le cadre d'un tel processus, les mises à jour de toutes les ressources sont validées ou annulées ensemble.

Pour vous aider à gérer les unités de travail, IBM MQ fournit l'attribut **BackoutCount** . Cette valeur est incrémentée chaque fois qu'un message d'une unité de travail est annulé. Si le message provoque à plusieurs reprises la fin anormale de l'unité de travail, la valeur de *BackoutCount* est finalement supérieure à celle de *BackoutThreshold*. Cette valeur est définie lorsque la file d'attente est définie. Dans ce cas, l'application peut supprimer le message de l'unité de travail et le placer dans une autre file d'attente, comme défini dans *BackoutRequeueQName*. Lorsque le message est déplacé, l'unité de travail peut être validée.

Utilisez les liens suivants pour en savoir plus sur la validation et l'annulation des unités de travail:

- «Remarques sur les points de synchronisation dans les applications IBM MQ», à la page 878
- z/OS «Syncpoints in IBM MQ for z/OS applications», à la page 879
- IBM i «Points de synchronisation dans les applications CICS for IBM i», à la page 881
- «Points de synchronisation dans IBM MQ for Multiplatforms», à la page 882

- IBM i «Interfaces avec le gestionnaire de points de synchronisation externe IBM i», à la page 887

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Remarques sur les points de synchronisation dans les applications IBM MQ

Utilisez ces informations pour en savoir plus sur l'utilisation des points de synchronisation dans les applications IBM MQ .

La validation en deux phases est prise en charge par les environnements suivants:

- Multi IBM MQ for Multiplatforms
- z/OS CICS Transaction Server pour z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS Lot z/OS avec RRS
- Autres coordinateurs externes utilisant l'interface X/Open XA

La validation en une phase est prise en charge par les environnements suivants:

- Multi IBM MQ for Multiplatforms
- z/OS z/OS par lots

Pour plus d'informations sur les interfaces externes, voir «Interfaces avec les gestionnaires de points de synchronisation externes sur Multiplatforms», à la page 885et la documentation XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publiée par The Open Group. Les gestionnaires de transactions (tels que CICS, IMS, Encina et Tuxedo) peuvent participer à la validation en deux phases, coordonnée avec d'autres ressources récupérables. Cela signifie que les fonctions de mise en

file d'attente fournies par IBM MQ peuvent être placées dans la portée d'une unité de travail, gérée par le gestionnaire de transactions.

Les exemples fournis avec IBM MQ montrent IBM MQ la coordination des bases de données compatibles XA. Pour plus d'informations sur ces exemples, voir «Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085.

Dans votre application IBM MQ , vous pouvez spécifier sur chaque appel put et get si vous souhaitez que l'appel soit sous contrôle de point de synchronisation. Pour qu'une opération d'insertion fonctionne sous le contrôle d'un point de synchronisation, utilisez la valeur MQPMO_SYNCPOINT dans la zone *Options* de la structure MQPMO lorsque vous appelez MQPUT. Pour une opération d'extraction, utilisez la valeur MQGMO_SYNCPOINT dans la zone *Options* de la structure MQGMO. Si vous ne choisissez pas explicitement une option, l'action par défaut dépend de la plateforme:

- Multi La valeur par défaut du contrôle de point de synchronisation est NO.
- z/OS La valeur par défaut du contrôle de point de synchronisation est YES.

Lorsqu'un appel MQPUT1 est émis avec MQPMO_SYNCPOINT, le comportement par défaut change, de sorte que l'opération d'insertion est exécutée de manière asynchrone. Cela peut entraîner une modification du comportement de certaines applications qui s'appuient sur certaines zones des structures MQOD et MQMD renvoyées, mais qui contiennent désormais des valeurs non définies. Une application peut spécifier MQPMO_SYNC_RESPONSE pour s'assurer que l'opération d'insertion est effectuée de manière synchrone et que toutes les valeurs de zone appropriées sont terminées.

Lorsque votre application reçoit un code anomalie MQRC_BACKED_OUT en réponse à un MQPUT ou MQGET sous un point de synchronisation, elle doit normalement rétablir la transaction en cours à l'aide de MQBACK, puis, le cas échéant, relancer l'intégralité de la transaction. Si l'application reçoit MQRC_BACKED_OUT en réponse à un appel MQCMIT ou MQDISC, elle n'a pas besoin d'appeler MQBACK.

Chaque fois qu'un appel MQGET est annulé, la zone *BackoutCount* de la structure MQMD du message affecté est incrémentée. Un *BackoutCount* élevé indique un message qui a été annulé à plusieurs reprises. Cela peut indiquer un problème lié à ce message, que vous devez examiner. Voir BackoutCount pour plus de détails sur *BackoutCount*.

Si un programme émet l'appel MQDISC alors qu'il existe des demandes non validées, un point de synchronisation implicite se produit (sauf sur le z/OS lot avec RRS). Si le programme s'arrête de manière anormale, une annulation implicite se produit.

z/OS Sous z/OS, un point de synchronisation implicite se produit également si le programme se termine normalement sans avoir d'abord appelé MQDISC. Le programme est considéré comme s'étant arrêté normalement si le bloc de contrôle des tâches connecté à MQ s'arrête normalement. Lors de l'exécution sous z/OS UNIX System Services et Language Environment (LE), le traitement des conditions par défaut est appelé pour les fins anormales ou les signaux. Les gestionnaires de conditions LE traitent la condition d'erreur et le bloc de contrôle des tâches se termine normalement. Dans ces conditions, MQ valide l'unité de travail. Pour plus d'informations, voir Introduction à Language Environment Condition Handling.

z/OS Pour les programmes IBM MQ for z/OS , vous pouvez utiliser l'option MQGMO_MARK_SKIP_BACKOUT pour indiquer qu'un message ne doit pas être annulé en cas d'annulation (afin d'éviter une boucle *MQGET-error-backout*). Pour plus d'informations sur l'utilisation de cette option, voir «Annulation ignorée», à la page 820.

Les modifications apportées aux attributs de file d'attente (par l'appel MQSET ou par des commandes) ne sont pas affectées par la validation ou l'annulation des unités d'oeuvre.

z/OS *Syncpoints in IBM MQ for z/OS applications*

This topic explains how to use syncpoints in transaction manager (CICS and IMS) and batch applications.

z/OS *Syncpoints in CICS Transaction Server for z/OS applications*

In a CICS application you establish a syncpoint by using the EXEC CICS SYNCPOINT command.

To back out all changes to the previous syncpoint, you can use the EXEC CICS SYNCPOINT ROLLBACK command. For more information, see the *CICS Application Programming Reference*.

If other recoverable resources are involved in the unit of work, the queue manager (in conjunction with the CICS syncpoint manager) participates in a two-phase commit protocol; otherwise, the queue manager performs a single-phase commit process.

If a CICS application issues the MQDISC call, no implicit syncpoint is taken. If the application closes down normally, any open queues are closed and an implicit commit occurs. If the application closes down abnormally, any open queues are closed and an implicit backout occurs.

z/OS *Syncpoints in IMS applications*

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

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

The queue manager (in conjunction with the IMS syncpoint manager) participates in a two-phase commit protocol if other recoverable resources are also involved in the unit of work.

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

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

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

z/OS *Syncpoints in z/OS batch applications*

For batch applications, you can use the IBM MQ syncpoint management calls: MQCMIT and MQBACK. For compatibility with earlier versions, CSQBCMT and CSQBBAK are available as synonyms.

Note: If you need to commit or back out updates to resources managed by different resource managers, such as IBM MQ and Db2, within a single unit of work you can use RRS. For further information see “Transaction management and recoverable resource manager services” on page 881.

Committing changes using the MQCMIT call

As input, you must supply the connection handle (*Hconn*) that is returned by the MQCONN or MQCONNX call.

The output from MQCMIT is a completion code and a reason code. The call completes with a warning if the syncpoint was completed but the queue manager backed out the put and get operations since the previous syncpoint.

Successful completion of the MQCMIT call indicates to the queue manager that the application has reached a syncpoint and that all put and get operations made since the previous syncpoint have been made permanent.

Not all failure responses mean that the MQCMIT did not complete. For example, the application can receive MQRC_CONNECTION_BROKEN.

There is a description of the MQCMIT call in MQCMIT.

Backing out changes using the MQBACK call

As input, you must supply a connection handle (*Hconn*). Use the handle that is returned by the MQCONN or MQCONNX call.

The output from MQBACK is a completion code and a reason code.

The output indicates to the queue manager that the application has reached a syncpoint and that all gets and puts that have been made since the last syncpoint have been backed out.

There is a description of the MQBACK call in MQBACK.

Transaction management and recoverable resource manager services

Transaction management and recoverable resource manager services (RRS) is a z/OS facility to provide two-phase syncpoint support across participating resource managers.

An application can update recoverable resources managed by various z/OS resource managers such as IBM MQ and Db2, and then commit or back out these updates as a single unit of work. RRS provides the necessary unit-of-work status logging during normal execution, coordinates the syncpoint processing, and provides appropriate unit-of-work status information during subsystem restart.

IBM MQ for z/OS RRS participant support enables IBM MQ applications in the batch, TSO, and Db2 stored procedure environments to update both IBM MQ and non-IBM MQ resources (for example, Db2) within a single logical unit of work. For information about RRS participant support, see *z/OS MVS Programming: Resource Recovery*.

Your IBM MQ application can use either MQCMIT and MQBACK or the equivalent RRS calls, SRRCMIT and SRRBACK. See “The RRS batch adapter” on page 917 for more information.

RRS availability

If RRS is not active on your z/OS system, any IBM MQ call issued from a program linked with either RRS stub (CSQBRSTB or CSQBRRSI) returns MQRC_ENVIRONMENT_ERROR.

Db2 stored procedures

If you use Db2 stored procedures with RRS, be aware of the following:

- Db2 stored procedures that use RRS must be managed by workload manager (WLM-managed).
- If a Db2-managed stored procedure contains IBM MQ calls, and it is linked with either RRS stub (CSQBRSTB or CSQBRRSI), the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR.
- If a WLM-managed stored procedure contains IBM MQ calls, and is linked with a non-RRS stub, the MQCONN or MQCONNX call returns MQRC_ENVIRONMENT_ERROR, unless it is the first IBM MQ call executed since the stored procedure address space started.
- If your Db2 stored procedure contains IBM MQ calls and is linked with a non-RRS stub, IBM MQ resources updated in that stored procedure are not committed until the stored procedure address space ends, or until a subsequent stored procedure does an MQCMIT (using an IBM MQ Batch/TSO stub).
- Multiple copies of the same stored procedure can execute concurrently in the same address space. Ensure that your program is coded in a reentrant manner if you want Db2 to use a single copy of your stored procedure. Otherwise you might receive MQRC_HCONN_ERROR on any IBM MQ call in your program.
- Do not code MQCMIT or MQBACK in a WLM-managed Db2 stored procedure.
- Design all programs to run in Language Environment (LE).

IBM i *Points de synchronisation dans les applications CICS for IBM i*

IBM MQ for IBM i participe à CICS pour les unités de travail IBM i . Vous pouvez utiliser l'interface MQI dans une application CICS for IBM i pour insérer et extraire des messages dans l'unité d'oeuvre en cours.

Vous pouvez utiliser la commande EXEC CICS SYNCPOINT pour établir un point de synchronisation qui inclut les opérations IBM MQ for IBM i . Pour annuler toutes les modifications jusqu'au point de synchronisation précédent, vous pouvez utiliser la commande EXEC CICS SYNCPOINT ROLLBACK.

Si vous utilisez l'option MQPUT, MQPUT1ou MQGET avec l'option MQPMO_SYNCPOINT ou MQGMO_SYNCPOINT dans une application CICS for IBM i , vous ne pouvez pas vous déconnecter de CICS for IBM i tant que IBM MQ for IBM i n'a pas supprimé son enregistrement en tant que ressource de validation d'API. Validez ou déconnectez les opérations d'insertion ou d'extraction en attente avant de vous déconnecter du gestionnaire de files d'attente. Vous pouvez ainsi vous déconnecter de CICS for IBM i.

Multi *Points de synchronisation dans IBM MQ for Multiplatforms*

La prise en charge des points de synchronisation fonctionne sur deux types d'unités de travail: locale et globale.

Une unité d'oeuvre *locale* est une unité d'oeuvre dans laquelle les seules ressources mises à jour sont celles du gestionnaire de files d'attente IBM MQ. Dans ce cas, la coordination des points de synchronisation est assurée par le gestionnaire de files d'attente lui-même à l'aide d'une procédure de validation en une phase.

Une unité d'oeuvre *globale* est une unité dans laquelle les ressources appartenant à d'autres gestionnaires de ressources, tels que les bases de données, sont également mises à jour. IBM MQ peut coordonner ces unités de travail. Ils peuvent également être coordonnés par un contrôleur d'engagement externe. Exemple :

- Un autre gestionnaire de transactions
- IBM i Le contrôleur de validation IBM i

Pour une intégrité totale, utilisez une procédure de validation en deux phases. La validation en deux phases peut être fournie par des gestionnaires de transactions et des bases de données compatibles XA. Exemple :

- TXSeries
- Universal Database
- IBM i le contrôleur de validation IBM i

ALW Les produits IBM MQ peuvent coordonner des unités de travail globales à l'aide d'un processus de validation en deux phases.

IBM i IBM MQ for IBM i peut agir en tant que gestionnaire de ressources pour les unités d'oeuvre globales dans un environnement WebSphere Application Server , mais ne peut pas agir en tant que gestionnaire de transactions.

Point de synchronisation implicite

Lors de l'insertion de messages persistants, IBM MQ est optimisé pour l'insertion de messages persistants sous un point de synchronisation. Plusieurs applications plaçant des messages persistants dans la même file d'attente sont plus performantes si ces applications utilisent le point de synchronisation. Cela est dû au fait qu'il y a moins de conflits pour la file d'attente, si le point de synchronisation est utilisé pour insérer des messages persistants.

ImplSyncOpenOutput ajoute un point de synchronisation implicite lorsque les applications placent des messages persistants en dehors du point de synchronisation. Cela permet d'améliorer les performances, sans que les applications soient conscientes du point de synchronisation implicite.

Le point de synchronisation implicite n'offre une amélioration des performances que lorsque plusieurs applications sont placées dans la file d'attente, car il réduit les conflits de la file d'attente. Ainsi, **ImplSyncOpenOutput** indique le nombre minimal d'applications ayant une file d'attente ouverte pour la sortie avant l'ajout d'un point de synchronisation implicite. La valeur par défaut est 2. Cela signifie

que si vous ne spécifiez pas **ImplSyncOpenOutput**, le point de synchronisation implicite est ajouté uniquement si plusieurs applications sont placées dans la file d'attente.

Pour plus d'informations, voir Paramètres d'optimisation .

Unités locales de travail sur Multiplatforms
Les unités de travail qui impliquent uniquement le gestionnaire de files d'attente sont appelées unités de travail *locales* . La coordination des points de synchronisation est assurée par le gestionnaire de files d'attente lui-même (coordination interne) à l'aide d'un processus de validation en une phase.

Pour démarrer une unité d'oeuvre locale, l'application émet des demandes MQGET, MQPUT ou MQPUT1 en spécifiant l'option de point de synchronisation appropriée. L'unité de travail est validée à l'aide de MQCMIT ou annulée à l'aide de MQBACK. Toutefois, l'unité d'oeuvre se termine également lorsque la connexion entre l'application et le gestionnaire de files d'attente est interrompue, intentionnellement ou non.

Si une application se déconnecte (MQDISC) d'un gestionnaire de files d'attente alors qu'une unité d'oeuvre globale coordonnée par IBM MQ est toujours active, une tentative de validation de l'unité d'oeuvre est effectuée. Toutefois, si l'application s'arrête sans se déconnecter, l'unité de travail est annulée car l'application est considérée comme s'étant terminée de façon anormale.

ALW *Unités d'oeuvre globales sur AIX, Linux, and Windows*
Utilisez des unités d'oeuvre globales lorsque vous devez également inclure des mises à jour des ressources appartenant à d'autres gestionnaires de ressources. La coordination peut être interne ou externe au gestionnaire de files d'attente.

Coordination des points de synchronisation internes

La coordination des unités d'oeuvre globales par le gestionnaire de files d'attente n'est pas prise en charge dans un environnement IBM MQ MQI client.

Ici, IBM MQ effectue la coordination. Pour démarrer une unité d'oeuvre globale, l'application émet l'appel MQBEGIN.

En tant qu'entrée de l'appel MQBEGIN, vous devez fournir le descripteur de connexion (*Hconn*) renvoyé par l'appel MQCONN ou MQCONNX. Cet identificateur représente la connexion au gestionnaire de files d'attente IBM MQ .

L'application émet des demandes MQGET, MQPUT ou MQPUT1 en spécifiant l'option de point de synchronisation appropriée. Cela signifie que vous pouvez utiliser MQBEGIN pour lancer une unité de travail globale qui met à jour les ressources locales, les ressources appartenant à d'autres gestionnaires de ressources, ou les deux. Les mises à jour apportées aux ressources appartenant à d'autres gestionnaires de ressources sont effectuées à l'aide de l'API de ce gestionnaire de ressources. Toutefois, vous ne pouvez pas utiliser l'interface MQI pour mettre à jour des files d'attente qui appartiennent à d'autres gestionnaires de files d'attente. Emettez MQCMIT ou MQBACK avant de démarrer d'autres unités de travail (locales ou globales).

L'unité de travail globale est validée à l'aide de MQCMIT ; cette opération lance une validation en deux phases de tous les gestionnaires de ressources impliqués dans l'unité de travail. Un processus de validation en deux phases est utilisé dans lequel les gestionnaires de ressources (par exemple, les gestionnaires de base de données compatibles XA tels que Db2, Oracleet Sybase) sont d'abord invités à se préparer à la validation. Ce n'est que si tous sont préparés qu'ils sont invités à s'engager. Si un gestionnaire de ressources signale qu'il ne peut pas effectuer de validation, il est demandé à chacun d'entre eux de le faire à la place. Vous pouvez également utiliser MQBACK pour annuler les mises à jour de tous les gestionnaires de ressources.

Si une application se déconnecte (MQDISC) alors qu'une unité d'oeuvre globale est toujours active, l'unité d'oeuvre est validée. Toutefois, si l'application s'arrête sans se déconnecter, l'unité de travail est annulée car l'application est considérée comme s'étant terminée de façon anormale.

La sortie de MQBEGIN est un code achèvement et un code anomalie.

Lorsque vous utilisez MQBEGIN pour démarrer une unité d'oeuvre globale, tous les gestionnaires de ressources externes qui ont été configurés avec le gestionnaire de files d'attente sont inclus. Toutefois, l'appel démarre une unité de travail mais se termine avec un avertissement si:

- Il n'existe aucun gestionnaire de ressources participant (c'est-à-dire qu'aucun gestionnaire de ressources n'a été configuré avec le gestionnaire de files d'attente)

ou

- Un ou plusieurs gestionnaires de ressources ne sont pas disponibles.

Dans ces cas, l'unité de travail doit inclure uniquement les mises à jour des gestionnaires de ressources qui étaient disponibles lorsque l'unité de travail a été démarrée.

Si l'un des gestionnaires de ressources ne peut pas valider ses mises à jour, tous les gestionnaires de ressources sont invités à annuler leurs mises à jour et MQCMIT se termine avec un avertissement. Dans des circonstances inhabituelles (en général, une intervention de l'opérateur), un appel MQCMIT peut échouer si certains gestionnaires de ressources valident leurs mises à jour mais que d'autres les annulent ; le travail est considéré comme terminé avec un résultat *mixte* . Ces occurrences sont diagnostiquées dans le journal des erreurs du gestionnaire de files d'attente afin que des mesures correctives puissent être prises.

Un MQCMIT d'une unité d'oeuvre globale aboutit si tous les gestionnaires de ressources impliqués valident leurs mises à jour.

Pour une description de l'appel MQBEGIN, voir MQBEGIN.

Coordination des points de synchronisation externes

Cela se produit lorsqu'un coordinateur de point de synchronisation autre que IBM MQ a été sélectionné ; par exemple, CICS, Encina ou Tuxedo.

Dans cette situation, les systèmes IBM MQ for AIX, Linux, and Windows enregistrent leur intérêt pour le résultat de l'unité de travail avec le coordinateur de point de synchronisation afin qu'ils puissent valider ou annuler des opérations d'extraction ou d'insertion non validées, selon les besoins. Le coordinateur de point de synchronisation externe détermine si des protocoles de validation en une ou deux phases sont fournis.

Lorsque vous utilisez un coordinateur externe, MQCMIT, MQBACK et MQBEGIN ne peuvent pas être émis. Les appels à ces fonctions échouent avec le code anomalie MQRC_ENVIRONMENT_ERROR.

Le mode de démarrage d'une unité de travail coordonnée en externe dépend de l'interface de programmation fournie par le coordinateur du point de synchronisation. Un appel explicite peut être requis. Si un appel explicite est requis et que vous émettez un appel MQPUT spécifiant l'option MQPMO_SYNCPOINT lorsqu'une unité de travail n'est pas démarrée, le code achèvement MQRC_SYNCPOINT_NOT_AVAILABLE est renvoyé.

La portée de l'unité de travail est déterminée par le coordinateur du point de synchronisation. L'état de la connexion entre l'application et le gestionnaire de files d'attente affecte la réussite ou l'échec des appels MQI émis par une application, et non l'état de l'unité d'oeuvre. Une application peut, par exemple, se déconnecter et se reconnecter à un gestionnaire de files d'attente pendant une unité d'oeuvre active et effectuer d'autres opérations MQGET et MQPUT dans la même unité d'oeuvre. Il s'agit d'une déconnexion en attente.

Vous pouvez utiliser les appels d'API IBM MQ dans les programmes CICS , que vous choisissiez d'utiliser les fonctions XA de CICS. Si vous n'utilisez pas XA, les insertions et les extractions de messages vers et depuis les files d'attente ne seront pas gérées dans les unités de travail atomiques CICS . L'une des raisons pour lesquelles vous avez choisi cette méthode est que la cohérence globale de l'unité de travail n'est pas importante pour vous.

Si l'intégrité de vos unités de travail est importante pour vous, vous devez utiliser XA. Lorsque vous utilisez XA, CICS utilise un protocole de validation en deux phases pour s'assurer que toutes les ressources de l'unité de travail sont mises à jour ensemble.

Pour plus d'informations sur la configuration du support transactionnel, voir Scénarios de support transactionnel, ainsi que la documentation TXSeries CICS , par exemple *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Multi *Point de synchronisation implicite sur Multiplatforms*

La prise en charge du point de synchronisation implicite active les insertions de messages persistants en dehors du point de synchronisation.

Lors de l'insertion de messages persistants, IBM MQ est optimisé pour l'insertion de messages persistants sous un point de synchronisation. Plusieurs applications qui placent simultanément des messages persistants dans la même file d'attente sont généralement plus performantes si ces applications utilisent un point de synchronisation. En effet, la stratégie de verrouillage de IBM MQ est plus efficace si le point de synchronisation est utilisé lors de l'insertion de messages persistants.

Le paramètre **ImplSyncOpenOutput** du fichier `qm.ini` contrôle si un point de synchronisation implicite peut être ajouté lorsque des applications placent des messages persistants en dehors du point de synchronisation. Cela peut améliorer les performances, sans que les applications soient conscientes du point de synchronisation implicite.

Le point de synchronisation implicite n'offre une amélioration des performances que lorsque plusieurs applications sont placées simultanément dans la file d'attente, car il réduit les conflits d'accès. **ImplSyncOpenOutput** indique le nombre minimal d'applications ayant une file d'attente ouverte pour la sortie avant qu'un point de synchronisation implicite puisse être ajouté. La valeur par défaut est *2*. En d'autres termes, si vous ne spécifiez pas explicitement **ImplSyncOpenOutput**, le point de synchronisation implicite est ajouté uniquement si plusieurs applications sont placées dans la file d'attente.

Si vous ajoutez un point de synchronisation implicite, les statistiques le reflètent et vous pouvez voir une sortie de transaction de **runmqsc display conn**.

Définissez **ImplSyncOpenOutput**=*OFF* si vous ne souhaitez jamais qu'un point de synchronisation implicite soit ajouté.

Pour plus d'informations, voir Paramètres d'optimisation .

Interfaces avec les gestionnaires de points de synchronisation externes sur Multiplatforms

IBM MQ for Multiplatforms prend en charge la coordination des transactions par les gestionnaires de points de synchronisation externes qui utilisent l'interface X/Open XA.

Certains gestionnaires de transactions XA (TXSeries) requièrent que chaque gestionnaire de ressources XA fournisse son nom. Il s'agit de la chaîne appelée `name` dans la structure de commutateur XA.

- ALW Le gestionnaire de ressources pour IBM MQ sous AIX, Linux, and Windows est nommé MQSeries_XA_RMI.
- IBM i Pour IBM i, le nom du gestionnaire de ressources est MQSeries XA RMI.

Pour plus de détails sur les interfaces XA, voir la documentation XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publiée par The Open Group.

Dans une configuration XA, IBM MQ for Multiplatforms remplit le rôle de gestionnaire de ressources XA. Un coordinateur de point de synchronisation XA peut gérer un ensemble de gestionnaires de ressources XA et synchroniser la validation ou l'annulation des transactions dans les deux gestionnaires de ressources. Voici comment il fonctionne pour un gestionnaire de ressources enregistré de manière statique:

1. Une application informe le coordinateur de point de synchronisation qu'elle souhaite démarrer une transaction.
2. Le coordinateur de point de synchronisation émet un appel à tous les gestionnaires de ressources qu'il connaît pour les informer de la transaction en cours.

3. L'application émet des appels pour mettre à jour les ressources gérées par les gestionnaires de ressources associés à la transaction en cours.
4. L'application demande au coordinateur du point de synchronisation de valider ou d'annuler la transaction.
5. Le coordinateur de point de synchronisation émet des appels à chaque gestionnaire de ressources à l'aide de protocoles de validation en deux phases pour terminer la transaction comme demandé.

La spécification XA requiert que chaque gestionnaire de ressources fournisse une structure appelée commutateur XA. Cette structure déclare les fonctionnalités du gestionnaire de ressources et les fonctions qui doivent être appelées par le coordinateur de point de synchronisation.

Il existe deux versions de cette structure:

Tableau 128. Versions du commutateur XA

Version	**Description**
MQRMIXASwitch	Gestion des ressources XA statiques
MQRMIXASwitchDynamic	Gestion des ressources XA dynamiques

Pour obtenir la liste des bibliothèques contenant cette structure, voir Structure du commutateur XA IBM MQ.

La méthode qui doit être utilisée pour les lier à un coordinateur de point de synchronisation XA est définie par le coordinateur ; consultez la documentation fournie par ce coordinateur pour déterminer comment permettre à IBM MQ de coopérer avec votre coordinateur de point de synchronisation XA.

La structure *xa_info* transmise lors d'un appel *xa_open* par le coordinateur de point de synchronisation peut être le nom du gestionnaire de files d'attente à administrer. Il prend la même forme que le nom de gestionnaire de files d'attente transmis à MQCONN ou MQCONNX et peut être vide si le gestionnaire de files d'attente par défaut doit être utilisé. Toutefois, vous pouvez utiliser les deux paramètres supplémentaires TPM et AXLIB

TPM vous permet d'indiquer à IBM MQ le nom du gestionnaire de transactions, par exemple, CICS. AXLIB permet d'indiquer le nom réel de la bibliothèque dans le gestionnaire de transactions où se trouvent les points d'entrée XA AX.

Si vous utilisez l'un de ces paramètres ou un gestionnaire de files d'attente autre que celui par défaut, vous devez spécifier le nom du gestionnaire de files d'attente à l'aide du paramètre QMNAME. Pour plus d'informations, voir Les paramètres CHANNEL, TRPTYPE, CONNAME et QMNAME de la chaîne xa_open.

Restrictions

1. Les unités d'oeuvre globales ne sont pas autorisées avec un Hconn partagé (comme décrit dans «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 757.
2. IBM i IBM MQ for IBM i ne prend pas en charge l'enregistrement dynamique des gestionnaires de ressources XA.

   Le seul gestionnaire de transactions pris en charge est WebSphere Application Server.
3. Windows Sur les systèmes Windows , toutes les fonctions déclarées dans le commutateur XA sont déclarées en tant que fonctions _cdecl.
4. Un coordinateur de point de synchronisation externe ne peut administrer qu'un seul gestionnaire de files d'attente à la fois. En effet, le coordinateur dispose d'une connexion effective à chaque gestionnaire de files d'attente et est donc soumis à la règle selon laquelle une seule connexion est autorisée à la fois.

   Remarque : Remarque: Une application client JMS (application CLIENT JEE) exécutée sur un serveur JEE ne fait pas l'objet de cette restriction, de sorte qu'une seule transaction gérée par le serveur JEE peut coordonner plusieurs gestionnaires de files d'attente dans la même transaction. Toutefois, une

application serveur JMS , s'exécutant en mode liaisons, est toujours soumise à la règle selon laquelle une seule connexion est autorisée à la fois.

5. Toutes les applications qui sont exécutées à l'aide du coordinateur de point de synchronisation ne peuvent se connecter qu'au gestionnaire de files d'attente administré par le coordinateur car elles sont déjà effectivement connectées à ce gestionnaire de files d'attente. Ils doivent émettre MQCONN ou MQCONNX pour obtenir un descripteur de connexion et doivent émettre MQDISC avant de quitter. Ils peuvent également utiliser l'exit UE014015 pour TXSeries CICS.

IBM i *Interfaces avec le gestionnaire de points de synchronisation externe IBM i*

IBM MQ for IBM i peut utiliser le contrôle de validation IBM i natif comme coordinateur de point de synchronisation externe.

Les connexions indépendantes de l'unité d'exécution (partagées) ne sont pas autorisées avec le contrôle de validation. Pour plus d'informations sur les fonctions de contrôle de validation d' IBM i, voir *IBM i Programming: Backup and Recovery Guide, SC21-8079* .

Pour démarrer les fonctions de contrôle de validation IBM i , utilisez la commande système STRCMTCTL. Pour arrêter le contrôle de validation, utilisez la commande système ENDCMTCTL.

Remarque : La valeur par défaut de *Portée de la définition de validation* est *ACTGRP. Il doit être défini comme *JOB pour IBM MQ for IBM i. Exemple :

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

IBM MQ for IBM i peut également effectuer des unités de travail locales contenant uniquement des mises à jour des ressources IBM MQ . Le choix entre les unités de travail locales et la participation dans les unités de travail globales coordonnées par IBM i est effectué dans chaque application lorsque l'application appelle MQPUT, MQPUT1ou MQGET, en spécifiant MQPMO_SYNCPOINT ou MQGMO_SYNCPOINT ou MQBEGIN. Si le contrôle de validation n'est pas actif lorsque le premier appel de ce type est émis, IBM MQ démarre une unité d'oeuvre locale et toutes les autres unités d'oeuvre pour cette connexion à IBM MQ utilisent également des unités d'oeuvre locales, que le contrôle de validation soit démarré ou non. Pour valider une unité d'oeuvre locale, utilisez MQCMIT. Pour rétablir une unité d'oeuvre locale, utilisez MQBACK. Les appels de validation et d'invalidation IBM i , tels que la commande CL COMMIT, n'ont aucun effet sur les unités d'oeuvre locales IBM MQ .

Si vous souhaitez utiliser IBM MQ for IBM i avec le contrôle de validation IBM i natif en tant que coordinateur de point de synchronisation externe, vérifiez que tout travail avec le contrôle de validation est actif et que vous utilisez IBM MQ dans un travail à unité d'exécution unique. Si vous appelez MQPUT, MQPUT1ou MQGET en spécifiant MQPMO_SYNCPOINT ou MQGMO_SYNCPOINT dans un travail à unités d'exécution multiples dans lequel le contrôle de validation a été démarré, l'appel échoue avec le code anomalie MQRC_SYNCPOINT_NOT_AVAILABLE.

Il est possible d'utiliser des unités d'oeuvre locales et les appels MQCMIT et MQBACK dans un travail à unités d'exécution multiples.

Si vous appelez MQPUT, MQPUT1ou MQGET, en spécifiant MQPMO_SYNCPOINT ou MQGMO_SYNCPOINT, après avoir démarré le contrôle de validation, IBM MQ for IBM i s'ajoute en tant que ressource de validation d'API à la définition de validation. Il s'agit généralement du premier appel de ce type dans un travail. Si des ressources de validation d'API sont enregistrées dans une définition de validation particulière, vous ne pouvez pas mettre fin au contrôle de validation pour cette définition.

IBM MQ for IBM i supprime son enregistrement en tant que ressource de validation d'API lorsque vous vous déconnectez du gestionnaire de files d'attente, s'il n'y a pas d'opérations MQI en attente dans l'unité de travail en cours.

Si vous vous déconnectez du gestionnaire de files d'attente alors que des opérations MQPUT, MQPUT1ou MQGET sont en attente dans l'unité d'oeuvre en cours, IBM MQ for IBM i reste enregistré en tant que ressource de validation d'API afin d'être informé de la prochaine validation ou annulation. Lorsque le point de synchronisation suivant est atteint, IBM MQ for IBM i valide ou annule les modifications selon les besoins. Une application peut se déconnecter et se reconnecter à un gestionnaire de files d'attente

pendant une unité d'oeuvre active et effectuer d'autres opérations MQGET et MQPUT dans la même unité d'oeuvre (il s'agit d'une déconnexion en attente).

Si vous tentez d'émettre une commande système ENDCMTCTL pour cette définition de validation, le message CPF8355 est émis, indiquant que les modifications en attente étaient actives. Ce message apparaît également dans l'historique du travail à la fin du travail. Pour éviter cela, validez ou annulez toutes les opérations IBM MQ for IBM i en attente et déconnectez-vous du gestionnaire de files d'attente. Ainsi, l'utilisation des commandes COMMIT ou ROLLBACK avant ENDCMTCTL permet au contrôle de fin de validation de s'exécuter correctement.

Lorsque vous utilisez le contrôle de validation IBM i en tant que coordinateur de point de synchronisation externe, vous ne pouvez pas émettre d'appels MQCMIT, MQBACK et MQBEGIN. Les appels à ces fonctions échouent avec le code anomalie MQRC_ENVIRONMENT_ERROR.

Pour valider ou annuler (c'est-à-dire annuler) votre unité de travail, utilisez l'un des langages de programmation prenant en charge le contrôle de validation. Exemple :

- Commandes CL: COMMIT et ROLLBACK
- Fonctions de programmation ILE C: _Rcommit et _Rrollback
- ILE RPG: COMMIT et ROLBK
- COBOL/400: COMMIT et ROLLBACK

Lorsque vous utilisez le contrôle de validation IBM i en tant que coordinateur de point de synchronisation externe avec IBM MQ for IBM i, IBM i exécute un protocole de validation en deux phases auquel participe IBM MQ . Etant donné que chaque unité de travail est validée en deux phases, le gestionnaire de files d'attente peut devenir indisponible pour la deuxième phase après avoir voté pour la validation dans la première phase. Cela peut se produire, par exemple, si les travaux internes du gestionnaire de files d'attente sont arrêtés. Dans cette situation, le journal des travaux exécutant la validation contient le message CPF835F indiquant qu'une opération de validation ou d'invalidation a échoué. Les messages qui précèdent indiquent la cause de l'incident, qu'il se soit produit lors d'une opération de validation ou d'invalidation, ainsi que l'ID d'unité d'oeuvre logique (LUWID) de l'unité d'oeuvre défaillante.

Si l'incident est dû à l'échec de la ressource de validation de l'API IBM MQ lors de la validation ou de l'invalidation d'une unité d'oeuvre préparée, vous pouvez utiliser la commande WRKMQMTRN pour terminer l'opération et restaurer l'intégrité de la transaction. La commande requiert que vous connaissiez l'identificateur LUWID de l'unité d'oeuvre à valider et à sauvegarder.

Démarrage des applications IBM MQ à l'aide de déclencheurs

Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

Certaines applications IBM MQ qui servent des files d'attente s'exécutent en continu, de sorte qu'elles sont toujours disponibles pour extraire les messages qui arrivent dans les files d'attente. Toutefois, vous pouvez ne pas le vouloir lorsque le nombre de messages arrivant dans les files d'attente est imprévisible. Dans ce cas, les applications peuvent consommer des ressources système même lorsqu'il n'y a pas de messages à extraire.

IBM MQ fournit une fonction qui permet de démarrer automatiquement une application lorsque des messages sont disponibles pour être extraits. Cette fonction est appelée *déclenchement*.

Pour plus d'informations sur le déclenchement de canaux, voir Déclenchement de canaux.

Qu'est-ce qui se déclenche?

Le gestionnaire de files d'attente définit certaines conditions comme constituant des *événements déclencheurs*.

Si le déclenchement est activé pour une file d'attente et qu'un événement déclencheur se produit, le gestionnaire de files d'attente envoie un *message de déclenchement* à une file d'attente appelée *file d'attente d'initialisation*. La présence du message de déclenchement dans la file d'attente d'initialisation indique qu'un événement déclencheur s'est produit.

Les messages de déclenchement générés par le gestionnaire de files d'attente ne sont pas persistants. Cela réduit la consignation (ce qui améliore les performances) et réduit les doublons lors du redémarrage, ce qui améliore le temps de redémarrage.

Le programme qui traite la file d'attente d'initialisation est appelé *application de moniteur de déclenchement*et sa fonction est de lire le message de déclenchement et de prendre les mesures appropriées, en fonction des informations contenues dans le message de déclenchement. En règle générale, cette action consiste à démarrer une autre application pour traiter la file d'attente qui a généré le message de déclenchement. Du point de vue du gestionnaire de files d'attente, il n'y a rien de particulier dans l'application du moniteur de déclenchement ; il s'agit simplement d'une autre application qui lit les messages d'une file d'attente (la file d'attente d'initialisation).

Si le déclenchement est activé pour une file d'attente, vous pouvez créer un *objet de définition de processus* qui lui est associé. Cet objet contient des informations sur l'application qui traite le message à l'origine de l'événement déclencheur. Si l'objet de définition de processus est créé, le gestionnaire de files d'attente extrait ces informations et les place dans le message de déclenchement, à utiliser par l'application trigger-monitor. Le nom de la définition de processus associée à une file d'attente est donné par l'attribut de file d'attente locale *ProcessName* . Chaque file d'attente peut spécifier une définition de processus différente ou plusieurs files d'attente peuvent partager la même définition de processus.

Si vous souhaitez déclencher le démarrage d'un canal, vous n'avez pas besoin de définir un objet de définition de processus. La définition de file d'attente de transmission est utilisée à la place.

Le déclenchement est pris en charge par les clients IBM MQ s'exécutant sur AIX, Linux, and Windows. Une application s'exécutant dans un environnement client est identique à une application s'exécutant dans un environnement IBM MQ complet, sauf que vous la liez aux bibliothèques client. Toutefois, le moniteur de déclenchement et l'application à démarrer doivent tous deux se trouver dans le même environnement.

Le déclenchement implique:

File d"attente d'application
Une *file d'attente d'application* est une file d'attente locale qui, lorsque le déclenchement est activé et que les conditions sont remplies, requiert l'écriture de messages de déclenchement.

Définition de processus
Une file d'attente d'application peut être associée à un *objet de définition de processus* qui contient les détails de l'application qui va extraire des messages de la file d'attente d'application. (Pour obtenir la liste des attributs, voir Attributs des définitions de processus .)

N'oubliez pas que si vous souhaitez qu'un déclencheur démarre un canal, vous n'avez pas besoin de définir un objet de définition de processus.

File d'attente de transmission
Vous avez besoin d'une file d'attente de transmission si vous souhaitez qu'un déclencheur démarre un canal.

Pour une file d'attente de transmission sur toute plateforme autre que Linux, l'attribut *TriggerData* de la file d'attente de transmission peut spécifier le nom du canal à démarrer. Cette option peut remplacer la définition de processus pour le déclenchement des canaux, mais elle est utilisée uniquement lorsqu'une définition de processus n'est pas créée.

Evénement déclencheur
Un *événement déclencheur* est un événement qui provoque la génération d'un message de déclenchement par le gestionnaire de files d'attente. Il s'agit généralement d'un message arrivant dans une file d'attente d'application, mais il peut également se produire à d'autres moments. Par exemple, voir «Conditions d'un événement déclencheur», à la page 895.

IBM MQ dispose d'une série d'options qui vous permettent de contrôler les conditions à l'origine d'un événement déclencheur (voir «Contrôle des événements déclencheurs», à la page 900).

Message de déclenchement
Le gestionnaire de files d'attente crée un *message de déclenchement* lorsqu'il reconnaît un événement déclencheur. Il copie dans le message de déclenchement les informations relatives à l'application à démarrer. Ces informations proviennent de la file d'attente d'application et de l'objet de définition de processus associé à la file d'attente d'application.

Les messages de déclenchement ont un format fixe (voir «Format des messages de déclenchement», à la page 908).

File d'initialisation

Une *file d'attente d'initialisation* est une file d'attente locale dans laquelle le gestionnaire de files d'attente insère des messages de déclenchement. Notez qu'une file d'attente d'initialisation ne peut pas être une file d'attente d'alias ou une file d'attente modèle.

Un gestionnaire de files d'attente peut posséder plusieurs files d'attente d'initialisation et chacune d'elles est associée à une ou plusieurs files d'attente d'application.

z/OS Une file d'attente partagée, une file d'attente locale accessible par les gestionnaires de files d'attente d'un groupe de partage de files d'attente, peut être une file d'attente d'initialisation dans IBM MQ for z/OS.

moniteur de déclenchement

Un *moniteur de déclenchement* est un programme en cours d'exécution qui sert une ou plusieurs files d'attente d'initialisation. Lorsqu'un message de déclenchement arrive dans une file d'attente d'initialisation, le moniteur de déclenchement le récupère. Le moniteur de déclenchement utilise les informations du message de déclenchement. Il émet une commande pour démarrer l'application qui consiste à extraire les messages arrivant dans la file d'attente d'application, en transmettant les informations contenues dans l'en-tête de message de déclencheur, qui inclut le nom de la file d'attente d'application.

Sur toutes les plateformes, un moniteur de déclenchement spécial appelé initiateur de canal est chargé de démarrer les canaux.

z/OS Sous z/OS, l'initiateur de canal est généralement démarré manuellement ou peut être exécuté automatiquement lorsqu'un gestionnaire de files d'attente démarre en modifiant CSQINP2 dans le JCL de démarrage du gestionnaire de files d'attente.

Multi Sous Multiplateformes, l'initiateur de canal est démarré automatiquement au démarrage du gestionnaire de files d'attente ou il peut être démarré manuellement à l'aide de la commande **runmqchi** .

Pour plus d'informations, voir «Traitement de la file d'attente d'initialisation par les moniteurs de déclenchement», à la page 904.

Pour comprendre le fonctionnement du déclenchement, prenez en compte Figure 95, à la page 891, qui est un exemple de type de déclencheur FIRST (MQTT_FIRST).

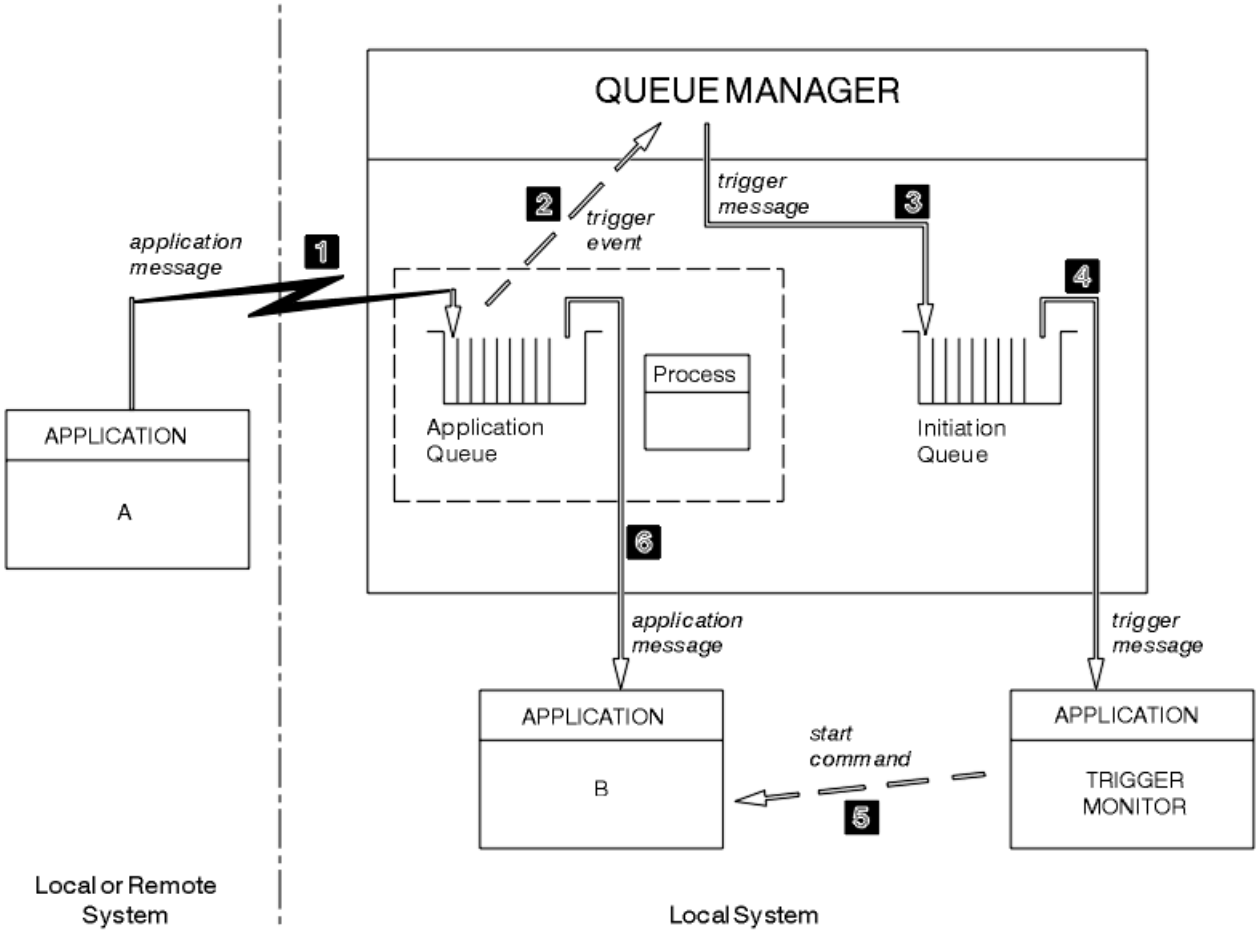


Figure 95. Flux de messages d'application et de déclenchement

Dans Figure 95, à la page 891, la séquence d'événements est la suivante:

1. L'application A, qui peut être locale ou éloignée du gestionnaire de files d'attente, place un message dans la file d'attente de l'application. Aucune application n'a cette file d'attente ouverte pour l'entrée. Toutefois, ce fait n'est pertinent que pour le type de déclencheur FIRST et DEPTH.
2. Le gestionnaire de files d'attente vérifie si les conditions dans lesquelles il doit générer un événement déclencheur sont remplies. Ils le sont et un événement déclencheur est généré. Les informations contenues dans l'objet de définition de processus associé sont utilisées lors de la création du message de déclenchement.
3. Le gestionnaire de files d'attente crée un message de déclenchement et le place dans la file d'attente d'initialisation associée à cette file d'attente d'application, mais uniquement si une application (moniteur de déclenchement) a la file d'attente d'initialisation ouverte pour entrée.
4. Le moniteur de déclenchement extrait le message de déclenchement de la file d'attente d'initialisation.
5. Le moniteur de déclenchement émet une commande pour démarrer l'application B (l'application serveur).
6. L'application B ouvre la file d'attente d'application et extrait le message.

Remarque :

1. Si la file d'attente d'application est ouverte en entrée, par n'importe quel programme, et que le déclenchement est défini pour FIRST ou DEPTH, aucun événement déclencheur ne se produit car la file d'attente est déjà en cours de traitement.

2. Si la file d'attente d'initialisation n'est pas ouverte en entrée, le gestionnaire de files d'attente ne génère pas de messages de déclenchement ; il attend qu'une application ouvre la file d'attente d'initialisation en entrée.
3. Lorsque vous utilisez le déclenchement pour les canaux, utilisez le type de déclencheur FIRST ou DEPTH.
4. Les applications déclenchées s'exécutent sous l'ID utilisateur et le groupe de l'utilisateur qui a démarré le moniteur de déclenchement, l'utilisateur CICS ou l'utilisateur qui a démarré le gestionnaire de files d'attente.

Jusqu'à présent, la relation entre les files d'attente dans le déclenchement n'a été que sur une base un à un. Prenez en compte Figure 96, à la page 892.

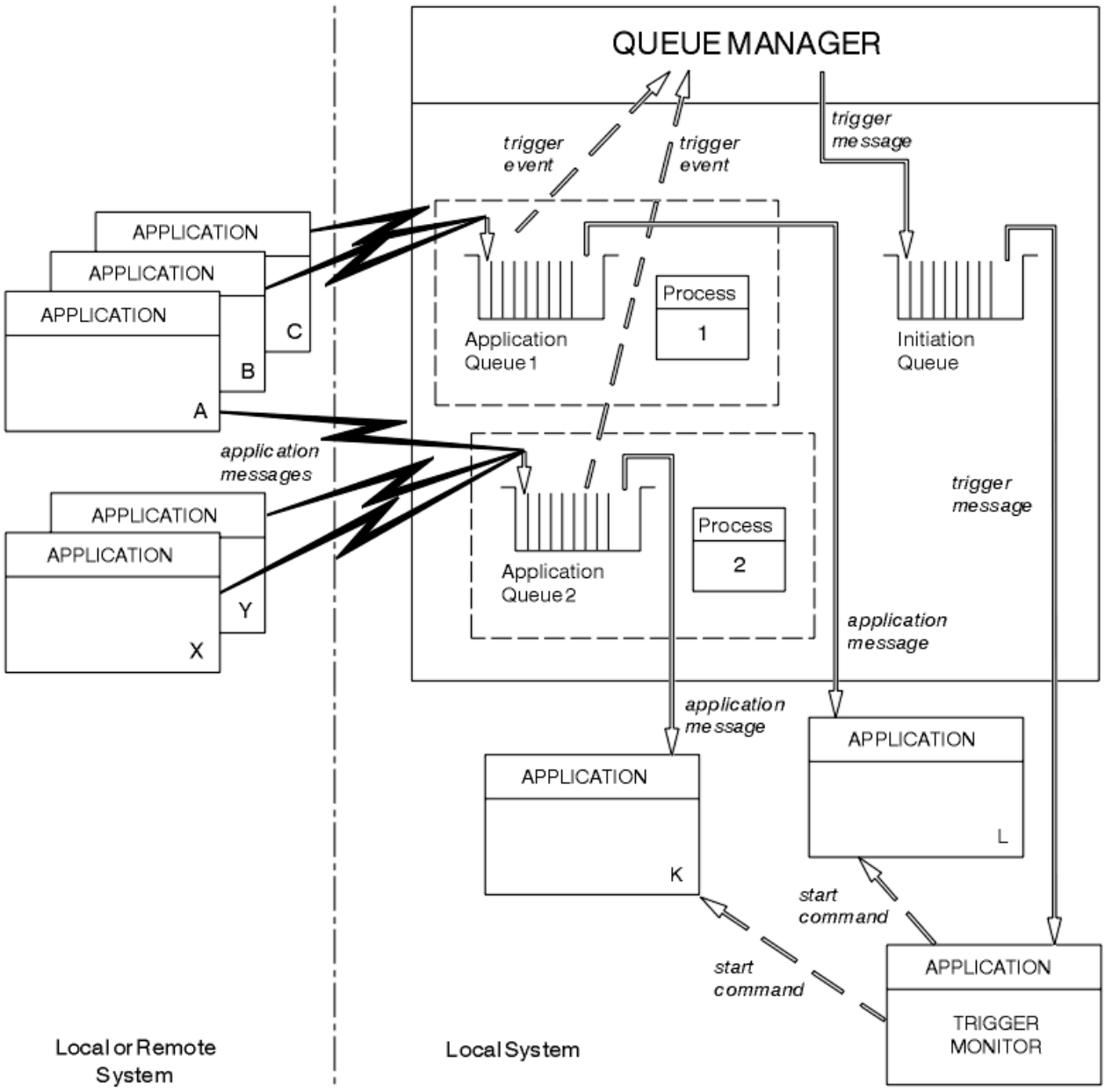


Figure 96. Relation des files d'attente dans le déclenchement

Une file d'attente d'application est associée à un objet de définition de processus qui contient les détails de l'application qui traitera le message. Le gestionnaire de files d'attente place les informations dans le message de déclenchement, de sorte qu'une seule file d'attente d'initialisation est nécessaire.

Le moniteur de déclenchement extrait ces informations du message de déclenchement et démarre l'application appropriée pour traiter le message dans chaque file d'attente d'application.

N'oubliez pas que, si vous souhaitez déclencher le démarrage d'un canal, vous n'avez pas besoin de définir un objet de définition de processus. La définition de la file d'attente de transmission peut déterminer le canal à déclencher.

Utilisez les liens suivants pour en savoir plus sur le démarrage d'applications IBM MQ à l'aide de déclencheurs:

- «Prérequis pour le déclenchement», à la page 893
- «Conditions d'un événement déclencheur», à la page 895
- «Contrôle des événements déclencheurs», à la page 900
- «Conception d'une application qui utilise des files d'attente déclenchées», à la page 902
- «Traitement de la file d'attente d'initialisation par les moniteurs de déclenchement», à la page 904
- «Propriétés des messages de déclenchement», à la page 907
- «Lorsque le déclenchement ne fonctionne pas», à la page 909

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Utilisation de l'interface MQI et des clusters», à la page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

Prérequis pour le déclenchement

Utilisez ces informations pour en savoir plus sur les étapes à suivre avant d'utiliser le déclenchement.

Avant que votre application puisse tirer parti du déclenchement, procédez comme suit:

1. L'un ou l'autre :
   a. Créez une file d'attente d'initialisation pour votre file d'attente d'application. Exemple :

```
DEFINE QLOCAL (initiation.queue) REPLACE      +
```

```
          LIKE (SYSTEM.DEFAULT.INITIATION.QUEUE)      +
          DESCR ('initiation queue description')
```

ou

b. Déterminez le nom d'une file d'attente locale qui existe et qui peut être utilisée par votre application (généralement, ce nom est SYSTEM.DEFAULT.INITIATION.QUEUE ou, si vous démarrez des canaux avec des déclencheurs, SYSTEM.CHANNEL.INITQ) et indiquez son nom dans la zone *InitiationQName* de la file d'attente d'application.

2. Associez la file d'attente d'initialisation à la file d'attente d'application. Un gestionnaire de files d'attente peut posséder plusieurs files d'attente d'initialisation. Vous pouvez souhaiter que certaines de vos files d'attente d'application soient servies par des programmes différents, auquel cas, vous pouvez utiliser une file d'attente d'initialisation pour chaque programme de service, même si vous n'avez pas besoin de le faire. Voici un exemple de création d'une file d'attente d'application:

```
DEFINE QLOCAL (application.queue) REPLACE   +
LIKE (SYSTEM.DEFAULT.LOCAL.QUEUE)           +
DESCR ('appl queue description')    +
INITQ (initiation.queue)          +
PROCESS (process.name)            +
TRIGGER                                     +
TRIGTYPE (FIRST)
```

IBM i Voici un extrait d'un programme CL pour IBM MQ for IBM i qui crée une file d'attente d'initialisation:

```
/*   Queue used by AMQSINQA                                        */
           CRTMQMQ     QNAME('SYSTEM.SAMPLE.INQ')                  +
                       QTYPE(*LCL)  REPLACE(*YES)                  +
                       MQMNAME                                     +
                       TEXT('queue for AMQSINQA')                  +
                       SHARE(*YES)                /* Shareable   */+
                       DFTMSGPST(*YES)/* Persistent messages OK  */+
                                                                   +
                       TRGENBL(*YES)  /* Trigger control on      */+
                       TRGTYPE(*FIRST)/* Trigger on first message*/+
                       PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       INITQNAME('SYSTEM.SAMPLE.TRIGGER')
```

3. Si vous déclenchez une application, créez un objet de définition de processus contenant des informations relatives à l'application qui doit servir votre file d'attente d'application. Par exemple, pour déclencher-démarrer une transaction de paie CICS appelée PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i Voici un extrait d'un programme CL pour IBM MQ for IBM i qui crée un objet de définition de processus:

```
/*   Process definition                                            */
           CRTMQMPRC   PRCNAME('SYSTEM.SAMPLE.INQPROCESS')         +
                       REPLACE(*YES)                               +
                       MQMNAME                                     +
                       TEXT('trigger process for AMQSINQA')        +
                       ENVDATA('JOBPTY(3)') /* Submit parameter  */+
                       APPID('AMQSINQA')    /* Program name      */
```

Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il copie les informations des attributs de l'objet de définition de processus dans le message de déclenchement.

Plateforme	Pour créer un objet de définition de processus
AIX, Linux, and Windows systèmes	Utilisez DEFINE PROCESS ou SYSTEM.DEFAULT.PROCESS et modification à l'aide de ALTER PROCESS
z/OS z/OS z/OS	Utilisez DEFINE PROCESS (voir l'exemple de code à l'étape «3», à la page 894) ou utilisez les panneaux d'opérations et de contrôle.
IBM i IBM i IBM i	Utilisez un programme CL contenant du code comme à l'étape «3», à la page 894.

4. Facultatif: créez une définition de file d'attente de transmission et utilisez des blancs pour l'attribut **ProcessName** .

   L'attribut **TrigData** peut contenir le nom du canal à déclencher ou il peut être laissé vide ; sauf sur IBM MQ for z/OS, s'il est laissé vide, l'initiateur de canal recherche les fichiers de définition de canal jusqu'à ce qu'il trouve un canal associé à la file d'attente de transmission nommée. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il copie les informations de l'attribut **TrigData** de la définition de file d'attente de transmission dans le message de déclenchement.

5. Si vous avez créé un objet de définition de processus pour spécifier les propriétés de l'application devant servir votre file d'attente d'application, associez l'objet de processus à votre file d'attente d'application en le nommant dans l'attribut **ProcessName** de la file d'attente.

Plateforme	Utiliser des commandes
AIX, Linux, and Windows systèmes	ALTER QLOCAL
z/OS z/OS z/OS	ALTER QLOCAL
IBM i IBM i IBM i	CHGMQMQ

6. Démarrez les instances des moniteurs de déclenchement IBM i (ou des serveurs de déclenchement dans IBM MQ for IBM i) qui doivent servir les files d'attente d'initialisation que vous avez définies. Pour plus d'informations, voir «Traitement de la file d'attente d'initialisation par les moniteurs de déclenchement», à la page 904.

Si vous souhaitez connaître les messages de déclenchement non distribués, assurez-vous que votre gestionnaire de files d'attente dispose d'une file d'attente de rebut (messages non livrés) définie. Indiquez le nom de la file d'attente dans la zone du gestionnaire de files d'attente *DeadLetterQName* .

Vous pouvez ensuite définir les conditions de déclenchement dont vous avez besoin, à l'aide des attributs de l'objet file d'attente qui définit votre file d'attente d'application. Pour plus d'informations, voir «Contrôle des événements déclencheurs», à la page 900.

Conditions d'un événement déclencheur

Le gestionnaire de files d'attente crée un message de déclenchement lorsque certaines conditions sont remplies.

Les conditions suivantes entraînent la création d'un message de déclenchement par le gestionnaire de files d'attente:

1. Un message est *inséré* dans une file d'attente.

2. Le message a une priorité supérieure ou égale à la priorité de déclenchement de seuil de la file d'attente. Cette priorité est définie dans l'attribut de file d'attente locale **TriggerMsgPriority** ; si elle est définie sur zéro, tout message est qualifié.
3. Le nombre de messages dans la file d'attente avec une priorité supérieure ou égale à *TriggerMsgPriority* était précédemment, en fonction de *TriggerType*:
   - Zéro (pour le type de déclencheur MQTT_FIRST)
   - N'importe quel nombre (pour le type de déclencheur MQTT_EVERY)
   - *TriggerDepth* moins 1 (pour le type de déclencheur MQTT_DEPTH)

   Remarque :

   - Pour les files d'attente locales non partagées, le gestionnaire de files d'attente compte les messages validés et non validés lorsqu'il évalue si les conditions d'un événement déclencheur existent. Par conséquent, une application peut être démarrée lorsqu'il n'y a pas de messages à extraire car les messages de la file d'attente n'ont pas été validés. Dans ce cas, envisagez d'utiliser l'option d'attente avec un *WaitInterval*approprié, de sorte que l'application attende l'arrivée de ses messages.
   - z/OS Pour les files d'attente partagées locales, le gestionnaire de files d'attente compte uniquement les messages validés.
4. Pour le déclenchement de type FIRST ou DEPTH, aucun programme ne dispose de la file d'attente d'application ouverte pour la suppression de messages (c'est-à-dire que l'attribut de file d'attente locale **OpenInputCount** a pour valeur zéro).

   Remarque : z/OS

   - Pour les files d'attente partagées, des conditions spéciales s'appliquent lorsque plusieurs gestionnaires de files d'attente ont des moniteurs de déclenchement s'exécutant sur une file d'attente. Dans cette situation, si un ou plusieurs gestionnaires de files d'attente ont la file d'attente ouverte pour l'entrée partagée, les critères de déclenchement sur les autres gestionnaires de files d'attente sont traités comme *TriggerType* MQTT_FIRST et *TriggerMsgPriority* zéro. Lorsque tous les gestionnaires de files d'attente ferment la file d'attente pour entrée, les conditions de déclenchement reviennent aux conditions spécifiées dans la définition de file d'attente.

     Un exemple de scénario affecté par cette condition est constitué de plusieurs gestionnaires de files d'attente QM1, QM2et QM3 avec un moniteur de déclenchement en cours d'exécution pour une file d'attente d'application A. Un message arrive sur A satisfaisant les conditions de déclenchement et un message de déclenchement est généré dans la file d'attente d'initialisation. Le moniteur de déclenchement sur QM1 extrait le message de déclenchement et déclenche une application. L'application déclenchée ouvre la file d'attente d'application pour l'entrée partagée. A partir de ce point, les conditions de déclenchement de la file d'attente d'application A sont évaluées comme *TriggerType* MQTT_FIRST et *TriggerMsgPriority* zéro sur les gestionnaires de files d'attente QM2 et QM3, jusqu'à ce que QM1 ferme la file d'attente d'application.
   - Pour les files d'attente partagées, cette condition est appliquée pour chaque gestionnaire de files d'attente. Autrement dit, le *OpenInputCount* d'un gestionnaire de files d'attente pour une file d'attente doit être égal à zéro pour qu'un message de déclenchement soit généré pour la file d'attente par ce gestionnaire de files d'attente. Toutefois, si un gestionnaire de files d'attente du groupe de partage de files d'attente a la file d'attente ouverte à l'aide de l'option MQOO_INPUT_EXCLUSIVE, aucun message de déclenchement n'est généré pour cette file d'attente par l'un des gestionnaires de files d'attente du groupe de partage de files d'attente.

     Le changement dans la façon dont les conditions de déclenchement sont évaluées se produit lorsque l'application déclenchée ouvre la file d'attente pour entrée. Dans les scénarios où un seul moniteur de déclenchement est en cours d'exécution, d'autres applications peuvent avoir le même effet car elles ouvrent de la même manière la file d'attente d'application pour l'entrée. Peu importe que la file d'attente d'application ait été ouverte par une application démarrée par un moniteur de déclenchement ou par une autre application ; c'est le fait que la file d'attente soit ouverte pour

une entrée sur un autre gestionnaire de files d'attente qui entraîne la modification des critères de déclenchement.

5. z/OS Sous IBM MQ for z/OS, si la file d'attente d'application est une file d'attente dont l'attribut **Usage** est MQUS_NORMAL, les demandes d'extraction ne sont pas interdites (c'est-à-dire que l'attribut de file d'attente **InhibitGet** est MQQA_GET_ALLOWED). De plus, si la file d'attente d'application déclenchée est une file d'attente dont l'attribut **Usage** est MQUS_XMITQ, les demandes d'extraction pour cette file d'attente ne sont pas interdites.
6. L'un ou l'autre :
   - L'attribut de file d'attente locale **ProcessName** de la file d'attente n'est pas vide et l'objet de définition de processus identifié par cet attribut a été créé, ou
   - L'attribut de file d'attente locale **ProcessName** de la file d'attente est vide, mais la file d'attente est une file d'attente de transmission. Comme la définition de processus est facultative, l'attribut **TriggerData** peut également contenir le nom du canal à démarrer. Dans ce cas, le message de déclenchement contient des attributs avec les valeurs suivantes:
     - **QName**: nom de la file d'attente
     - **ProcessName**: blancs
     - **TriggerData**: données de déclenchement
     - **ApplType**: MQAT_INCONNU
     - **ApplId**: blancs
     - **EnvData**: blancs
     - **UserData**: blancs
7. Une file d'attente d'initialisation a été créée et a été spécifiée dans l'attribut de file d'attente locale **InitiationQName** . De même :
   - Les demandes d'extraction ne sont pas interdites pour la file d'attente d'initialisation (c'est-à-dire que la valeur de l'attribut de file d'attente **InhibitGet** est MQQA_GET_ALLOWED).
   - Les demandes d'insertion ne doivent pas être interdites pour la file d'attente d'initialisation (c'est-à-dire que la valeur de l'attribut de file d'attente **InhibitPut** doit être MQQA_PUT_ALLOWED).
   - La valeur de l'attribut **Usage** de la file d'attente d'initialisation doit être MQUS_NORMAL.
   - Dans les environnements où les files d'attente dynamiques sont prises en charge, la file d'attente d'initialisation ne doit pas être une file d'attente dynamique qui a été marquée comme supprimée logiquement.
8. Un moniteur de déclenchement dispose actuellement de la file d'attente d'initialisation ouverte pour la suppression des messages (c'est-à-dire que l'attribut de file d'attente locale **OpenInputCount** est supérieur à zéro).
9. Le contrôle de déclenchement (attribut de file d'attente locale**TriggerControl**) de la file d'attente d'application est défini sur MQTC_ON. Pour ce faire, définissez l'attribut **trigger** lorsque vous définissez votre file d'attente ou utilisez la commande ALTER QLOCAL.
10. Le type de déclencheur (attribut de file d'attente locale**TriggerType**) n'est pas MQTT_NONE.

    Si toutes les conditions requises sont remplies et que le message à l'origine de la condition de déclenchement est inséré dans une unité de travail, le message de déclenchement ne peut pas être récupéré par l'application du moniteur de déclenchement tant que l'unité de travail n'est pas terminée, que l'unité de travail soit validée ou, pour le type de déclencheur MQTT_FIRST ou MQTT_DEPTH, annulée.
11. Un message approprié est placé dans la file d'attente, pour un **TriggerType** de MQTT_FIRST ou MQTT_DEPTH, et la file d'attente:
    - N'était pas vide auparavant (MQTT_FIRST) ou
    - Au moins **TriggerDepth** messages (MQTT_DEPTH)

et les conditions «2», à la page 896 à «10», à la page 897 (à l'exception de «3», à la page 896) sont satisfaites si, dans le cas de MQTT_FIRST, un intervalle suffisant (attribut de gestionnaire de files d'attente**TriggerInterval**) s'est écoulé depuis l'écriture du dernier message de déclenchement pour cette file d'attente.

Cela permet à un serveur de file d'attente de se terminer avant de traiter tous les messages de la file d'attente. L'objectif de l'intervalle de déclenchement est de réduire le nombre de messages de déclenchement en double générés.

Remarque : Si vous arrêtez et redémarrez le gestionnaire de files d'attente, le temporisateur **TriggerInterval** est réinitialisé. Il existe une petite fenêtre pendant laquelle il est possible de produire deux messages de déclenchement. La fenêtre existe lorsque l'attribut de déclencheur de la file d'attente est défini sur activé en même temps qu'un message arrive et que la file d'attente n'était pas vide auparavant (MQTT_FIRST) ou qu'elle comportait **TriggerDepth** messages ou plus (MQTT_DEPTH).

12. La seule application servant une file d'attente émet un appel MQCLOSE, pour un **TriggerType** de MQTT_FIRST ou MQTT_DEPTH, et il existe au moins:

    - Un (MQTT_FIRST) ou
    - **TriggerDepth** (MQTT_DEPTH)

    Les messages de la file d'attente dont la priorité est suffisante (condition «2», à la page 896) et les conditions «6», à la page 897 à «10», à la page 897 sont également satisfaits.

    Cela permet à un serveur de files d'attente d'exécuter un appel MQGET, de trouver la file d'attente vide et ainsi de se terminer. Toutefois, dans l'intervalle entre les appels MQGET et MQCLOSE, un ou plusieurs messages arrivent.

    Remarque :

    a. Si le programme servant la file d'attente d'application n'extrait pas tous les messages, cela peut provoquer une boucle fermée. Chaque fois que le programme ferme la file d'attente, le gestionnaire de files d'attente crée un autre message de déclenchement qui oblige le moniteur de déclenchement à redémarrer le programme serveur.

    b. Si le programme qui sert la file d'attente d'application annule sa demande d'extraction (ou si le programme s'arrête de façon anormale) avant de fermer la file d'attente, la même chose se produit. Toutefois, si le programme ferme la file d'attente avant d'annuler la demande d'extraction et que la file d'attente est vide, aucun message de déclenchement n'est créé.

    c. Pour éviter qu'une telle boucle ne se produise, utilisez la zone *BackoutCount* de MQMD pour détecter les messages qui sont annulés à plusieurs reprises. Pour plus d'informations, voir «Messages annulés», à la page 48.

13. Les conditions suivantes sont satisfaites à l'aide de MQSET ou d'une commande:

    a. • **TriggerControl** est remplacé par MQTC_ON, ou
    • **TriggerControl** est déjà MQTC_ON et la valeur de **TriggerType**, **TriggerMsgPriority**ou **TriggerDepth** (le cas échéant) est modifiée,

    et il y a au moins:

    - Un (MQTT_FIRST ou MQTT_EVERY) ou
    - **TriggerDepth** (MQTT_DEPTH)

    messages dans la file d'attente de priorité suffisante (condition «2», à la page 896) et conditions «4», à la page 896 à «10», à la page 897 (à l'exclusion de «8», à la page 897) sont également satisfaits.

    Cela permet à une application ou à un opérateur de modifier les critères de déclenchement, lorsque les conditions d'un déclencheur sont déjà remplies.

    b. La valeur de l'attribut de file d'attente **InhibitPut** d'une file d'attente d'initialisation passe de MQQA_PUT_ALLOWED à MQQA_PUT_ALLOWED, et au moins:

- Un (MQTT_FIRST ou MQTT_EVERY) ou
- **TriggerDepth** (MQTT_DEPTH)

messages de priorité suffisante (condition «2», à la page 896) sur l'une des files d'attente pour lesquelles il s'agit de la file d'attente d'initialisation, et les conditions «4», à la page 896 à «10», à la page 897 sont également satisfaites. (Un message de déclenchement est généré pour chaque file d'attente répondant aux conditions.)

Cela permet de ne pas générer de messages de déclenchement en raison de la condition MQQA_PUT_INHIBÉE dans la file d'attente d'initialisation, mais cette condition a été modifiée.

c. La valeur de l'attribut de file d'attente **InhibitGet** d'une file d'attente d'application passe de MQQA_GET_ALLOWED à MQQA_GET_ALLOWED, et au moins:

- Un (MQTT_FIRST ou MQTT_EVERY) ou
- **TriggerDepth** (MQTT_DEPTH)

messages de priorité suffisante (condition «2», à la page 896) sur la file d'attente et les conditions «4», à la page 896 à «10», à la page 897, à l'exception de «5», à la page 897, sont également satisfaites.

Cela permet aux applications d'être déclenchées uniquement lorsqu'elles peuvent extraire des messages de la file d'attente d'application.

d. Une application de moniteur de déclenchement émet un appel MQOPEN pour l'entrée à partir d'une file d'attente d'initialisation et il existe au moins:

- Un (MQTT_FIRST ou MQTT_EVERY) ou
- **TriggerDepth** (MQTT_DEPTH)

messages de priorité suffisante (condition «2», à la page 896) sur les files d'attente d'application pour lesquelles il s'agit de la file d'attente d'initialisation et les conditions «4», à la page 896 à «10», à la page 897 (à l'exception de «8», à la page 897) sont également satisfaites, et aucune autre application n'a la file d'attente d'initialisation ouverte pour l'entrée (un message de déclenchement est généré pour chaque file d'attente répondant aux conditions).

Cela permet d'autoriser l'arrivée de messages dans des files d'attente alors que le moniteur de déclenchement n'est pas en cours d'exécution, ainsi que le redémarrage du gestionnaire de files d'attente et la perte de messages de déclenchement (non persistants).

14. MSGDLVSQ est défini correctement. Si vous définissez MSGDLVSQ=FIFO, les messages sont distribués à la file d'attente selon la méthode du premier entré, premier sorti. La priorité du message est ignorée et la priorité par défaut de la file d'attente est affectée au message. Si **TriggerMsgPriority** est défini sur une valeur supérieure à la priorité par défaut de la file d'attente, aucun message n'est déclenché. Si **TriggerMsgPriority** est défini sur une valeur inférieure ou égale à la priorité par défaut de la file d'attente, le déclenchement est effectué pour le type FIRST, EVERY et DEPTH. Pour plus d'informations sur ces types, voir la description de la zone **TriggerType** sous «Contrôle des événements déclencheurs», à la page 900.

    Si vous définissez MSGDLVSQ=PRIORITY et que la priorité du message est supérieure ou égale à la zone *TriggerMsgPriority* , les messages ne sont pris en compte que pour un événement déclencheur. Dans ce cas, le déclenchement se produit pour le type FIRST, EVERY et DEPTH. Par exemple, si vous placez 100 messages de priorité inférieure à celle de **TriggerMsgPriority**, la longueur effective de la file d'attente à des fins de déclenchement reste égale à zéro. Si vous placez ensuite un autre message dans la file d'attente, mais que cette fois la priorité est supérieure ou égale à **TriggerMsgPriority**, la longueur effective de la file d'attente passe de zéro à un et la condition de **TriggerType** FIRST est satisfaite.

Remarques :

1. A partir de l'étape «12», à la page 898 (où les messages de déclenchement sont générés suite à un événement autre qu'un message arrivant dans la file d'attente de l'application), le message de déclenchement n'est pas inséré dans une unité de travail. En outre, si **TriggerType** est MQTT_EVERY

et s'il existe un ou plusieurs messages dans la file d'attente de l'application, un seul message de déclenchement est généré.

2. Si IBM MQ segmente un message pendant MQPUT, un événement déclencheur ne sera pas traité tant que tous les segments n'auront pas été placés dans la file d'attente. Toutefois, une fois que les segments de message sont dans la file d'attente, IBM MQ les traite comme des messages individuels à des fins de déclenchement. Par exemple, un message logique unique divisé en trois parties entraîne le traitement d'un seul événement déclencheur lorsqu'il est d'abord MQPUT et segmenté. Cependant, chacun des trois segments entraîne le traitement de ses propres événements déclencheurs à mesure qu'ils sont déplacés via le réseau IBM MQ .
3. z/OS Pour IBM MQ for z/OS, si une file d'attente partagée est configurée pour le déclenchement et que la connexion à l'unité de couplage hébergeant la file d'attente partagée est perdue, un événement déclencheur peut être généré et un message inséré dans la file d'attente d'initialisation. Cela peut se produire même si aucun message n'a été inséré dans la configuration de la file d'attente partagée d'origine pour le déclenchement. Cela est dû à la surindication des bits par la macro IXLVECTR, comme indiqué dans The List Notification Vector.

Contrôle des événements déclencheurs

Vous contrôlez les événements déclencheurs à l'aide de certains des attributs qui définissent votre file d'attente d'application. Ces informations fournissent également des exemples d'utilisation des types de déclencheur: EVERY, FIRST et DEPTH.

Vous pouvez activer et désactiver le déclenchement, et vous pouvez sélectionner le nombre ou la priorité des messages qui comptent pour un événement déclencheur. Vous trouverez une description complète de ces attributs dans Attributs des objets.

Les attributs pertinents sont les suivants:

TriggerControl
Utilisez cet attribut pour activer et désactiver le déclenchement pour une file d'attente d'application.

TriggerMsgPriority
Priorité minimale qu'un message doit avoir pour être comptabilisé dans un événement déclencheur. Si un message de priorité inférieure à *TriggerMsgPriority* arrive dans la file d'attente de l'application, le gestionnaire de files d'attente ignore le message lorsqu'il détermine s'il convient de créer un message de déclenchement. Si *TriggerMsgPriority* est défini sur zéro, tous les messages sont comptabilisés dans un événement déclencheur.

TriggerType
En plus du type de déclencheur NONE (qui désactive le déclenchement tout comme la définition de *TriggerControl* sur OFF), vous pouvez utiliser les types de déclencheur suivants pour définir la sensibilité d'une file d'attente pour déclencher des événements:

EVERY
Un événement déclencheur se produit chaque fois qu'un message arrive dans la file d'attente de l'application. Utilisez ce type de déclencheur si vous souhaitez que plusieurs instances d'une application soient démarrées.

PREMIER
Un événement déclencheur se produit uniquement lorsque le nombre de messages dans la file d'attente d'application passe de zéro à un. Utilisez ce type de déclencheur si vous souhaitez qu'un programme de service démarre lorsque le premier message arrive dans une file d'attente, continuez jusqu'à ce qu'il n'y ait plus de messages à traiter, puis arrêtez. Vous devez toujours traiter la file d'attente jusqu'à ce qu'elle soit vide. Voir aussi «Cas particulier du type de déclencheur FIRST», à la page 901.

PROFONDEUR
Un événement déclencheur se produit uniquement lorsque le nombre de messages dans la file d'attente d'application atteint la valeur de l'attribut **TriggerDepth** . Une utilisation typique de ce type de déclenchement est de démarrer un programme lorsque toutes les réponses à un ensemble de requêtes sont reçues.

Déclenchement par profondeur : Avec le déclenchement par nombre de lignes, le gestionnaire de files d'attente désactive le déclenchement (à l'aide de l'attribut *TriggerControl*) après la création d'un message de déclenchement. Votre application doit réactiver elle-même le déclenchement (à l'aide de l'appel MQSET) une fois que cela s'est produit.

L'action de désactivation du déclenchement n'étant pas sous contrôle de point de synchronisation, le déclenchement ne peut pas être réactivé par l'annulation d'une unité d'oeuvre. Si un programme annule une demande d'insertion à l'origine d'un événement déclencheur ou si le programme s'arrête de manière anormale, vous devez réactiver le déclenchement à l'aide de l'appel MQSET ou de la commande ALTER QLOCAL.

TriggerDepth
Nombre de messages dans une file d'attente qui provoquent un événement déclencheur lors de l'utilisation du déclenchement par nombre de lignes.

Les conditions qui doivent être remplies pour qu'un gestionnaire de files d'attente puisse créer un message de déclenchement sont décrites dans «Conditions d'un événement déclencheur», à la page 895.

Exemple d'utilisation du type de déclencheur EVERY

Prenons l'exemple d'une application qui génère des demandes d'assurance automobile. L'application peut envoyer des messages de demande à un certain nombre de compagnies d'assurance, en spécifiant la même file d'attente de réponses à chaque fois. Il peut définir un déclencheur de type EVERY dans cette file d'attente de réponse de sorte que chaque fois qu'une réponse arrive, la réponse peut déclencher une instance du serveur pour traiter la réponse.

Exemple d'utilisation du type de déclencheur FIRST

Prenons l'exemple d'une organisation avec un certain nombre de succursales qui transmettent chacune les détails des jours ouvrables au siège social. Ils le font tous en même temps, à la fin de la journée de travail, et au siège social, il y a une demande qui traite les détails de toutes les succursales. Le premier message à parvenir au siège social peut provoquer un événement déclencheur qui démarre cette application. Cette application poursuit le traitement jusqu'à ce qu'il n'y ait plus de messages dans sa file d'attente.

Exemple d'utilisation du type de déclencheur DEPTH

Prenons l'exemple d'une application d'agence de voyage qui crée une demande unique pour confirmer une réservation de vol, pour confirmer une réservation pour une chambre d'hôtel, pour louer une voiture et pour commander des chèques de voyage. L'application peut séparer ces éléments en quatre messages de demande, en les envoyant chacun à une destination distincte. Il peut définir un déclencheur de type DEPTH dans sa file d'attente de réponse (la longueur étant définie sur la valeur 4), de sorte qu'il ne soit redémarré que lorsque les quatre réponses sont arrivées.

Si un autre message (provenant éventuellement d'une demande différente) arrive dans la file d'attente de réponse avant la dernière des quatre réponses, l'application demandeuse est déclenchée de manière anticipée. Pour éviter cela, lorsque vous utilisez le déclenchement DEPTH pour collecter plusieurs réponses à une demande, utilisez toujours une nouvelle file d'attente de réponse pour chaque demande.

Cas particulier du type de déclencheur FIRST

Avec le type de déclencheur FIRST, s'il existe déjà un message dans la file d'attente de l'application lorsqu'un autre message arrive, le gestionnaire de files d'attente ne crée généralement pas un autre message de déclenchement.

Toutefois, l'application qui sert la file d'attente risque de ne pas ouvrir la file d'attente (par exemple, l'application risque de s'arrêter, peut-être en raison d'un problème système). Si un nom d'application incorrect a été inséré dans l'objet de définition de processus, l'application qui sert la file d'attente ne récupère aucun des messages. Dans ces situations, si un autre message arrive dans la file d'attente de

l'application, aucun serveur n'est en cours d'exécution pour traiter ce message (et tout autre message de la file d'attente).

Pour résoudre ce problème, le gestionnaire de files d'attente crée d'autres messages de déclenchement dans les cas suivants:

- Si un autre message arrive dans la file d'attente de l'application, mais uniquement si un intervalle de temps prédéfini s'est écoulé depuis que le gestionnaire de files d'attente a créé le dernier message de déclenchement pour cette file d'attente. Cet intervalle de temps est défini dans l'attribut de gestionnaire de files d'attente *TriggerInterval*. Sa valeur par défaut est 999 999 999 millisecondes.
- z/OS Sous IBM MQ for z/OS, les files d'attente d'application qui noment une file d'attente d'initialisation ouverte sont analysées régulièrement. Si *TRIGINT* millisecondes se sont écoulées depuis l'envoi du dernier message de déclenchement et que la file d'attente remplit les conditions d'un événement déclencheur et que CURDEPTH est supérieur à zéro, un message de déclenchement est généré. Ce processus est appelé déclenchement de backstop.

Tenez compte des points suivants lorsque vous déterminez une valeur pour l'intervalle de déclenchement à utiliser dans votre application:

- Si vous définissez *TriggerInterval* sur une valeur faible et qu'aucune application ne sert la file d'attente d'application, le type de déclencheur FIRST peut se comporter comme le type de déclencheur EVERY. Cela dépend de la fréquence à laquelle les messages sont placés dans la file d'attente de l'application, qui à son tour peut dépendre d'autres activités du système. En effet, si l'intervalle de déclenchement est très petit, un autre message de déclenchement est généré chaque fois qu'un message est inséré dans la file d'attente de l'application, même si le type de déclencheur est FIRST et non EVERY. (Le type de déclencheur FIRST avec un intervalle de déclenchement de zéro est équivalent au type de déclencheur EVERY.)
- z/OS Sous IBM MQ for z/OS , si vous affectez une valeur faible à *TRIGINT* et qu'il n'existe aucune application servant la file d'attente d'application de type FIRST, le déclenchement par backstop génère un message de déclenchement chaque fois que l'analyse périodique des files d'attente d'application qui noment des files d'attente d'initialisation d'ouverture a lieu.
- Si une unité de travail est annulée (voir Messages de déclenchement et unités de travail) et l'intervalle de déclenchement a été défini sur une valeur élevée (ou la valeur par défaut), un message de déclenchement est généré lorsque l'unité de travail est annulée. Toutefois, si vous avez défini l'intervalle de déclenchement sur une valeur faible ou sur zéro (entraînant le comportement du type de déclencheur FIRST comme le type de déclencheur EVERY), de nombreux messages de déclenchement peuvent être générés. Si l'unité d'oeuvre est annulée, tous les messages de déclenchement restent disponibles. Le nombre de messages de déclenchement générés dépend de l'intervalle de déclenchement. Si l'intervalle de déclenchement est défini sur zéro, le nombre maximal de messages est généré.

Conception d'une application qui utilise des files d'attente déclenchées

Vous avez vu comment configurer et contrôler le déclenchement pour vos applications. Voici quelques conseils à prendre en compte lors de la conception de votre application.

Messages de déclenchement et unités d'oeuvre

Les messages de déclenchement créés en raison d'événements de déclenchement qui ne font pas partie d'une unité de travail sont placés dans la file d'attente d'initialisation, en dehors de toute unité de travail, sans dépendance vis-à-vis d'autres messages, et peuvent être récupérés immédiatement par le moniteur de déclenchement.

Les messages de déclenchement créés en raison d'événements de déclenchement qui font partie d'une unité de travail sont mis à disposition dans la file d'attente d'initialisation lorsque l'unité de travail est résolue, que l'unité de travail soit validée ou annulée

Si le gestionnaire de files d'attente ne parvient pas à insérer un message de déclenchement dans une file d'attente d'initialisation, il est placé dans la file d'attente de rebut (message non distribué).

Remarque :

1. Le gestionnaire de files d'attente compte à la fois les messages validés et les messages non validés lorsqu'il évalue si les conditions d'un événement déclencheur existent.

   Avec le déclenchement de type FIRST ou DEPTH, les messages de déclenchement sont rendus disponibles même si l'unité d'oeuvre est annulée, de sorte qu'un message de déclenchement est toujours disponible lorsque les conditions requises sont remplies. Par exemple, considérons une demande d'insertion dans une unité de travail pour une file d'attente qui est déclenchée avec le type de déclencheur FIRST. Le gestionnaire de files d'attente crée alors un message de déclenchement. Si une autre demande d'insertion se produit à partir d'une autre unité de travail, cela ne provoque pas d'autre événement déclencheur car le nombre de messages dans la file d'attente d'application est passé de un à deux, ce qui ne répond pas aux conditions d'un événement déclencheur. Désormais, si la première unité de travail est annulée, mais que la seconde est validée, un message de déclenchement est toujours créé.

   Toutefois, cela signifie que des messages de déclenchement sont parfois créés lorsque les conditions d'un événement déclencheur ne sont pas satisfaites. Les applications qui utilisent le déclenchement doivent toujours être préparées pour gérer cette situation. Il est recommandé d'utiliser l'option d'attente avec l'appel MQGET, en définissant *WaitInterval* sur une valeur appropriée.

   Les messages de déclenchement créés sont toujours disponibles, que l'unité de travail soit annulée ou validée.
2. Pour les files d'attente partagées locales (c'est-à-dire les files d'attente partagées dans un groupe de partage de files d'attente), le gestionnaire de files d'attente compte uniquement les messages validés.

Obtention de messages à partir d'une file d'attente de déclenchement

Lorsque vous concevez des applications qui utilisent le déclenchement, sachez qu'il peut y avoir un délai entre le démarrage d'un programme par un moniteur de déclenchement et la disponibilité d'autres messages dans la file d'attente de l'application. Cela peut se produire lorsque le message qui provoque l'événement déclencheur est validé avant les autres.

Pour laisser le temps aux messages d'arriver, utilisez toujours l'option d'attente lorsque vous utilisez l'appel MQGET pour supprimer des messages d'une file d'attente pour laquelle des conditions de déclenchement sont définies. La valeur *WaitInterval* doit être suffisante pour que le délai entre l'envoi d'un message et la validation de l'appel d'insertion soit le plus long possible. Si le message arrive d'un gestionnaire de files d'attente éloignées, cette heure est affectée par:

- Nombre de messages insérés avant d'être validés
- Vitesse et disponibilité de la liaison de communication
- Tailles des messages

Pour obtenir un exemple de situation dans laquelle vous devez utiliser l'appel MQGET avec l'option d'attente, examinez le même exemple que celui utilisé lors de la description des unités de travail. Il s'agissait d'une demande d'insertion dans une unité de travail pour une file d'attente déclenchée avec le type de déclencheur FIRST. Cet événement entraîne la création d'un message de déclenchement par le gestionnaire de files d'attente. Si une autre demande d'insertion se produit à partir d'une autre unité de travail, cela ne provoque pas d'autre événement déclencheur car le nombre de messages dans la file d'attente de l'application n'est pas passé de zéro à un. Désormais, si la première unité de travail est annulée, mais que la seconde est validée, un message de déclenchement est toujours créé. Le message de déclenchement est donc créé au moment de l'annulation de la première unité d'oeuvre. S'il y a un délai important avant la validation du deuxième message, l'application déclenchée peut avoir besoin de l'attendre.

Avec le déclenchement de type DEPTH, un retard peut se produire même si tous les messages pertinents sont finalement validés. Supposons que l'attribut de file d'attente **TriggerDepth** ait la valeur 2. Lorsque deux messages arrivent dans la file d'attente, le second provoque la création d'un message de déclenchement. Toutefois, si le deuxième message est le premier à être validé, c'est à ce moment que le message de déclenchement devient disponible. Le moniteur de déclenchement démarre le programme serveur, mais le programme ne peut extraire que le deuxième message jusqu'à ce que le premier

soit validé. Par conséquent, le programme peut avoir besoin d'attendre que le premier message soit disponible.

Concevez votre application de sorte qu'elle s'arrête si aucun message n'est disponible pour être récupéré à l'expiration de votre intervalle d'attente. Si un ou plusieurs messages arrivent ultérieurement, comptez sur le fait que votre application soit redéclenchée pour les traiter. Cette méthode empêche les applications d'être inactives et d'utiliser inutilement des ressources.

Traitement de la file d'attente d'initialisation par les moniteurs de déclenchement

Pour un gestionnaire de files d'attente, un moniteur de déclenchement est similaire à toute autre application qui sert une file d'attente. Toutefois, un moniteur de déclenchement sert les files d'attente d'initialisation.

Un moniteur de déclenchement est généralement un programme à exécution continue. Lorsqu'un message de déclenchement arrive dans une file d'attente d'initialisation, le moniteur de déclenchement extrait ce message. Il utilise les informations du message pour émettre une commande permettant de démarrer l'application qui doit traiter les messages dans la file d'attente de l'application.

Le moniteur de déclenchement doit transmettre suffisamment d'informations au programme qu'il démarre pour que le programme puisse effectuer les actions appropriées dans la file d'attente d'application appropriée.

Un initiateur de canal est un exemple de type spécial de moniteur de déclenchement pour les agents MCA. Toutefois, dans ce cas, vous devez utiliser le type de déclencheur FIRST ou DEPTH.

ALW *Moniteurs de déclenchement sur les systèmes AIX, Linux, and Windows*

Cette rubrique contient des informations sur les moniteurs de déclenchement fournis sur les systèmes AIX, Linux, and Windows .

Les moniteurs de déclenchement suivants sont fournis pour l'environnement de serveur:

amqstrg0
: Il s'agit d'un exemple de moniteur de déclenchement qui fournit un sous-ensemble de la fonction fournie par **runmqtrm**. Voir «Utilisation des exemples de programme sur Multiplatforms», à la page 1086 pour plus d'informations sur amqstrg0.

runmqtrm
: La syntaxe de cette commande est **runmqtrm** [*-m QMgrName*] [*-q InitQ*], où QMgrName est le gestionnaire de files d'attente et InitQ est la file d'attente d'initialisation. La file d'attente par défaut est SYSTEM.DEFAULT.INITIATION.QUEUE sur le gestionnaire de files d'attente par défaut. Il appelle des programmes pour les messages de déclenchement appropriés. Ce moniteur de déclenchement prend en charge le type d'application par défaut.

  La chaîne de commande transmise par le moniteur de déclenchement au système d'exploitation est générée comme suit:

  1. Le fichier *ApplId* de la définition PROCESS appropriée (s'il a été créé)
  2. La structure MQTMC2 , placée entre guillemets
  3. Le fichier *EnvData* de la définition PROCESS appropriée (s'il a été créé)

  où *ApplId* est le nom du programme à exécuter tel qu'il serait entré sur la ligne de commande.

  Le paramètre transmis est la structure de caractères MQTMC2 . Une chaîne de commande est appelée avec cette chaîne, exactement comme elle est fournie, entre guillemets, afin que la commande système l'accepte comme un paramètre.

  Le moniteur de déclenchement ne recherche pas s'il existe un autre message dans la file d'attente d'initialisation tant que l'application qu'il vient de démarrer n'est pas terminée. Si l'application a beaucoup de traitement à effectuer, le moniteur de déclenchement risque de ne pas être en mesure de suivre le nombre de messages de déclenchement qui arrivent. Deux options s'offrent à vous :

  - Avoir plus de moniteurs de déclenchement en cours d'exécution

- Exécuter les applications démarrées en arrière-plan

  Si plusieurs moniteurs de déclenchement sont en cours d'exécution, vous pouvez contrôler le nombre maximal d'applications pouvant être exécutées simultanément. Si vous exécutez des applications en arrière-plan, aucune restriction n'est imposée par IBM MQ sur le nombre d'applications pouvant être exécutées.

  Linux AIX Pour exécuter l'application démarrée en arrière-plan sur AIX and Linux, placez un & à la fin du *EnvData* de la définition PROCESS.

  Pour exécuter l'application démarrée en arrière-plan sur les systèmes Windows , dans la zone *ApplId* , préfixez le nom de votre application avec une commande START. Par exemple :

  ```
  START ?B AMQSECHA
  ```

  Remarque : Windows Lorsqu'un chemin Windows comporte des espaces comme partie du nom de chemin, ils doivent être placés entre guillemets (") pour s'assurer qu'il est traité comme un seul argument. Par exemple, "`C:\Program Files\Application Directory\Application.exe`".

  Voici un exemple de chaîne APPLICID dans laquelle le nom de fichier inclut des espaces dans le chemin d'accès:

  ```
  START "" /B "C:\Program Files\Application Directory\Application.exe"
  ```

  La syntaxe de la commande Windows START dans l'exemple inclut une chaîne vide entre guillemets. START indique que le premier argument entre guillemets sera traité comme le titre de la nouvelle commande. Pour vous assurer que Windows ne prend pas le chemin d'application pour un argument'title', ajoutez une chaîne de titre entre guillemets à la commande avant le nom de l'application.

Les moniteurs de déclenchement suivants sont fournis pour le client IBM MQ :

runmqtmc
Il est identique à runmqtrm, sauf qu'il est lié aux bibliothèques IBM MQ MQI client .

ALW *Moniteur de déclenchement pour CICS*

Le moniteur de déclenchement amqltmc0 est fourni pour CICS. Il fonctionne de la même manière que le moniteur de déclenchement standard, runmqtrm, mais vous l'exécutez de manière différente et il déclenche des transactions CICS .

Cette rubrique s'applique uniquement aux systèmes Windows, AIX and Linux x86-64 .

Le moniteur de déclenchement est fourni en tant que programme CICS ; définissez-le avec un nom de transaction à 4 caractères. Entrez le nom à 4 caractères pour démarrer le moniteur de déclenchement. Il utilise le gestionnaire de files d'attente par défaut (comme indiqué dans le fichier qm.ini ou, sous IBM MQ for Windows, le registre) et SYSTEM.CICS.INITIATION.QUEUE.

Si vous souhaitez utiliser un autre gestionnaire de files d'attente ou une autre file d'attente, générez la structure MQTMC2 du moniteur de déclenchement: vous devez alors écrire un programme à l'aide de l'appel EXEC CICS START, car la structure est trop longue pour être ajoutée en tant que paramètre. Transmettez ensuite la structure MQTMC2 en tant que données à la demande START pour le moniteur de déclenchement.

Lorsque vous utilisez la structure MQTMC2 , vous devez fournir uniquement les paramètres *StrucId*, *Version*, *QName*et **QMgrName** au moniteur de déclenchement car il ne fait référence à aucune autre zone.

Les messages sont lus à partir de la file d'attente d'initialisation et utilisés pour démarrer les transactions CICS à l'aide de la commande EXEC CICS START, en supposant que le paramètre APPL_TYPE du message de déclenchement est MQAT_CICS. La lecture des messages de la file d'attente d'initialisation est effectuée sous le contrôle du point de synchronisation CICS .

Des messages sont générés lorsque le moniteur démarre et s'arrête, et lorsqu'une erreur se produit. Ces messages sont envoyés à la file d'attente de données transitoires CSMT.

Tableau 129. Versions disponibles du moniteur de déclenchement.

Tableau à deux colonnes. la première colonne répertorie les versions disponibles du moniteur de déclenchement et la deuxième colonne indique les plateformes pour lesquelles chaque version est utilisée.

Version	Utilisation
amqltmc0	TXSeries Pour : • AIX AIX • Linux Systèmes Linux x86-64
amqltmc4	Windows TXSeries pour Windows 5.1
amqltmcc	Version liée au client du moniteur de déclenchement CICS
V 9.4.0 V 9.4.0 amqltmc064	TXSeries 64 bits pour les systèmes Linux x86-64
V 9.4.0 V 9.4.0 amqltmcc64	Version client de amqltmc064

Si vous avez besoin d'un moniteur de déclenchement pour d'autres environnements, écrivez un programme qui peut traiter les messages de déclenchement que le gestionnaire de files d'attente place dans les files d'attente d'initialisation. Un tel programme doit effectuer les actions suivantes:

1. Utilisez l'appel MQGET pour attendre qu'un message arrive dans la file d'attente d'initialisation.
2. Examinez les zones de la structure MQTM du message de déclenchement pour trouver le nom de l'application à démarrer et l'environnement dans lequel elle s'exécute.
3. Emettez une commande de démarrage spécifique à l'environnement.

   z/OS Par exemple, sur un lot z/OS , soumettez un travail au lecteur interne.
4. Convertissez la structure MQTM en structure MQTMC2 si nécessaire.
5. Transmettez la structure MQTMC2 ou MQTM à l'application démarrée. Il peut contenir des données utilisateur.
6. Associez à votre file d'attente d'application l'application qui doit servir cette file d'attente. Pour ce faire, nommez l'objet de définition de processus (s'il est créé) dans l'attribut **ProcessName** de la file d'attente. Pour nommer l'objet de définition de processus, vous pouvez utiliser la commande **DEFINE QLOCAL** ou **ALTER QLOCAL** .

   IBM i Sous IBM i, vous pouvez également utiliser CRTMQMQ ou CHGMQMQ.

Pour plus d'informations sur l'interface du moniteur de déclenchement, voir MQTMC2.

IBM i *Moniteurs de déclenchement sous IBM i*

Sous IBM i, à la place de la commande de contrôle **runmqtrm** , utilisez la IBM MQ for IBM i commande CL **STRMQMTRM**.

Utilisez la commande STRMQMTRM comme suit:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Les détails sont comme pour runmqtrm.

Les exemples de programmes suivants sont également fournis, que vous pouvez utiliser comme modèles pour écrire vos propres moniteurs de déclenchement:

AMQSTRG4
Il s'agit d'un moniteur de déclenchement qui soumet un travail IBM i pour le processus à démarrer, mais cela signifie qu'un traitement supplémentaire est associé à chaque message de déclenchement.

AMQSERV4
Il s'agit d'un serveur de déclenchement. Pour chaque message de déclenchement, ce serveur exécute la commande du processus dans son propre travail et peut appeler des transactions CICS .

Le moniteur de déclenchement et le serveur de déclenchement transmettent une structure MQTMC2 aux programmes qu'ils démarrent. Pour une description de cette structure, voir MQTMC2. Ces deux exemples sont fournis dans des formulaires source et exécutables.

Etant donné que ces moniteurs de déclenchement ne peuvent appeler que des programmes IBM i natifs, ils ne peuvent pas déclencher directement les programmes Java , car les classes Java se trouvent dans le système IFS. Toutefois, les programmes Java peuvent être déclenchés indirectement en déclenchant un programme CL qui appelle ensuite le programme Java et passe par la structure TMC2 . La taille minimale de la structure TMC2 est de 732 octets.

Voici la source d'un exemple d'interpréteur de commandes:

```
PGM PARM(&TMC2)
  DCL &TMC2 *CHAR LEN(800)
    ADDENVVAR ENVVAR(TM) VALUE(&TMC2)
    QSH  CMD('java_pgmname $TM')
    RMVENVVAR ENVVAR(TM)
ENDPGM
```

Le programme de moniteur de déclenchement suivant est fourni pour IBM MQ MQI client: RUNMQTMC

Appelez RUNMQTMC comme suit:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QMgrName '-q' InitQ)
```

Propriétés des messages de déclenchement

Les rubriques suivantes décrivent d'autres propriétés des messages de déclenchement.

- «Persistance et priorité des messages de déclenchement», à la page 907
- «Messages de redémarrage et de déclenchement du gestionnaire de files d'attente», à la page 908
- «Messages de déclenchement et modifications des attributs d'objet», à la page 908
- «Format des messages de déclenchement», à la page 908

Persistance et priorité des messages de déclenchement

Les messages de déclenchement ne sont pas persistants car ils ne sont pas requis.

Cependant, les conditions de génération des événements déclencheurs sont conservées, de sorte que les messages de déclenchement sont générés chaque fois que ces conditions sont remplies. Si un message de déclenchement est perdu, la persistance du message d'application dans la file d'attente d'application garantit que le gestionnaire de files d'attente génère un message de déclenchement dès que toutes les conditions sont remplies.

Si une unité de travail est annulée, les messages de déclenchement qu'elle génère sont toujours distribués.

Les messages de déclenchement prennent la priorité par défaut de la file d'attente d'initialisation.

Messages de redémarrage et de déclenchement du gestionnaire de files d'attente

Après le redémarrage d'un gestionnaire de files d'attente, lorsqu'une file d'attente d'initialisation est ensuite ouverte en entrée, un message de déclenchement peut être inséré dans cette file d'attente d'initialisation si une file d'attente d'application associée contient des messages et est définie pour le déclenchement.

Messages de déclenchement et modifications des attributs d'objet

Les messages de déclenchement sont créés en fonction des valeurs des attributs de déclencheur en vigueur au moment de l'événement déclencheur.

Si le message de déclenchement n'est pas mis à la disposition d'un moniteur de déclenchement avant une date ultérieure (car le message à l'origine de sa génération a été placé dans une unité d'oeuvre), les modifications apportées aux attributs de déclencheur entre-temps n'ont aucun effet sur le message de déclenchement. En particulier, la désactivation du déclenchement n'empêche pas la mise à disposition d'un message de déclenchement une fois qu'il a été créé. De plus, il se peut que la file d'attente d'application n'existe plus au moment où le message de déclenchement est rendu disponible.

Format des messages de déclenchement

Le format d'un message de déclenchement est défini par la structure MQTM.

Il comporte les zones suivantes, que le gestionnaire de files d'attente remplit lorsqu'il crée le message de déclenchement, à l'aide des informations des définitions d'objet de la file d'attente d'application et du processus associé à cette file d'attente:

StrucId
Identificateur de structure.

Version
Version de la structure.

QName
Nom de la file d'attente d'application dans laquelle l'événement déclencheur s'est produit. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **QName** de la file d'attente d'application.

ProcessName
Nom de l'objet de définition de processus associé à la file d'attente d'application. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **ProcessName** de la file d'attente d'application.

TriggerData
Zone à format libre à utiliser par le moniteur de déclenchement. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **TriggerData** de la file d'attente d'application. Sous IBM MQ for Multiplatforms, cette zone peut être utilisée pour spécifier le nom du canal à déclencher.

ApplType
Type de l'application que le moniteur de déclenchement doit démarrer. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **ApplType** de l'objet de définition de processus identifié dans *ProcessName*.

ApplId
Chaîne de caractères qui identifie l'application que le moniteur de déclenchement doit démarrer. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **ApplId** de l'objet de définition de processus identifié dans *ProcessName*.

Lorsque vous utilisez le moniteur de déclenchement CKTI, fourni par CICS, l'attribut **ApplId** de l'objet de définition de processus est un identificateur de transaction CICS .

z/OS Lorsque vous utilisez CSQQTRMN fourni par IBM MQ for z/OS, l'attribut **ApplId** de l'objet de définition de processus est un identificateur de transaction IMS .

EnvData
Zone alphanumérique contenant les données relatives à l'environnement à utiliser par le moniteur de déclenchement. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **EnvData** de l'objet de définition de processus identifié dans *ProcessName*. Le CICSmoniteur de déclenchement fourni par (CKTI) ou le IBM MQ for z/OSmoniteur de déclenchement fourni par (CSQQTRMN) n'utilise pas cette zone, mais d'autres moniteurs de déclenchement peuvent choisir de l'utiliser.

UserData
Zone alphanumérique contenant les données utilisateur à utiliser par le moniteur de déclenchement. Lorsque le gestionnaire de files d'attente crée un message de déclenchement, il remplit cette zone à l'aide de l'attribut **UserData** de l'objet de définition de processus identifié dans *ProcessName*. Cette zone peut être utilisée pour indiquer le nom du canal à déclencher.

Il existe une description complète de la structure de message de déclencheur dans MQTM.

Lorsque le déclenchement ne fonctionne pas

Un programme n'est pas déclenché si le moniteur de déclenchement ne peut pas démarrer le programme ou si le gestionnaire de files d'attente ne peut pas distribuer le message de déclenchement. Par exemple, l'ID application de l'objet processus doit indiquer que le programme doit être démarré en arrière-plan ; sinon, le moniteur de déclenchement ne peut pas démarrer le programme.

Si un message de déclenchement est créé mais ne peut pas être inséré dans la file d'attente d'initialisation (par exemple, parce que la file d'attente est pleine ou que la longueur du message de déclenchement est supérieure à la longueur maximale de message spécifiée pour la file d'attente d'initialisation), le message de déclenchement est inséré dans la file d'attente de rebut (message non distribué).

Si l'opération d'insertion dans la file d'attente de rebut ne peut pas aboutir, le message de déclenchement est supprimé et un message d'avertissement est envoyé z/OS à la console z/OS ou à l'opérateur système ou placé dans le journal des erreurs.

L'insertion du message de déclenchement dans la file d'attente de rebut peut générer un message de déclenchement pour cette file d'attente. Ce second message de déclenchement est supprimé s'il ajoute un message à la file d'attente des messages non livrés.

Si le programme est déclenché avec succès mais qu'il s'arrête de façon anormale avant de recevoir le message de la file d'attente, utilisez un utilitaire de trace (par exemple, CICS AUXTRACE si le programme s'exécute sous CICS) pour trouver la cause de l'échec.

Utilisation de l'interface MQI et des clusters

Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

Utilisez les liens suivants pour en savoir plus sur les options disponibles sur les appels et les codes retour à utiliser avec les clusters:

- «MQOPEN et clusters», à la page 910
- «MQPUT, MQPUT1 et clusters», à la page 911
- «MQINQ et clusters», à la page 912
- «MQSET et clusters», à la page 913
- «Codes retour», à la page 913

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 739
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

«Ouverture et fermeture d'objets», à la page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

«Insertion de messages dans une file d'attente», à la page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Obtention de messages à partir d'une file d'attente», à la page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

«Validation et annulation d'unités de travail», à la page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

«Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

«Using and writing applications on IBM MQ for z/OS», à la page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

«IMS and IMS bridge applications on IBM MQ for z/OS», à la page 72
This information helps you to write IMS applications using IBM MQ.

MQOPEN et clusters

La file d'attente dans laquelle un message est inséré ou lu lorsqu'une file d'attente de cluster est ouverte dépend de l'appel MQOPEN .

Sélection de la file d'attente cible

Si vous n'indiquez pas de nom de gestionnaire de files d'attente dans le descripteur d'objet, MQOD, le gestionnaire de files d'attente sélectionne le gestionnaire de files d'attente auquel envoyer le message. Si vous indiquez un nom de gestionnaire de files d'attente dans le descripteur d'objet, les messages sont toujours envoyés au gestionnaire de files d'attente que vous avez sélectionné.

Si le gestionnaire de files d'attente sélectionne le gestionnaire de files d'attente cible, la sélection dépend des options de liaison, MQOO_BIND_* et de l'existence d'une file d'attente locale. S'il existe une instance locale de la file d'attente, elle est toujours ouverte de préférence à une instance distante, sauf si l'attribut CLWLUSEQ est défini sur ANY. Sinon, la sélection dépend des options de liaison. MQOO_BIND_ON_OPEN ou MQOO_BIND_ON_GROUP doit être spécifié lors de l'utilisation de groupes de messages avec des clusters pour garantir que tous les messages du groupe sont traités à la même destination.

Si le gestionnaire de files d'attente sélectionne le gestionnaire de files d'attente cible, il le fait de manière circulaire, à l'aide de l'algorithme de gestion de la charge de travail ; voir Equilibrage de la charge de travail dans les clusters.

Lorsque l'algorithme d'équilibrage de charge est utilisé, il dépend de la manière dont la file d'attente de cluster est ouverte:

- MQOO_BIND_ON_OPEN -l'algorithme est utilisé une fois au moment de l'ouverture de la file d'attente par l'application.
- MQOO_BIND_NOT_FIXED -l'algorithme est utilisé pour chaque insertion de message dans la file d'attente.
- MQOO_BIND_ON_GROUP -l'algorithme est utilisé une fois au début de chaque groupe de messages.

MQOO_BIND_ON_OPEN
L'option MQOO_BIND_ON_OPEN de l'appel MQOPEN indique que le gestionnaire de files d'attente cible doit être corrigé. Utilisez l'option MQOO_BIND_ON_OPEN s'il existe plusieurs instances de la même file d'attente dans un cluster. Tous les messages insérés dans la file d'attente spécifiant le descripteur d'objet renvoyé par l'appel MQOPEN sont dirigés vers le même gestionnaire de files d'attente.

- Utilisez l'option MQOO_BIND_ON_OPEN si les messages ont des affinités. Par exemple, si un lot de messages doit tous être traité par le même gestionnaire de files d'attente, indiquez MQOO_BIND_ON_OPEN lorsque vous ouvrez la file d'attente. IBM MQ corrige le gestionnaire de files d'attente et la route à prendre par tous les messages insérés dans cette file d'attente.
- Si l'option MQOO_BIND_ON_OPEN est spécifiée, la file d'attente doit être rouverte pour qu'une nouvelle instance de la file d'attente soit sélectionnée.

MQOO_BIND_NOT_FIXED
L'option MQOO_BIND_NOT_FIXED de l'appel MQOPEN indique que le gestionnaire de files d'attente cible n'est pas fixe. Les messages écrits dans la file d'attente spécifiant l'identificateur d'objet renvoyé par l'appel MQOPEN sont acheminés vers un gestionnaire de files d'attente à MQPUT , message par message. Utilisez l'option MQOO_BIND_NOT_FIXED si vous ne souhaitez pas forcer l'écriture de tous vos messages sur la même destination.

- Ne spécifiez pas MQOO_BIND_NOT_FIXED et MQMF_SEGMENTATION_ALLOWED en même temps. Dans ce cas, les segments de votre message peuvent être distribués à différents gestionnaires de files d'attente, dispersés dans le cluster.

MQOO_BIND_ON_GROUP
Permet à une application de demander qu'un groupe de messages soit alloué à la même instance de destination. Cette option est valide uniquement pour les files d'attente et n'affecte que les files d'attente de cluster. Si cette option est spécifiée pour une file d'attente qui n'est pas une file d'attente de cluster, elle est ignorée.

- Les groupes ne sont routés vers une destination unique que lorsque MQPMO_LOGICAL_ORDER est spécifié dans MQPUT. Lorsque MQOO_BIND_ON_GROUP est spécifié, mais qu'un message ne fait pas partie d'un groupe logique, le comportement BIND_NOT_FIXED est utilisé à la place.

MQOO_BIND_AS_Q_DEF
Si vous ne spécifiez pas MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED ou MQOO_BIND_ON_GROUP, l'option par défaut est MQOO_BIND_AS_Q_DEF. L'utilisation de MQOO_BIND_AS_Q_DEF entraîne l'utilisation de la liaison utilisée pour l'identificateur de file d'attente à partir de l'attribut de file d'attente DefBind .

Pertinence des options MQOPEN

Les MQOPEN options MQOO_BROWSE, MQOO_INPUT_*ou MQOO_SET requièrent une instance locale de la file d'attente de cluster pour que MQOPEN aboutisse.

Les options MQOPEN MQOO_OUTPUT, MQOO_BIND_*ou MQOO_INQUIRE ne nécessitent pas d'instance locale de la file d'attente de cluster pour réussir.

Nom de gestionnaire de files d'attente résolu

Lorsqu'un nom de gestionnaire de files d'attente est résolu à MQOPEN , le nom résolu est renvoyé à l'application. Si l'application tente d'utiliser ce nom lors d'un appel MQOPEN ultérieur, il se peut qu'elle ne soit pas autorisée à accéder au nom.

MQPUT, MQPUT1 et clusters

Si MQOO_BIND_NOT_FIXED est spécifié sur un MQOPEN , les routines de gestion de charge de travail choisissent la destination MQPUT ou MQPUT1 sélectionnée.

Si MQOO_BIND_NOT_FIXED est spécifié sur un appel MQOPEN , chaque appel MQPUT ultérieur appelle la routine de gestion de la charge de travail pour déterminer à quel gestionnaire de files d'attente le message doit être envoyé. La destination et la route à prendre sont sélectionnées message par message. La destination et la route peuvent changer après l'insertion du message si les conditions du réseau changent. L'appel MQPUT1 fonctionne toujours comme si MQOO_BIND_NOT_FIXED était actif, c'est-à-dire qu'il appelle toujours la routine de gestion de la charge de travail.

Lorsque la routine de gestion de la charge de travail a sélectionné un gestionnaire de files d'attente, le gestionnaire de files d'attente local termine l'opération d'insertion. Le message peut être placé dans différentes files d'attente:

1. Si la destination est l'instance locale de la file d'attente, le message est placé dans la file d'attente locale.
2. Si la destination est un gestionnaire de files d'attente dans un cluster, le message est placé dans une file d'attente de transmission de cluster.
3. Si la destination est un gestionnaire de files d'attente en dehors d'un cluster, le message est placé dans une file d'attente de transmission portant le même nom que le gestionnaire de files d'attente cible.

Si MQOO_BIND_ON_OPEN est spécifié dans l'appel MQOPEN , les appels MQPUT n'appellent pas la routine de gestion de la charge de travail car la destination et la route ont déjà été sélectionnées.

MQINQ et clusters

L'interrogation de la file d'attente de cluster dépend des options que vous combinez à MQOO_INQUIRE.

Avant de pouvoir consulter une file d'attente, ouvrez-la à l'aide de l'appel MQOPEN et spécifiez MQOO_INQUIRE.

Pour interroger une file d'attente de cluster, utilisez l'appel MQOPEN et combinez d'autres options avec MQOO_INQUIRE. Les attributs qui peuvent être questionnés dépendent de la présence ou non d'une instance locale de la file d'attente de cluster et du mode d'ouverture de la file d'attente:

- La combinaison de MQOO_BROWSE, MQOO_INPUT_*ou MQOO_SET avec MQOO_INQUIRE requiert une instance locale de la file d'attente de cluster pour que l'ouverture aboutisse. Dans ce cas, vous pouvez vous renseigner sur tous les attributs valides pour les files d'attente locales.
- En combinant MQOO_OUTPUT avec MQOO_INQUIREet en ne spécifiant aucune des options précédentes, l'instance ouverte est l'une des suivantes:
  - L'instance sur le gestionnaire de files d'attente local, s'il en existe une. Dans ce cas, vous pouvez vous renseigner sur tous les attributs valides pour les files d'attente locales.
  - Une instance ailleurs dans le cluster, s'il n'existe pas d'instance de gestionnaire de files d'attente local. Dans ce cas, seuls les attributs suivants peuvent être utilisés. L'attribut QType a la valeur MQQT_CLUSTER dans ce cas.
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - QName
    - QType

Pour consulter l'attribut DefBind d'une file d'attente de cluster, utilisez l'appel MQINQ avec le sélecteur MQIA_DEF_BIND. La valeur renvoyée est MQBND_BIND_ON_OPEN , MQBND_BIND_NOT_FIXEDou MQBND_BIND_ON_GROUP. MQBND_BIND_ON_OPEN ou MQBND_BIND_ON_GROUP doit être spécifié lors de l'utilisation de groupes avec des clusters.

Pour consulter les attributs CLUSTER et CLUSNL de l'instance locale d'une file d'attente, utilisez l'appel MQINQ avec le sélecteur MQCA_CLUSTER_NAME ou le sélecteur MQCA_CLUSTER_NAMELIST.

Remarque : Si vous ouvrez une file d'attente de cluster sans corriger la file d'attente à laquelle MQOPEN est lié, les appels MQINQ successifs peuvent s'interroger sur différentes instances de la file d'attente de cluster.

Concepts associés

«Option MQOPEN pour la file d'attente de cluster», à la page 767
La liaison utilisée pour l'identificateur de file d'attente est extraite de l'attribut de file d'attente **DefBind** , qui peut prendre la valeur MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXEDou MQBND_BIND_ON_GROUP.

MQSET et clusters

L'option MQOPEN option MQOO_SET requiert l'existence d'une instance locale d'une file d'attente de cluster pour que MQSET aboutisse.

Vous ne pouvez pas utiliser l'appel MQSET pour définir les attributs d'une file d'attente ailleurs dans le cluster.

Vous pouvez ouvrir un alias local ou une file d'attente éloignée définie avec l'attribut de cluster et utiliser l'appel MQSET . Vous pouvez définir les attributs de l'alias local ou de la file d'attente éloignée. Peu importe que la file d'attente cible soit une file d'attente de cluster définie sur un autre gestionnaire de files d'attente.

Codes retour

Codes retour spécifiques aux clusters

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Un appel MQOPEN, MQPUTou MQPUT1 est émis pour ouvrir une file d'attente de cluster ou y placer un message. L'exit de charge de travail de cluster, défini par l'attribut ClusterWorkloadExit d'un gestionnaire de files d'attente, échoue de manière inattendue ou ne répond pas dans le temps.

z/OS Un message est consigné dans le journal système sur IBM MQ for z/OS pour fournir plus d'informations sur cette erreur.

Les appels MQOPEN, MQPUTet MQPUT1 ultérieurs de ce descripteur de file d'attente sont traités comme si l'attribut ClusterWorkloadExit était vide.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

z/OS Sous z/OS, l'exit de charge de travail de cluster ne peut pas être chargé.

Un message est consigné dans le journal système et le traitement continue comme si l'attribut ClusterWorkloadExit était vide.

Multi Sous Multiplateformes, un appel MQCONN ou MQCONNX est émis pour la connexion à un gestionnaire de files d'attente. L'appel échoue car l'exit de charge de travail de cluster, défini par l'attribut ClusterWorkloadExit du gestionnaire de files d'attente, ne peut pas être chargé.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Un appel MQOPEN avec les options MQOO_OUTPUT et MQOO_BIND_ON_OPEN en vigueur est émis pour une file d'attente de cluster. Toutes les instances de la file d'attente dans le cluster sont actuellement interdites d'insertion en ayant l'attribut InhibitPut défini sur MQQA_PUT_INHIBITED. Etant donné qu'aucune instance de file d'attente n'est disponible pour recevoir des messages, l'appel MQOPEN échoue.

Ce code anomalie se produit uniquement lorsque les deux instructions suivantes sont vraies:

- Il n'existe aucune instance locale de la file d'attente. S'il existe une instance locale, l'appel MQOPEN aboutit, même si l'instance locale est désactivée.
- Il n'existe pas d'exit de charge de travail de cluster pour la file d'attente ou il existe un exit de charge de travail de cluster mais il ne choisit pas d'instance de file d'attente. (Si l'exit de charge de travail du cluster choisit une instance de file d'attente, l'appel MQOPEN aboutit, même si cette instance est interdite.)

Si l'option MQOO_BIND_NOT_FIXED est spécifiée dans l'appel MQOPEN , l'appel peut aboutir même si toutes les files d'attente du cluster sont interdites d'insertion. Toutefois, un appel MQPUT suivant peut échouer si toutes les files d'attente sont toujours interdites d'insertion au moment de cet appel.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Un appel MQOPEN, MQPUTou MQPUT1 est émis pour ouvrir une file d'attente de cluster ou y placer un message. La définition de file d'attente ne peut pas être résolue correctement car une réponse est requise du gestionnaire de files d'attente du référentiel complet, mais aucune n'est disponible.

2. Un appel MQOPEN, MQPUT, MQPUT1 ou MQSUB est émis pour un objet de rubrique spécifiant PUBSCOPE (ALL) ou SUBSCOPE (ALL). La définition de rubrique de cluster ne peut pas être résolue correctement car une réponse est requise du gestionnaire de files d'attente de référentiel complet, mais aucune n'est disponible.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Un appel MQOPEN, MQPUTou MQPUT1 est émis pour une file d'attente de cluster. Une erreur s'est produite lors de la tentative d'utilisation d'une ressource requise pour la mise en cluster.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Un appel MQPUT ou MQPUT1 est émis pour insérer un message dans une file d'attente de cluster. Au moment de l'appel, il n'y a plus aucune instance de la file d'attente dans le cluster. Le MQPUT échoue et le message n'est pas envoyé.

L'erreur peut se produire si MQOO_BIND_NOT_FIXED est spécifié dans l'appel MQOPEN qui ouvre la file d'attente ou si MQPUT1 est utilisé pour insérer le message.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Un appel MQOPEN, MQPUTou MQPUT1 est émis pour ouvrir ou insérer un message dans une file d'attente de cluster. L'exit de charge de travail du cluster rejette l'appel.

z/OS Using and writing applications on IBM MQ for z/OS

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

This information explains the IBM MQ facilities available to programs running in each of the supported environments. In addition,

- For information about using the IBM MQ-CICS bridge, see Using IBM MQ with CICS .
- For information about using IMS and the IMS bridge, see “IMS and IMS bridge applications on IBM MQ for z/OS” on page 72.

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

- “Environment-dependent IBM MQ for z/OS functions” on page 915
- “Debugging facilities, syncpoint support, and recovery support” on page 915
- “The IBM MQ for z/OS interface with the application environment” on page 916
- “Writing z/OS UNIX System Services applications” on page 918
- “Application programming with shared queues” on page 921

Related concepts

“Présentation de l'interface de file d'attente de messages” on page 739
Découvrez les composants MQI (Message Queue Interface).

“Connexion et déconnexion d'un gestionnaire de files d'attente” on page 753
Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

“Ouverture et fermeture d'objets” on page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

“Insertion de messages dans une file d'attente” on page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

“Obtention de messages à partir d'une file d'attente” on page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

“Interrogation et définition des attributs d'objet” on page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

“Validation et annulation d'unités de travail” on page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

“Démarrage des applications IBM MQ à l'aide de déclencheurs” on page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

“Utilisation de l'interface MQI et des clusters” on page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

“IMS and IMS bridge applications on IBM MQ for z/OS” on page 72
This information helps you to write IMS applications using IBM MQ.

z/OS *Environment-dependent IBM MQ for z/OS functions*

Use this information when considering IBM MQ for z/OS functions.

The main differences to be considered between IBM MQ functions in the environments in which IBM MQ for z/OS runs are:

- IBM MQ for z/OS supplies the following trigger monitors:
  - CKTI for use in the CICS environment
  - CSQQTRMN for use in the IMS environment

  You must write your own module to start applications in other environments.
- Syncpointing using two-phase commit is supported in the CICS and IMS environments. It is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). Single-phase commit is supported in the z/OS environment by IBM MQ itself.
- For the batch and IMS environments, the MQI provides calls to connect programs to, and to disconnect them from, a queue manager. Programs can connect to more than one queue manager.
- A CICS system can connect to only one queue manager. This can be made to happen when CICS is initiated if the subsystem name is defined in the CICS system startup job. The MQI connect and disconnect calls are tolerated, but have no effect, in the CICS environment.
- The API-crossing exit allows a program to intervene in the processing of all MQI calls. This exit is available in the CICS environment only.
- In CICS on multiprocessor systems, some performance advantage is gained because MQI calls can be executed under multiple z/OS TCBs. For more information, see the Planification sur z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

These features are summarized in Table 130 on page 915.

Table 130. z/OS environmental features

	CICS	**IMS**	**Batch/TSO**
Trigger monitor supplied	Yes	Yes	No
Two-phase commit	Yes	Yes	Yes
Single-phase commit	Yes	No	Yes
Connect/disconnect MQI calls	Tolerated	Yes	Yes
API-crossing exit	Yes	No	No

Note: Two-phase commit is supported in the Batch/TSO environment using RRS.

z/OS *Debugging facilities, syncpoint support, and recovery support*

Use this information to learn about program debugging facilities, syncpoint support, and recovery support.

Program debugging facilities

IBM MQ for z/OS provides a trace facility that you can use to debug your programs in all environments.

Additionally, in the CICS environment you can use:

- The CICS Execution Diagnostic Facility (CEDF)
- The CICS Trace Control Transaction (CETR)
- The IBM MQ for z/OS API-crossing exit

On the z/OS platform, you can use any available interactive debugging tool that is supported by the programming language that you are using.

Syncpoint support

Synchronizing the start and end of units of work is necessary in a transaction processing environment so that transaction processing can be used safely.

This is fully supported by IBM MQ for z/OS in the CICS and IMS environments. Full support means cooperation between resource managers so that units of work can be committed or backed out in unison, under control of CICS or IMS. Examples of resource managers are Db2, CICS File Control, IMS, and IBM MQ for z/OS.

z/OS batch applications can use IBM MQ for z/OS calls to give a single-phase commit facility. This means that an application-defined set of queue operations can be committed, or backed out, without reference to other resource managers.

Two-phase commit is also supported in the z/OS batch environment using transaction management and recoverable resource manager services (RRS). For further information see Syncpoints in z/OS batch applications.

Recovery support

If the connection between a queue manager and a CICS or IMS system is broken during a transaction, some units of work might not be backed out successfully.

However, these units of work are resolved by the queue manager (under the control of the syncpoint manager) when its connection with the CICS or IMS system is reestablished.

z/OS *The IBM MQ for z/OS interface with the application environment*

To allow applications running in different environments to send and receive messages through a message queuing network, IBM MQ for z/OS provides an *adapter* for each of the environments it supports.

These adapters are the interface between application programs and IBM MQ for z/OS subsystems. They allow the programs to use the MQI.

z/OS *The batch adapter*

Use this information to learn about the batch adapter and the commit protocol it supports.

The *batch adapter* provides access to IBM MQ for z/OS resources for programs running in:

- Task (TCB) mode
- Problem or supervisor state
- Primary address space control mode

The programs must not be in cross-memory mode.

Connections between application programs and IBM MQ for z/OS are at the task level. The adapter provides a single connection thread from an application task control block (TCB) to IBM MQ for z/OS.

The adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ for z/OS ; it does not support multiphase-commit protocols.

z/OS *The RRS batch adapter*

Use this information to learn about the RRS batch adapter and the two RRS batch adapters provided by IBM MQ.

The transaction management and recoverable resource manager services (RRS) adapter:

- Uses z/OS RRS for commit control.
- Supports simultaneous connections to multiple IBM MQ subsystems running on a single z/OS instance from a single task.
- Provides z/OS-wide coordinated commitment control (using z/OS RRS) for recoverable resources accessed through z/OS RRS-compliant recoverable managers for:
  - Applications that connect to IBM MQ using the RRS batch adapter.
  - Db2-stored procedures executing in a Db2-stored procedures address space that is managed by a workload manager (WLM) on z/OS.
- Supports the ability to switch an IBM MQ batch thread between TCBs.

IBM MQ for z/OS provides two RRS batch adapters:

CSQBRSTB
: This adapter requires you to change any MQCMIT statement to SRRCMIT and any MQBACK statement to SRRBACK in your IBM MQ application. (If you code MQCMIT or MQBACK in an application linked with CSQBRSTB, you receive MQRC_ENVIRONMENT_ERROR.)

CSQBRRSI
: This adapter allows your IBM MQ application to use either MQCMIT and MQBACK or SRRCMIT and SRRBACK.

Note: CSQBRSTB and CSQBRRSI are shipped with linkage attributes AMODE(31) RMODE(ANY). If your application loads either stub below the 16 MB line, first relink the stub with RMODE(24).

Migration

You can migrate existing Batch/TSO IBM MQ applications to use RRS coordination with few or no changes.

If you link-edit your IBM MQ application with the CSQBRRSI adapter, MQCMIT and MQBACK syncpoint your unit of work across IBM MQ and all other RRS-enabled resource managers. If you link-edit your IBM MQ application with the CSQBRSTB adapter, change MQCMIT to SRRCMIT and MQBACK to SRRBACK. The latter approach is preferable; it clearly indicates that the syncpoint is not restricted to IBM MQ resources only.

z/OS *The IMS adapter*

If you are using the IMS adapter from an IBM MQ for z/OS system, ensure that IMS can obtain sufficient storage to accommodate messages up to 100 MB long.

Note to users

The *IMS adapter* provides access to IBM MQ for z/OS resources for:

- Online message processing programs (MPPs)
- Interactive fast path programs (IFPs)
- Batch message processing programs (BMPs)

To use these resources, the programs must be running in task (TCB) mode and problem state; they must not be in cross-memory mode or access-register mode.

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ. The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ for z/OS, with IMS acting as the syncpoint coordinator.

The adapter also provides a trigger monitor program that can start programs automatically when certain trigger conditions on a queue are met. For more information, see “Démarrage des applications IBM MQ à l'aide de déclencheurs” on page 888.

If you are writing batch DL/I programs, follow the guidance given in this topic for z/OS batch programs.

z/OS *Writing z/OS UNIX System Services applications*

The batch adapter supports queue manager connections from batch and TSO address spaces.

For a batch address space, the adapter supports connections from multiple TCBs within that address space as follows:

- Each TCB can connect to multiple queue managers using the MQCONN or MQCONNX call (but a TCB can only have one instance of a connection to a particular queue manager at any one time).
- Multiple TCBs can connect to the same queue manager (but the queue manager handle returned on any MQCONN or MQCONNX call is bound to the issuing TCB and cannot be used by any other TCB).

z/OS UNIX System Services supports two types of pthread_create call:

1. Heavyweight threads, run one for each TCB, that are ATTACHed and DETACHed at thread start and end by z/OS.
2. Medium-weight threads, run one for each TCB, but the TCB can be one of a pool of long-running TCBs. The application must perform all necessary application cleanup, because, if it is connected to a server, the default thread termination that might be provided by the server at task (TCB) termination, is **not** always driven.

Lightweight threads are not supported. (If an application creates permanent threads that dispatch their own work requests, the ***application*** is responsible for cleaning up any resources before starting the next work request.)

IBM MQ for z/OS supports z/OS UNIX System Services threads using the Batch Adapter as follows:

1. Heavyweight threads are fully supported as batch connections. Each thread runs in its own TCB, which is attached and detached at thread start and end. Should the thread end before issuing an MQDISC call, IBM MQ for z/OS performs its standard task cleanup, which includes committing any outstanding unit of work if the thread terminated normally, or backing it out if the thread terminated abnormally.
2. Medium-weight threads are fully supported, but if the TCB is going to be reused by another thread, the application must ensure that an MQDISC call, preceded by either MQCMIT or MQBACK, is issued before the next thread start. This implies that if the application has established a Program Interrupt Handler, and the application then abends, the Interrupt Handler must issue MQCMIT and MQDISC calls before reusing the TCB for another thread.

Note: Threading models do **not** support access to common IBM MQ resources from multiple threads.

z/OS *The API-crossing exit for z/OS*

This topic contains product-sensitive programming interface information.

An exit is a point in IBM-supplied code where you can run your own code. IBM MQ for z/OS provides an *API-crossing exit* that you can use to intercept calls to the MQI, and to monitor or modify the function of the MQI calls. This section describes how to use the API-crossing exit, and describes the sample exit program that is supplied with IBM MQ for z/OS.

This section is applicable only for users of CICS TS V3.1 and earlier. Users of CICS TS V3.2 and later should refer to the section CICS Integration with IBM MQ in the CICS product documentation.

Note

The API-crossing exit is invoked only by the CICS adapter of IBM MQ for z/OS. The exit program runs in the CICS address space.

z/OS *Writing your own exit program*

You can use the sample API-crossing exit program (CSQCAPX) that is supplied with IBM MQ for z/OS as a framework for your own program.

This is described in "The sample API-crossing exit program, CSQCAPX" on page 920.

When writing an exit program, to find the name of an MQI call issued by an application, examine the *ExitCommand* field of the MQXP structure. To find the number of parameters on the call, examine the *ExitParmCount* field. You can use the 16-byte *ExitUserArea* field to store the address of any dynamic storage that the application obtains. This field is retained across invocations of the exit and has the same lifetime as a CICS task.

If you are using CICS Transaction Server V3.2, you must write your exit program to be threadsafe and declare your exit program as threadsafe. If you are using earlier CICS releases, you are also recommended to write and declare your exit programs as threadsafe to be ready for migrating to CICS Transaction Server V3.2.

Your exit program can suppress execution of an MQI call by returning MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION in the *ExitResponse* field. To allow the call to be executed (and the exit program to be reinvoked after the call has completed), your exit program must return MQXCC_OK.

When invoked after an MQI call, an exit program can inspect and modify the completion and reason codes set by the call.

Usage notes

Here are some general points to consider when writing your exit program:

- For performance reasons, write your program in assembler-language. If you write it in any of the other languages supported by IBM MQ for z/OS, you must provide your own data definition file.
- Link-edit your program as AMODE(31) and RMODE(ANY).
- To define the exit parameter block to your program, use the assembler-language macro, CMQXPA.
- Specify CONCURRENCY(THREADSAFE) when you define your exit program and any programs that your exit program calls.
- If you are using the CICS Transaction Server for z/OS storage protection feature, your program must run in CICS execution key. That is, you must specify EXECKEY(CICS) when defining both your exit program and any programs to which it passes control. For information about CICS exit programs and the CICS storage protection facility, see the *CICS Customization Guide*.
- Your program can use all the APIs (for example, IMS, Db2, and CICS) that a CICS task-related user exit program can use. It can also use any of the MQI calls except MQCONN, MQCONNX, and MQDISC. However, any MQI calls within the exit program do not invoke the exit program a second time.
- Your program can issue EXEC CICS SYNCPOINT or EXEC CICS SYNCPOINT ROLLBACK commands. However, these commands commit or roll back ***all*** the updates done by the task up to the point that the exit was used, and so their use is not recommended.
- Your program must end by issuing an EXEC CICS RETURN command. It must not transfer control with an XCTL command.
- Exits are written as extensions to the IBM MQ for z/OS code. Ensure that your exit does not disrupt any IBM MQ for z/OS programs or transactions that use the MQI. These are typically indicated with a prefix of CSQ or CK.
- If CSQCAPX is defined to CICS, the CICS system attempts to load the exit program when CICS connects to IBM MQ for z/OS. If this attempt is successful, message CSQC301I is sent to the CKQC panel or to the system console. If the load is unsuccessful (for example, if the load module does not exist in any of the libraries in the DFHRPL concatenation), message CSQC315 is sent to the CKQC panel or to the system console.
- Because the parameters in the communication area are addresses, the exit program must be defined as local to the CICS system (that is, not as a remote program).

z/OS *The sample API-crossing exit program, CSQCAPX*

The sample exit program is supplied as an assembler-language program. The source file (CSQCAPX) is supplied in the library **thlqual**.SCSQASMS (where **thlqual** is the high-level qualifier used by your installation). This source file includes pseudocode that describes the program logic.

The sample program contains initialization code and a layout that you can use when writing your own exit programs.

The sample shows how to:

- Set up the exit parameter block
- Address the call and exit parameter blocks
- Determine for which MQI call the exit is being invoked
- Determine whether the exit is being invoked before or after processing of the MQI call
- Put a message on a CICS temporary storage queue
- Use the macro DFHEIENT for dynamic storage acquisition to maintain reentrancy
- Use DFHEIBLK for the CICS exec interface control block
- Trap error conditions
- Return control to the caller

Design of the sample exit program

The sample exit program writes messages to a CICS temporary storage queue (CSQ1EXIT) to show the operation of the exit.

The messages show whether the exit is being invoked before or after the MQI call. If the exit is invoked after the call, the message contains the completion code and reason code returned by the call. The sample uses named constants from the CMQXPA macro to check on the type of entry (that is, before or after the call).

The sample does not perform any monitoring function, but simply places time-stamped messages into a CICS queue indicating the type of call it is processing. This provides an indication of the performance of the MQI, as well as the correct functioning of the exit program.

Note: The sample exit program issues six EXEC CICS calls for each MQI call that is made while the program is running. If you use this exit program, IBM MQ for z/OS performance is degraded.

z/OS *Preparing and using the API-crossing exit*

The sample exit is supplied in source form only.

To use the sample exit, or an exit program that you have written, create a load library, as you would for any other CICS program, as described in "Building CICS applications in z/OS" on page 1054.

- For CICS Transaction Server for z/OS and CICS for MVS™/ESA, when you update the CICS system definition (CSD) data set, the definitions you need are in the member **thlqual**.SCSQPROC(CSQ4B100).

  Note: The definitions use a suffix of MQ. If this suffix is already used in your enterprise, this must be changed before the assembly stage.

If you use the default CICS program definitions supplied, the exit program CSQCAPX is installed in a ***disabled*** state. This is because using the exit program can produce a significant reduction in performance.

To activate the API-crossing exit temporarily:

1. Issue the command **CEMT S PROGRAM(CSQCAPX) ENABLED** from the CICS master terminal.
2. Run the CKQC transaction, and use option 3 in the Connection pull-down to alter the status of the API-crossing exit to **Enabled**.

If you want to run IBM MQ for z/OS with the API-crossing exit permanently enabled, with CICS Transaction Server for z/OS and CICS for MVS/ESA, do one of the following:

- Alter the CSQCAPX definition in member CSQ4B100, changing STATUS(DISABLED) to STATUS(ENABLED). You can update the CICS CSD definition using the CICS-supplied batch program DFHCSDUP.
- Alter the CSQCAPX definition in the CSQCAT1 group by changing the status from DISABLED to ENABLED.

In both cases, you must reinstall the group. You can do this by cold-starting your CICS system or by using the CICS CEDA transaction to reinstall the group while CICS is running.

Note: Using CEDA might cause an error if any of the entries in the group are currently in use.

End of product-sensitive programming interface information.

z/OS *Application programming with shared queues*

This topic provides information on some of the factors that you need to take into account when designing new applications to use shared queues, and when migrating existing applications to the shared-queue environment.

z/OS *Serializing your applications*

Certain types of applications might have to ensure that messages are retrieved from a queue in exactly the same order as they arrived on the queue.

For example, if IBM MQ is being used to shadow database updates on to a remote system, a message describing the update to a record must be processed after a message describing the insert of that record. In a local queuing environment, this is often achieved by the application that is getting the messages opening the queue with the MQOO_INPUT_EXCLUSIVE option, thus preventing any other getting application from processing the queue at the same time.

IBM MQ allows applications to open shared queues exclusively in the same way. However, if the application is working from a partition of a queue (for example, all database updates are on the same queue, but those for table A have a correlation identifier of A, and those for table B a correlation identifier of B), and applications want to get messages for table A updates and table B updates concurrently, the simple mechanism of opening the queue exclusively is not possible.

If this type of application is to take advantage of the high availability of shared queues, you might decide that another instance of the application that accesses the same shared queues, running on a secondary queue manager, should take over if the primary getting application or queue manager fails.

If the primary queue manager fails, two things happen:

- Shared queue peer recovery ensures that any incomplete updates from the primary application are completed or backed out.
- The secondary application takes over processing the queue.

The secondary application might start before all the incomplete units of work have been dealt with, which could lead to the secondary application retrieving the messages out of sequence. To solve this type of problem, the application can choose to be a *serialized application*.

A serialized application uses the `MQCONNX` call to connect to the queue manager, specifying a connection tag when it connects that is unique to that application. Any units of work performed by the application are marked with the connection tag. IBM MQ ensures that units of work within the queue sharing group with the same connection tag are serialized (according to the serialization options on the `MQCONNX` call).

This means that, if the primary application uses the `MQCONNX` call with a connection tag of `Database shadow retriever`, and the secondary takeover application attempts to use the `MQCONNX` call with an identical connection tag, the secondary application cannot connect to the second IBM MQ until any outstanding primary units of work have been completed, in this case by peer recovery.

Consider using the serialized application technique for applications that depend on the exact sequence of messages on a queue. In particular:

- Applications that must not restart after an application or queue manager failure until all commit and backout operations for the previous execution of the application are complete.

In this case, the serialized application technique is only applicable if the application works in syncpoint.

- Applications that must not start while another instance of the same application is already running.

  In this case, the serialized application technique is only required if the application cannot open the queue for exclusive input.

Note: IBM MQ only guarantees to preserve the sequence of messages when certain criteria are met. These are described in the description of MQGET.

z/OS *Applications that are not suitable for use with shared queues*

Some features of IBM MQ are not supported when you are using shared queues, so applications that use these features are not suitable for the shared queue environment.

Consider the following points when designing your shared-queue applications:

- Queue indexing is limited for shared queues. If you want to use the message identifier or correlation identifier to select the message that you want to get from the queue, the queue should be indexed with the correct value. If you are selecting messages by message identifier alone, the queue needs an index type of MQIT_MSG_ID (although you can also use MQIT_NONE). If you are selecting messages by correlation identifier alone, the queue must have an index type of MQIT_CORREL_ID.
- You cannot use temporary dynamic queues as shared queues. However, you can use permanent dynamic queues. The models for shared dynamic queues have a DEFTYPE of SHAREDYN (shared dynamic) although they are created and destroyed in the same way as PERMDYN (permanent dynamic) queues.

z/OS *Deciding whether to share non-application queues*

Use this information when considering sharing non-application queues.

There are queues other than application queues that you might want to consider sharing:

Initiation queues

If you define a shared initiation queue, you do not need to have a trigger monitor running on every queue manager in the queue sharing group, as long as there is at least one trigger monitor running. (You can also use a shared initiation queue even if there is a trigger monitor running on each queue manager in the queue sharing group.)

If you have a shared application queue and use the trigger type of EVERY (or a trigger type of FIRST with a small trigger interval, which behaves like a trigger type of EVERY) your initiation queue must always be a shared queue. For more information about when to use a shared initiation queue, see Table 131 on page 923.

SYSTEM.* queues

You can define the SYSTEM.ADMIN.* queues used to hold event messages as shared queues. This can be useful to check load balancing if an exception occurs. Each event message created by IBM MQ contains a correlation identifier indicating which queue manager produced it.

You must define the SYSTEM.QSG.* queues used for shared channels and intra-group queuing as shared queues.

You can also change the definitions of the SYSTEM.DEFAULT.LOCAL.QUEUE to be shared, or define your own default shared queue definition. See Defining system objects for IBM MQ for z/OS for more information.

You cannot define any other SYSTEM.* queues as shared queues.

z/OS *Migrating your existing applications to use shared queues*

Reason codes, triggering, and the MQINQ API call can work differently in a shared queue environment.

See Migrating non-shared queues to shared queues for information on migrating your existing queues to shared queues.

When you migrate your existing applications, consider the following things, which might work in a different way in the shared queue environment:

Reason codes
When you migrate your existing applications to use shared queues, check for the new reason codes that can be issued.

Triggering
If you are using a shared application queue, triggering works on committed messages only (on a non-shared application queue, triggering works on all messages).

If you use triggering to start applications, you might want to use a shared initiation queue. Table 131 on page 923 describes what you need to consider when deciding which type of initiation queue to use.

Table 131. When to use a shared-initiation queue

	Non-shared application queue	**Shared application queue**
Non-shared initiation queue	As for previous releases.	If you use a trigger type of FIRST or DEPTH, you can use a non-shared initiation queue with a shared application queue. Extra trigger messages might be generated, but this setup is good for triggering long-running applications (like the CICS bridge) and provides high availability. For trigger type FIRST or DEPTH, a trigger message triggers an instance of the application on every queue manager that is running a trigger monitor and that does not already have the application queue open for input. One trigger message is generated for every queue manager; if there is more than one trigger monitor running against the non-shared local initiation queue, on a particular queue manager, they will compete to process the message.

Table 131. When to use a shared-initiation queue (continued)		
	Non-shared application queue	**Shared application queue**
Shared initiation queue	Do not use a shared initiation queue with a non-shared application queue.	For trigger type EVERY, when an application puts a message to a shared application queue, the putting queue manager determines which queue managers have an interest in the trigger-every event and sends a notification to one of those queue managers. On the notified queue manager, the resulting action is to generate a trigger message to the initiation queue., **Note:** If you have a shared application queue that has a trigger type of EVERY, use a shared initiation queue, or you might lose trigger messages in certain circumstances; for example, a queue manager failing. For trigger type FIRST or DEPTH, one trigger message is generated by each queue manager that has the named initiation queue open for input. **Note:** For trigger type FIRST or DEPTH, if one trigger monitor instance is busy, this leaves the potential for less busy trigger monitors to process more than one trigger message from the shared initiation queue. Hence, multiple instances of the server application may be started against a given queue manager. Note that these multiple instances are started as a result of processing multiple trigger messages. Ordinarily, for trigger type FIRST or DEPTH, if an application instance is already serving an application queue, another trigger message will not be generated by the queue manager that the application is connected to.

MQINQ

When you use the MQINQ call to display information about a shared queue, the values of the number of MQOPEN calls that have the queue open for input and output relate only to the queue manager that issued the call. No information is produced about other queue managers in the queue sharing group that have the queue open.

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

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

- To use syncpoints and MQI calls in IMS applications, see “Writing IMS applications using IBM MQ” on page 73.
- To write applications that use the IBM MQ - IMS bridge, see “Writing IMS bridge applications” on page 77.

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

- “Writing IMS applications using IBM MQ” on page 73
- “Writing IMS bridge applications” on page 77

Related concepts

“Présentation de l'interface de file d'attente de messages” on page 739
Découvrez les composants MQI (Message Queue Interface).

“Connexion et déconnexion d'un gestionnaire de files d'attente” on page 753

Pour utiliser les services de programmation IBM MQ , un programme doit avoir une connexion à un gestionnaire de files d'attente. Utilisez ces informations pour apprendre à se connecter et à se déconnecter d'un gestionnaire de files d'attente.

“Ouverture et fermeture d'objets” on page 760
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets IBM MQ .

“Insertion de messages dans une file d'attente” on page 772
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

“Obtention de messages à partir d'une file d'attente” on page 787
Utilisez ces informations pour en savoir plus sur l'obtention de messages à partir d'une file d'attente.

“Interrogation et définition des attributs d'objet” on page 873
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet IBM MQ .

“Validation et annulation d'unités de travail” on page 876
Ces informations décrivent comment valider et rétablir les opérations d'extraction et d'insertion récupérables qui se sont produites dans une unité d'oeuvre.

“Démarrage des applications IBM MQ à l'aide de déclencheurs” on page 888
Découvrez les déclencheurs et comment démarrer les applications IBM MQ à l'aide de déclencheurs.

“Utilisation de l'interface MQI et des clusters” on page 909
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

“Using and writing applications on IBM MQ for z/OS” on page 914
IBM MQ for z/OS applications can be made up from programs that run in many different environments. This means that they can take advantage of the facilities available in more than one environment.

z/OS *Writing IMS applications using IBM MQ*

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

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

- “Syncpoints in IMS applications” on page 73
- “MQI calls in IMS applications” on page 74

Restrictions

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

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

- MQCB
- MQCB_FUNCTION
- MQCTL

Related concepts

“Writing IMS bridge applications” on page 77
This topic contains information about writing applications to use the IBM MQ - IMS bridge.

z/OS *Syncpoints in IMS applications*

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

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

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

All open handles are closed by the IMS adapter at a syncpoint (except in a batch or non-message driven BMP environment). This is because a different user could initiate the next unit of work and IBM MQ

security checking is performed when the MQCONN, MQCONNX, and MQOPEN calls are made, not when the MQPUT or MQGET calls are made.

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

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

z/OS *MQI calls in IMS applications*

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

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

- “Server applications” on page 926
- “Inquiry applications” on page 928

Server applications

Here is an outline of the MQI server application model:

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

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

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

Return
```

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

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

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

Return to calling function
```

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

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

Return to calling function
```

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

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

  This involves:

  – Reading a message
  – Processing the required updates

– Putting the reply

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

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

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

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

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

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

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

Inquiry applications

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

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

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

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

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

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

z/OS *Writing IMS bridge applications*

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

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

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

- “How the IMS bridge deals with messages” on page 77
- “Writing IMS transaction programs through IBM MQ” on page 936

Related concepts

“Writing IMS applications using IBM MQ” on page 73
There are further considerations when using IBM MQ in IMS applications These include which MQ API calls can be used and the mechanism used for syncpoint.

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

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

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

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

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

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

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

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

Note:

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

- Messages containing IMS transaction data but no MQIIH structure:

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

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

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

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

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

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

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

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

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

- “Mapping IBM MQ messages to IMS transaction types” on page 79
- “If the message cannot be put to the IMS queue” on page 79
- “IMS bridge feedback codes” on page 80
- “The MQMD fields in messages from the IMS bridge” on page 80
- “The MQIIH fields in messages from the IMS bridge” on page 81
- “Reply messages from IMS” on page 82
- “Using alternate response PCBs in IMS transactions” on page 82
- “Sending unsolicited messages from IMS” on page 82
- “Message segmentation” on page 83
- “Data conversion for messages to and from the IMS bridge” on page 83

Related concepts

“Writing IMS transaction programs through IBM MQ” on page 936
The coding required to handle IMS transactions through IBM MQ depends on the message format required by the IMS transaction and the range of responses it can return. However, there are several points to consider when your application handles IMS screen formatting information.

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

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

Table 132. How IBM MQ messages map to IMS transaction types

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

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

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

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

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

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

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

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

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

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

z/OS *IMS bridge feedback codes*

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

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

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

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

z/OS *The MQMD fields in messages from the IMS bridge*

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

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

StrucID
"MD "

Version
MQMD_VERSION_1

Report
MQRO_NONE

MsgType
MQMT_REPLY

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

Feedback
MQFB_NONE

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

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

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

Priority
MQMD.Priority of the input message

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

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

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

BackoutCount
0

ReplyToQ
Blanks

ReplyToQMgr
Blanks (set to local qmgr name by the queue manager during the MQPUT)

UserIdentifier
MQMD.UserIdentifier of the input message

AccountingToken
MQMD.AccountingToken of the input message

ApplIdentityData
MQMD.ApplIdentityData of the input message

PutApplType
MQAT_XCF if no error, otherwise MQAT_BRIDGE

PutApplName
<XCFgroupName><XCFmemberName> if no error, otherwise QMGR name

PutDate
Date when message was put

PutTime
Time when message was put

ApplOriginData
Blanks

z/OS *The MQIIH fields in messages from the IMS bridge*

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

The MQIIH of a message received from IMS is built as follows:

StrucId
"IIH "

Version
1

StrucLength
84

Encoding
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat of the input message if MQIIH.ReplyToFormat is not blank, otherwise IOPCB.MODNAME

Flags
0

LTermOverride
LTERM name (Tpipe) from OTMA header

MFSMapName
Map name from OTMA header

ReplyToFormat
Blanks

Authenticator
MQIIH.Authenticator of the input message if the reply message is being put to an MQ-IMS bridge queue, otherwise blanks.

TranInstanceId
Conversation ID / Server Token from OTMA header if in conversation. In versions of IMS prior to V14, this field is always nulls if not in conversation. From IMS V14 onwards, this field may be set by IMS even if not in conversation.

TranState
"C" if in conversation, otherwise blank

CommitMode
Commit mode from OTMA header ("0" or "1")

SecurityScope
Blank

Reserved
Blank

z/OS *Reply messages from IMS*

When an IMS transaction ISRTs to its IOPCB, the message is routed back to the originating LTERM or TPIPE.

These are seen in IBM MQ as reply messages. Reply messages from IMS are put onto the reply-to queue specified in the original message. If the message cannot be put onto the reply-to queue, it is put onto the dead-letter queue using the authority of the bridge. If the message cannot be put onto the dead-letter queue, a negative acknowledgment is sent to IMS to say that the message cannot be received. Responsibility for the message is then returned to IMS. If you are using commit mode 0, messages from that Tpipe are not sent to the bridge, and remain on the IMS queue; that is, no further messages are sent until restart. If you are using commit mode 1, other work can continue.

If the reply has an MQIIH structure, its format type is MQFMT_IMS; if not, its format type is specified by the IMS MOD name used when inserting the message.

z/OS *Using alternate response PCBs in IMS transactions*

When an IMS transaction uses alternate response PCBs (ISRTs to the ALTPCB, or issues a CHNG call to a modifiable PCB), the pre-routing exit (DFSYPRX0) is invoked to determine if the message should be rerouted.

If the message is to be rerouted, the destination resolution exit (DFSYDRU0) is invoked to confirm the destination and prepare the header information See Using OTMA exits in IMS and The pre-routing exit DFSYPRX0 for information about these exit programs.

Unless action is taken in the exits, all output from IMS transactions initiated from an IBM MQ queue manager, whether to the IOPCB or to an ALTPCB, will be returned to the same queue manager.

z/OS *Sending unsolicited messages from IMS*

To send messages from IMS to an IBM MQ queue, you need to invoke an IMS transaction that ISRTs to an ALTPCB.

You need to write pre-routing and destination resolution exits to route unsolicited messages from IMS and build the OTMA user data, so that the MQMD of the message can be built correctly. See The pre-routing exit DFSYPRX0 and The destination resolution user exit for information about these exit programs.

Note: The IBM MQ - IMS bridge does not know whether a message that it receives is a reply or an unsolicited message. It handles the message the same way in each case, building the MQMD and MQIIH of the reply based on the OTMA UserData that arrived with the message

Unsolicited messages can create new Tpipes. For example, if an existing IMS transaction switched to a new LTERM (for example PRINT01), but the implementation requires that the output be delivered through OTMA, a new Tpipe (called PRINT01 in this example) is created. By default, this is a non-synchronized Tpipe. If the implementation requires the message to be recoverable, set the destination resolution exit output flag. See the *IMS Customization Guide* for more information.

z/OS *Message segmentation*

You can define IMS transactions as expecting single- or multi-segment input.

The originating IBM MQ application must construct the user input following the MQIIH structure as one or more LLZZ-data segments. All segments of an IMS message must be contained in a single IBM MQ message sent with a single MQPUT.

The maximum length of an LLZZ-data segment is defined by IMS/OTMA (32767 bytes). The total IBM MQ message length is the sum of the LL bytes, plus the length of the MQIIH structure.

All the segments of the reply are contained in a single IBM MQ message.

There is a further restriction on the 32 KB limitation on messages with format MQFMT_IMS_VAR_STRING. When the data in an ASCII-mixed CCSID message is converted to an EBCDIC-mixed CCSID message, a shift-in byte or a shift-out byte is added every time that there is a transition between SBCS and DBCS characters. The 32 KB restriction applies to the maximum size of the message. That is, because the LL field in the message cannot exceed 32 KB, the message must not exceed 32 KB including all shift-in and shift-out characters. The application building the message must allow for this.

z/OS *Data conversion for messages to and from the IMS bridge*

The data conversion is performed by either the distributed queuing facility (which may call any necessary exits) or by the intra group queuing agent (which does not support the use of exits) when it puts a message to a destination queue that has XCF information defined for its storage class. The data conversion does not occur when a message is delivered to a queue by publish/subscribe.

Any exits needed must be available to the distributed queuing facility in the data set referenced by the CSQXLIB DD statement. This means that you can send messages to an IMS application using the IBM MQ - IMS bridge from any IBM MQ platform.

If there are conversion errors, the message is put to the queue unconverted; this results eventually in it being treated as an error by the IBM MQ - IMS bridge, because the bridge cannot recognize the header format. If a conversion error occurs, an error message is sent to the z/OS console.

See “Ecriture des exits de conversion de données” on page 1010 for detailed information about data conversion in general.

Sending messages to the IBM MQ - IMS bridge

To ensure that conversion is performed correctly, you must tell the queue manager what the format of the message is.

If the message has an MQIIH structure, the *Format* in the MQMD must be set to the built-in format MQFMT_IMS, and the *Format* in the MQIIH must be set to the name of the format that describes your message data. If there is no MQIIH, set the *Format* in the MQMD to your format name.

If your data (other than the LLZZs) is all character data (MQCHAR), use as your format name (in the MQIIH or MQMD, as appropriate) the built-in format MQFMT_IMS_VAR_STRING. Otherwise, use your own format name, in which case you must also provide a data-conversion exit for your format. The exit must handle the conversion of the LLZZs in your message, in addition to the data itself (but it does not have to handle any MQIIH at the start of the message).

If your application uses *MFSMapName*, you can use messages with the MQFMT_IMS instead, and define the map name passed to the IMS transaction in the MFSMapName field of the MQIIH.

Receiving messages from the IBM MQ - IMS bridge

If an MQIIH structure is present on the original message that you are sending to IMS, one is also present on the reply message.

To ensure that your reply is converted correctly:

- If you have an MQIIH structure on your original message, specify the format that you want for your reply message in the MQIIH *ReplytoFormat* field of the original message. This value is placed in the MQIIH *Format* field of the reply message. This is particularly useful if all your output data is of the form LLZZ<character data>.

- If you do not have an MQIIH structure on your original message, specify the format that you want for the reply message as the MFS MOD name in the IMS application's ISRT to the IOPCB.

z/OS *Writing IMS transaction programs through IBM MQ*

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

When an IMS transaction is started from a 3270 screen, the message passes through IMS Message Format Services. This can remove all terminal dependency from the data stream seen by the transaction. When a transaction is started through OTMA, MFS is not involved. If application logic is implemented in MFS, this must be re-created in the new application.

In some IMS transactions, the end-user application can modify certain 3270 screen behavior, for example, highlighting a field that has had invalid data entered. This type of information is communicated by adding a two-byte attribute field to the IMS message for each screen field that needs to be modified by the program.

Thus, if you are coding an application to mimic a 3270, you need to take account of these fields when building or receiving messages.

You might need to code information in your program to process:

- Which key is pressed (for example, Enter and PF1)
- Where the cursor is when the message is passed to your application
- Whether the attribute fields have been set by the IMS application
  - High, normal, or zero intensity
  - Color
  - Whether IMS is expecting the field back the next time that Enter is pressed
- Whether the IMS application has used null characters (X'3F') in any fields.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are using an MQIIH structure, set the MQMD format to MQFMT_IMS and the MQIIH format to MQFMT_IMS_VAR_STRING.

If your IMS message contains only character data (apart from the LLZZ-data segment), and you are **not** using an MQIIH structure, set the MQMD format to MQFMT_IMS_VAR_STRING and ensure that your IMS application specifies MODname MQFMT_IMS_VAR_STRING when replying. If a problem occurs (for example, user not authorized to use the transaction) and IMS sends an error message, this has an MODname of the form DFSMOx, where x is a number in the range 1 through 5. This is put in the MQMD.Format.

If your IMS message contains binary, packed, or floating point data (apart from the LLZZ-data segment), code your own data-conversion routines. Refer to *IMS/ESA Application Programming: Transaction Manager* for information about IMS screen formatting.

Consider the following topics when writing code to handle IMS transactions through IBM MQ.

- “Writing IBM MQ applications to invoke IMS conversational transactions” on page 936
- “Writing programs containing IMS commands” on page 937
- “Triggering” on page 937

Writing IBM MQ applications to invoke IMS conversational transactions

Use this information as a guide for considerations when writing IBM MQ application to invoke IMS conversational transactions.

When you write an application that invokes an IMS conversation, consider the following:

- Include an MQIIH structure with your application message.

- Set the *CommitMode* in MQIIH to MQICM_SEND_THEN_COMMIT.
- To invoke a new conversation, set *TranState* in MQIIH to MQITS_NOT_IN_CONVERSATION.
- To invoke second and subsequent steps of a conversation, set *TranState* to MQITS_IN_CONVERSATION, and set *TranInstanceId* to the value of that field returned in the previous step of the conversation.
- There is no easy way in IMS to find the value of a *TranInstanceId*, should you lose the original message sent from IMS.
- The application must check the *TranState* of messages from IMS to check whether the IMS transaction has terminated the conversation.
- You can use /EXIT to end a conversation. You must also quote the *TranInstanceId*, set *TranState* to MQITS_IN_CONVERSATION, and use the IBM MQ queue on which the conversation is being carried out.
- You cannot use /HOLD or /REL to hold or release a conversation.
- Conversations invoked through the IBM MQ - IMS bridge are terminated if IMS is restarted.

Writing programs containing IMS commands

An application program can build an IBM MQ message of the form LLZZ*command*, instead of a transaction, where *command* is of the form /DIS TRAN PART or /DIS POOL ALL.

Most IMS commands can be issued in this way; see *IMS V11 Communications and Connections* for details. The command output is received in the IBM MQ reply message in the text form as would be sent to a 3270 terminal for display.

OTMA has implemented a special form of the IMS display transaction command, which returns an architected form of the output. The exact format is defined in *IMS V11 Communications and Connections*. To invoke this form from an IBM MQ message, build the message data as before, for example /DIS TRAN PART, and set the TranState field in the MQIIH to MQITS_ARCHITECTED. IMS processes the command, and returns the reply in the architected form. An architected response contains all the information that could be found in the text form of the output, and one additional piece of information: whether the transaction is defined as recoverable or non-recoverable.

Triggering

The IBM MQ - IMS bridge does not support trigger messages.

If you define an initiation queue that uses a storage class with XCF parameters, messages put to that queue are rejected when they get to the bridge.

Ecriture d'applications de procédure client

Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

Les applications peuvent être générées et exécutées dans l'environnement client IBM MQ . L'application doit être générée et liée au IBM MQ MQI client utilisé. La façon dont les applications sont générées et liées varie en fonction de la plateforme et du langage de programmation utilisés. Pour plus d'informations sur la génération d'applications client, voir «Génération d'applications pour IBM MQ MQI clients», à la page 944.

Vous pouvez exécuter une application IBM MQ à la fois dans un environnement IBM MQ complet et dans un environnement IBM MQ MQI client sans modifier votre code, à condition que certaines conditions soient remplies. Pour plus d'informations sur l'exécution de vos applications dans l'environnement client IBM MQ , voir «Exécution d'applications dans l'environnement IBM MQ MQI client», à la page 946.

Si vous utilisez l'interface de file d'attente de messages (MQI) pour écrire des applications à exécuter dans un environnement IBM MQ MQI client , il existe des contrôles supplémentaires à imposer lors d'un appel MQI afin de garantir que le traitement de l'application IBM MQ n'est pas interrompu. Pour plus

d'informations sur ces contrôles, voir «Utilisation de l'interface MQI dans une application client», à la page 938.

Pour plus d'informations sur la préparation et l'exécution d'autres types d'application en tant qu'applications client, voir les rubriques suivantes:

- «Préparation et exécution des applications CICS et Tuxedo», à la page 959
- «Préparation et exécution des applications Microsoft Transaction Server», à la page 51
- «Préparation et exécution des applications IBM MQJMS», à la page 962

Concepts associés

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

«Ecriture d'une application de procédure pour la mise en file d'attente», à la page 738
La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

«Ecriture d'applications de publication / abonnement», à la page 829
Commencez à écrire des applications IBM MQ de publication / abonnement.

«Création d'une application procédurale», à la page 1027
Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

«Traitement des erreurs de programme de procédure», à la page 1065
Ces informations expliquent les erreurs associées aux appels MQI de vos applications lors de l'exécution d'un appel ou de la distribution de son message à sa destination finale.

Tâches associées

«Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085
Ces exemples de programmes sont écrits dans des langages procéduraux et présentent des utilisations typiques de l'interface MQI (Message Queue Interface). Programmes IBM MQ sur différentes plateformes.

Utilisation de l'interface MQI dans une application client

Cette collection de rubriques prend en compte les différences entre l'écriture de votre application IBM MQ à exécuter dans un environnement client d'interface de file d'attente de messages (MQI) et à exécuter dans l'environnement de gestionnaire de files d'attente IBM MQ complet.

Lorsque vous concevez une application, tenez compte des contrôles que vous devez imposer lors d'un appel MQI pour vous assurer que le traitement de l'application IBM MQ n'est pas interrompu.

Avant de pouvoir exécuter des applications qui utilisent l'interface MQI, vous devez créer certains objets IBM MQ . Pour plus d'informations, voir Programmes d'application utilisant l'interface MQI.

Limitation de la taille d'un message dans une application client

Un gestionnaire de files d'attente a une longueur de message maximale, mais la taille maximale des messages que vous pouvez transmettre à partir d'une application client est limitée par la définition de canal.

L'attribut de longueur maximale de message (MaxMsgLength) d'un gestionnaire de files d'attente correspond à la longueur maximale d'un message pouvant être traité par ce gestionnaire de files d'attente.

Multi Sous Multiplateformes, vous pouvez augmenter l'attribut de longueur de message maximale d'un gestionnaire de files d'attente. Pour plus d'informations, voir ALTER QMGR.

Vous pouvez déterminer la valeur de la longueur MaxMsgpour un gestionnaire de files d'attente à l'aide de l'appel MQINQ.

Si l'attribut MaxMsgLength est modifié, il n'est pas vérifié qu'il n'existe pas déjà de files d'attente, ni même de messages, dont la longueur est supérieure à la nouvelle valeur. Après avoir modifié cet attribut, redémarrez les applications et les canaux afin de vous assurer que la modification a pris effet. Il n'est alors pas possible de générer de nouveaux messages dépassant la longueur MaxMsgdu gestionnaire de files d'attente ou de la file d'attente (sauf si la segmentation du gestionnaire de files d'attente est autorisée).

La longueur maximale d'un message dans une définition de canal limite la taille d'un message que vous pouvez transmettre via une connexion client. Si une application IBM MQ tente d'utiliser l'appel MQPUT ou l'appel MQGET avec un message plus grand que celui-ci, un code d'erreur est renvoyé à l'application. Le paramètre de taille de message maximale de la définition de canal n'affecte pas la taille de message maximale pouvant être consommée à l'aide de MQCB sur une connexion client.

Concepts associés

«Utilisation de MQCONNX», à la page 943
Vous pouvez utiliser l'appel MQCONNX pour spécifier une structure de définition de canal (MQCD) dans la structure MQCNO.

Référence associée

Longueur maximale des messages (MAXMSGL)
ALTER CHANNEL
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Choix du CCSID client ou serveur

Utilisez l'ID de jeu de caractères codés (CCSID) local pour le client. Le gestionnaire de files d'attente effectue la conversion nécessaire. Vous pouvez utiliser la variable d'environnement **MQCCSID** pour remplacer le CCSID. Si votre application exécute plusieurs PUT, les zones CCSID et de codage du MQMD peuvent être écrasées après la première opération PUT.

Les données transmises via l'interface de file d'attente de messages (MQI) de l'application au module de remplacement client doivent être dans le CCSID local, codées pour le IBM MQ MQI client. Si le gestionnaire de files d'attente connecté requiert la conversion des données, la conversion est effectuée par le code de support client sur le gestionnaire de files d'attente.

Dans IBM WebSphere MQ 7.0 et les versions ultérieures, le client Java peut effectuer la conversion si le gestionnaire de files d'attente ne peut pas le faire. Voir «IBM MQ classes for Java connexions client», à la page 382.

Le code client suppose que les données de type caractère qui traversent l'interface MQI dans le client sont dans le CCSID configuré pour ce poste de travail. Si ce CCSID n'est pas pris en charge ou n'est pas le CCSID requis, il peut être remplacé par la variable d'environnement **MQCCSID** à l'aide de l'une des commandes suivantes:

- Windows

```
SET MQCCSID=850
```

- Linux AIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Si ce paramètre est défini dans le profil, toutes les données MQI sont supposées être dans la page de codes 850.

Remarque : L'hypothèse concernant la page de codes 850 ne s'applique pas aux données d'application du message.

Si votre application exécute plusieurs opérations PUT qui incluent des en-têtes IBM MQ après le descripteur de message (MQMD), sachez que les zones CCSID et de codage du MQMD sont écrasées après la fin de la première opération PUT.

Après la première opération PUT, ces zones contiennent la valeur utilisée par le gestionnaire de files d'attente connecté pour convertir les en-têtes IBM MQ . Assurez-vous que votre application réinitialise les valeurs aux valeurs requises.

Utilisation de MQINQ dans une application client

Certaines valeurs interrogées à l'aide de MQINQ sont modifiées par le code client.

CCSID
est défini sur le CCSID client, et non sur celui du gestionnaire de files d'attente.

Longueur maximale des messages
est réduit s'il est restreint par la définition de canal. Il s'agit de la plus basse des valeurs suivantes:

- Valeur définie dans la définition de file d'attente, ou
- Valeur définie dans la définition de canal

Pour plus d'informations, voir MQINQ.

Utilisation de la coordination des points de synchronisation dans une application client

Une application exécutée sur le client de base peut émettre des commandes MQCMIT et MQBACK, mais la portée du contrôle de point de synchronisation est limitée aux ressources MQI. Vous pouvez utiliser un gestionnaire de transactions externe avec un client transactionnel étendu.

Dans IBM MQ, l'un des rôles du gestionnaire de files d'attente est le contrôle de point de synchronisation dans une application. Si une application s'exécute sur un client de base IBM MQ , elle peut émettre MQCMIT et MQBACK, mais la portée du contrôle de point de synchronisation est limitée aux ressources MQI. L'instruction IBM MQ MQBEGIN n'est pas valide dans un environnement client de base.

Les applications qui s'exécutent dans l'environnement de gestionnaire de files d'attente complet sur le serveur peuvent coordonner plusieurs ressources (par exemple des bases de données) via un moniteur de transactions. Sur le serveur, vous pouvez utiliser le moniteur de transactions fourni avec les produits IBM MQ ou un autre moniteur de transactions tel que CICS. Vous ne pouvez pas utiliser un moniteur de transactions avec une application client de base.

Vous pouvez utiliser un gestionnaire de transactions externe avec un client transactionnel étendu IBM MQ . Voir Qu'est-ce qu'un client transactionnel étendu? pour en savoir plus.

Utilisation de la lecture anticipée dans une application client

Vous pouvez utiliser la lecture anticipée sur un client pour permettre l'envoi de messages non persistants à un client sans que l'application client ait à demander les messages.

Lorsqu'un client requiert un message d'un serveur, il envoie une demande au serveur. Il envoie une demande distincte pour chacun des messages qu'il consomme. Pour améliorer les performances d'un client consommant des messages non persistants en évitant d'avoir à envoyer ces messages de demande, un client peut être configuré pour utiliser la lecture anticipée. La lecture anticipée permet d'envoyer des messages à un client sans qu'une application n'ait à les demander.

L'utilisation de la lecture anticipée peut améliorer les performances lors de la consommation de messages non persistants à partir d'une application client. Cette amélioration des performances est disponible pour les applications MQI et JMS . Les applications client utilisant MQGET ou la consommation asynchrone bénéficient des améliorations de performances lors de la consommation de messages non persistants.

Lorsque vous appelez MQOPEN avec MQOO_READ_AHEAD, le client IBM MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- L'application client doit être compilée et liée dans les bibliothèques client IBM MQ MQI des unités d'exécution.
- Le canal client doit utiliser le protocole TCP/IP.
- Le paramètre SharingConversations (SHARECNV) du canal doit avoir une valeur différente de zéro dans la définition de canal serveur et dans la définition de canal client.

Lorsque la lecture anticipée est activée, les messages sont envoyés à une mémoire tampon sur le client appelée mémoire tampon de lecture anticipée. Le client dispose d'une mémoire tampon de lecture anticipée pour chaque file d'attente qu'il ouvre avec la lecture anticipée activée. Les messages de la mémoire tampon de lecture anticipée ne sont pas conservés. Le client met régulièrement à jour le serveur avec des informations sur la quantité de données qu'il a consommée.

Toutes les conceptions d'application client ne sont pas adaptées à l'utilisation de la lecture anticipée car toutes les options ne sont pas prises en charge. Certaines options doivent être cohérentes entre les appels MQGET lorsque la lecture anticipée est activée. Si un client modifie ses critères de sélection entre les appels MQGET, les messages stockés dans la mémoire tampon de lecture anticipée restent bloqués dans la mémoire tampon de lecture anticipée du client. Pour plus d'informations, voir «Amélioration des performances des messages non persistants», à la page 806

La configuration de la lecture anticipée est contrôlée par trois attributs, MaximumSize, PurgeTimeet UpdatePercentage, qui sont spécifiés dans la section MessageBuffer du fichier de configuration du client IBM MQ .

Utilisation de l'insertion asynchrone dans une application client

Avec l'insertion asynchrone, une application peut placer un message dans une file d'attente sans attendre de réponse du gestionnaire de files d'attente. Vous pouvez utiliser cette méthode pour accélérer la distribution des messages dans certaines situations.

Normalement, lorsqu'une application insère un ou plusieurs messages dans une file d'attente à l'aide de MQPUT ou de MQPUT1, elle doit attendre que le gestionnaire de files d'attente confirme qu'elle a traité la demande MQI. Vous pouvez améliorer les performances de la messagerie, en particulier pour les applications qui utilisent des liaisons client, et les applications qui placent un grand nombre de messages de petite taille dans une file d'attente, en choisissant plutôt d'insérer des messages de manière asynchrone. Lorsqu'une application insère un message de manière asynchrone, le gestionnaire de files d'attente ne renvoie pas la réussite ou l'échec de chaque appel, mais vous pouvez rechercher les erreurs régulièrement.

Pour insérer un message dans une file d'attente de manière asynchrone, utilisez l'option MQPMO_ASYNC_RESPONSE dans la zone *Options* de la structure MQPMO.

Si un message n'est pas éligible pour une insertion asynchrone, il est inséré dans une file d'attente de manière synchrone.

Lorsque vous demandez une réponse d'insertion asynchrone pour MQPUT ou MQPUT1, un CompCode et une raison de MQCC_OK et MQRC_NONE ne signifient pas nécessairement que le message a été correctement inséré dans une file d'attente. Bien que la réussite ou l'échec de chaque appel MQPUT ou MQPUT1 individuel puisse ne pas être renvoyé immédiatement, la première erreur qui s'est produite lors d'un appel asynchrone peut être déterminée ultérieurement via un appel à MQSTAT.

Pour plus de détails sur MQPMO_ASYNC_RESPONSE, voir Options MQPMO.

L'exemple de programme Asynchronous put illustre certaines des fonctions disponibles. Pour plus de détails sur les fonctions et la conception du programme, ainsi que sur la façon de l'exécuter, voir «Exemple de programme d'insertion asynchrone», à la page 1108.

Utilisation du partage de conversations dans une application client

Dans un environnement dans lequel le partage de conversations est autorisé, les conversations peuvent partager une instance de canal MQI.

Le partage de conversations est contrôlé par deux zones appelées SharingConversations, l'une faisant partie de la structure de définition de canal (MQCD) et l'autre de la structure du paramètre d'exit de canal (MQCXP). La zone SharingConversations de la structure MQCD est une valeur entière déterminant le nombre maximal de conversations pouvant partager une instance de canal associée au canal. La zone SharingConversations de la structure MQCXP est une valeur booléenne indiquant si l'instance de canal est actuellement partagée.

Dans un environnement dans lequel le partage de conversations n'est pas autorisé, les nouvelles connexions client spécifiant des structures MQCD identiques ne partagent pas d'instance de canal.

Une nouvelle connexion d'application client partagera l'instance de canal lorsque les conditions suivantes sont remplies :

- Les deux extrémités (connexion client et connexion serveur) de l'instance de canal sont configurées pour le partage de conversations et ces valeurs ne sont pas remplacées par des exits de canal.
- La valeur MQCD de connexion client (fournie au niveau de l'appel client MQCONNX ou à partir de la table de définition de canal client (CCDT)) correspond exactement à la valeur MQCD de connexion client fournie au niveau de l'appel client MQCONNX ou à partir de la table de définition de canal client lorsque l'instance de canal existante a été établie pour la première fois. Il est à noter que la structure MQCD d'origine a pu être modifiée ultérieurement par des exits ou via des négociations de canal, mais la correspondance s'effectue au niveau de la valeur fournie au système client avant l'application de ces modifications.
- Le nombre maximal de conversations partagées côté serveur n'est pas dépassé.

Si une nouvelle connexion d'application client correspond aux critères d'exécution du partage d'une instance de canal avec d'autres conversations, cette décision est prise avant que des exits soient appelés pendant cette conversation. Les exits d'une telle conversation ne peuvent pas modifier le fait qu'elle partage l'instance de canal avec d'autres conversations. Si aucune instance de canal existante ne correspond à la nouvelle définition de canal, une nouvelle instance de canal est connectée.

La négociation de canal ne se produit que pour la première conversation sur une instance de canal ; les valeurs négociées pour l'instance de canal sont fixes à ce stade et ne peuvent pas être modifiées lors du démarrage des conversations ultérieures. L'authentification TLS se produit également uniquement pour la première conversation.

Si la valeur MQCD SharingConversations est modifiée pendant l'initialisation des exits de sécurité, d'émission ou de réception pour la première conversation sur le socket à l'extrémité de connexion client ou de connexion serveur de l'instance de canal, la nouvelle valeur qui lui est affectée après l'initialisation de tous ces exits est utilisée pour déterminer la valeur des conversations partagées de l'instance de canal (la valeur la plus faible prévaut).

Si la valeur négociée des conversations partagées est égale à zéro, cela signifie que l'instance de canal n'est jamais partagée. D'autres programmes d'exit qui paramètrent cette zone sur zéro s'exécutent de la même façon sur leur propre instance de canal.

Si la valeur négociée des conversations partagées est supérieure à zéro, MQCXP SharingConversations a pour valeur TRUE pour les appels ultérieurs émis aux exits, ce qui indique que d'autres programmes d'exit sur cette instance de canal peuvent être entrés simultanément avec celui-ci.

Lorsque vous écrivez un programme d'exit de canal, déterminez si celui-ci va s'exécuter sur une instance de canal pouvant impliquer le partage de conversations. Si l'instance de canal peut impliquer le partage de conversations, étudiez l'impact de la modification des zones MQCD sur d'autres instances de l'exit de canal ; toutes les zones MQCD comportent des valeurs communes à toutes les conversations partagées. Une fois l'instance de canal établie, si des programmes d'exit essaient de modifier des zones MQCD, ils

risquent de rencontrer des problèmes car d'autres instances des programmes d'exit en cours d'exécution sur l'instance de canal peuvent tenter de modifier les mêmes zones simultanément. Si cette situation se produit au niveau de vos programmes d'exit, vous devez sérialiser l'accès à la structure MQCD dans votre code d'exit.

Si vous utilisez un canal qui est défini pour le partage de conversations et que vous ne voulez pas de partage sur une instance de canal particulière, paramétrez la valeur MQCD de SharingConversations sur 1 ou 0 lorsque vous initialisez un exit de canal sur la première conversation de l'instance de canal. Pour une explication des valeurs de SharingConversations, voir SharingConversations.

Exemple

Le partage de conversations est activé.

Vous utilisez une définition de canal de connexion client spécifiant un programme d'exit.

Au premier démarrage de ce canal, le programme d'exit modifie certains paramètres MQCD lorsqu'il est initialisé. Ceux-ci sont mis en oeuvre par le canal, de sorte que la définition avec laquelle le canal est en cours d'exécution est désormais différente de celle initialement fournie. Le paramètre MQCXP SharingConversations a pour valeur TRUE.

Lors de la prochaine connexion de l'application à l'aide de ce canal, la conversation s'exécute sur l'instance de canal précédemment démarrée, car elle comporte la même définition de canal d'origine. L'instance de canal à laquelle l'application se connecte la deuxième fois est la même instance que lors de la première connexion. Elle utilise donc les définitions ayant été modifiées par le programme d'exit. Lorsque le programme d'exit est initialisé pour la seconde conversation, bien qu'il puisse modifier les zones MQCD, elles ne sont pas traitées par le canal. Ces mêmes caractéristiques s'appliquent à toutes les conversations ultérieures qui partagent l'instance de canal.

Utilisation de MQCONNX

Vous pouvez utiliser l'appel MQCONNX pour spécifier une structure de définition de canal (MQCD) dans la structure MQCNO.

Cela permet à l'application client appelante de spécifier la définition du canal de connexion client lors de l'exécution. Pour plus d'informations, voir Création d'un canal de connexion client sur le IBM MQ MQI client à l'aide de MQCNO. Lorsque vous utilisez MQCONNX, l'appel émis sur le serveur dépend du niveau du serveur et de la configuration du programme d'écoute.

Lorsque vous utilisez MQCONNX à partir d'un client, les options suivantes sont ignorées:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

La structure MQCD que vous pouvez utiliser dépend du numéro de version MQCD que vous utilisez. Pour plus d'informations sur les versions MQCD (MQCD_VERSION), voir Version MQCD. Vous pouvez utiliser la structure MQCD, par exemple, pour transmettre des programmes d'exit de canal au serveur. Si vous utilisez MQCD version 3 ou ultérieure, vous pouvez utiliser la structure pour transmettre un tableau d'exits au serveur. Vous pouvez utiliser cette fonction pour effectuer plusieurs opérations sur le même message, telles que le chiffrement et la compression, en ajoutant un exit pour chaque opération, au lieu de modifier un exit existant. Si vous ne spécifiez pas de tableau dans la structure MQCD, les zones d'exit unique seront vérifiées. Pour plus d'informations sur les programmes de sortie de chaîne, voir«Programmes d'exit de canal pour les canaux de messagerie», à la page 988 .

Poignées de connexion partagées sur MQCONNX

Vous pouvez partager des handles entre différents threads au sein du même processus, à l’aide de handles de connexion partagés.

Lorsque vous spécifiez un handle de connexion partagé, le handle de connexion renvoyé par l'appel MQCONNX peut être transmis lors des appels MQI suivants sur n'importe quel thread du processus.

Remarque : Vous pouvez utiliser un handle de connexion partagé sur unIBM MQ MQI client pour vous connecter à un gestionnaire de files d'attente de serveur qui ne prend pas en charge les descripteurs de connexion partagés.

Génération d'applications pour IBM MQ MQI clients

Les applications peuvent être générées et exécutées dans l'environnement IBM MQ MQI client . L'application doit être générée et liée au IBM MQ MQI client utilisé. La façon dont les applications sont générées et liées varie en fonction de la plateforme et du langage de programmation utilisés.

Si une application doit s'exécuter dans un environnement client, vous pouvez l'écrire dans les langues indiquées dans le tableau suivant:

Tableau 133. Langages de programmation pris en charge dans les environnements client

Plateforme client	C	C++	COBOL	pTAL	Report Program Generator	Visual Basic
AIX AIX	Oui	Oui	Oui			
IBM i IBM i	Oui		Oui		Oui	
Linux Linux	Oui	Oui	Oui			
Windows Windows	Oui	Oui	Oui			Oui

Multi *Liaison d'applications C avec le code IBM MQ MQI client*

Après avoir écrit votre application IBM MQ que vous souhaitez exécuter sur le IBM MQ MQI client, vous devez la lier au code IBM MQ MQI client .

Vous pouvez lier votre application au code IBM MQ MQI client de deux manières:

1. Directement, en connectant votre application à un gestionnaire de files d'attente, auquel cas le gestionnaire de files d'attente doit se trouver sur la même machine que votre application.
2. Vers un fichier de bibliothèque client, qui vous permet d'accéder aux gestionnaires de files d'attente sur la même machine ou sur une machine différente.

IBM MQ fournit un fichier de bibliothèque client pour chaque environnement:

AIX **AIX**
Bibliothèque libmqic.a pour les applications sans unités d'exécution ou bibliothèque libmqic_r.a pour les applications avec unités d'exécution.

Linux **Linux**
Bibliothèque libmqic.so pour les applications non à unités d'exécution ou bibliothèque libmqic_r.so pour les applications à unités d'exécution.

IBM i **IBM i**
Liez l'application client avec le programme de service client LIBMQIC pour les applications sans unités d'exécution ou le programme de service LIBMQIC_R pour les applications avec unités d'exécution.

Windows **Windows**
MQIC32.LIB.

ALW *Liaison d'applications C++ avec le code IBM MQ MQI client*

Vous pouvez écrire des applications à exécuter sur le client en C + +. Les méthodes de génération varient en fonction de l'environnement.

Pour plus d'informations sur la façon de lier vos applications C + +, voir Génération de programmes IBM MQ C++.

Pour plus de détails sur tous les aspects de l'utilisation de C + +, voir Utilisation de C++

Multi *Liaison d'applications COBOL avec le code IBM MQ MQI client*

Après avoir écrit une application COBOL que vous souhaitez exécuter sur le IBM MQ MQI client, vous devez la lier à une bibliothèque appropriée.

IBM MQ fournit un fichier de bibliothèque client pour chaque environnement:

AIX
Liez votre application COBOL sans unités d'exécution à la bibliothèque libmqicb.a ou à l'application COBOL à unités d'exécution avec libmqicb_r.a.

IBM i
Liez l'application client COBOL avec le programme de service AMQCSTUB pour les applications non à unités d'exécution ou le programme de service AMQCSTUB_R pour les applications à unités d'exécution.

Windows
Liez votre code d'application à la bibliothèque MQICCBB pour COBOL 32 bits. IBM MQ MQI client for Windows ne prend pas en charge le langage COBOL 16 bits.

Windows *Liaison d'applications Visual Basic avec le code IBM MQ MQI client*

Vous pouvez lier des applications Microsoft Visual Basic avec le code IBM MQ MQI client sous Windows.

Deprecated

Depuis la IBM MQ 9.0, la prise en charge de Microsoft Visual Basic 6.0 est obsolète. IBM MQ classes for .NET est la technologie de remplacement recommandée. Pour plus d'informations, voir Développement d'applications .NET.

Liez votre application Visual Basic aux fichiers d'inclusion suivants:

CMQB.bas
Interface MQI

CMQBB.bas
MQAI

CMQCFB.bas
Commandes PCF

CMQXB.bas
Canaux

Définissez `mqtype=2` pour le client dans le compilateur Visual Basic afin de garantir la sélection automatique correcte de la dll client:

MQIC32.dll
Windows 7, Windows 8, Windows 2008 et Windows 2012

Concepts associés

«Codage dans Visual Basic», à la page 1080
Informations à prendre en compte lors du codage des programmes IBM MQ dans Microsoft Visual Basic. Visual Basic est pris en charge uniquement sous Windows.

«Préparation des programmes Visual Basic dans Windows», à la page 1047
Informations à prendre en compte lors de l'utilisation des programmes Microsoft Visual Basic sous Windows.

Exécution d'applications dans l'environnement IBM MQ MQI client

Vous pouvez exécuter une application IBM MQ à la fois dans un environnement IBM MQ complet et dans un environnement IBM MQ MQI client sans modifier votre code, à condition que certaines conditions soient remplies.

Ces conditions sont les suivantes:

- L'application n'a pas besoin de se connecter simultanément à plusieurs gestionnaires de files d'attente.
- Le nom du gestionnaire de files d'attente n'est pas précédé d'un astérisque (*) dans un appel `MQCONN` ou `MQCONNX` .
- L'application n'a pas besoin d'utiliser les exceptions répertoriées dans Quelles applications s'exécutent sur un IBM MQ MQI client?

Remarque : Les bibliothèques que vous utilisez lors de l'édition de liens déterminent l'environnement dans lequel votre application doit s'exécuter.

Lorsque vous travaillez dans l'environnement IBM MQ MQI client , n'oubliez pas que:

- Chaque application exécutée dans l'environnement IBM MQ MQI client possède ses propres connexions aux serveurs. Une application établit une connexion à un serveur chaque fois qu'elle émet un appel `MQCONN` ou `MQCONNX` .
- Une application envoie et obtient des messages de manière synchrone. Cela implique une attente entre le moment où l'appel est émis sur le client et le retour d'un code achèvement et d'un code anomalie sur le réseau.
- Toutes les conversions de données sont effectuées par le serveur, mais voir aussi MQCCSID pour plus d'informations sur le remplacement du CCSID configuré de la machine.

Connexion des applications IBM MQ MQI client aux gestionnaires de files d'attente

Une application s'exécutant dans un environnement IBM MQ MQI client peut se connecter à un gestionnaire de files d'attente de différentes manières. Vous pouvez utiliser des variables d'environnement, la structure MQCNO ou une table de définition de client.

Lorsqu'une application s'exécutant dans un environnement client IBM MQ émet un appel MQCONN ou MQCONNX, le client identifie la façon dont il doit établir la connexion. Lorsqu'un appel MQCONNX est émis par une application sur un client IBM MQ , la bibliothèque client MQI recherche les informations de canal du client dans l'ordre suivant:

1. Utilisation du contenu des zones `ClientConnOffset` ou `ClientConnPtr` de la structure MQCNO (si fournie). Ces zones identifient la structure de définition de canal (MQCD) à utiliser comme définition du canal de connexion client. Les détails de connexion peuvent être remplacés à l'aide d'un exit de préconnexion. Pour plus d'informations, voir «Référencement des définitions de connexion à l'aide d'un exit de préconnexion à partir d'un référentiel», à la page 1020.
2. Si la variable d'environnement **MQSERVER** est définie, le canal qu'elle définit est utilisé.
3. Si un fichier `mqclient.ini` est défini et que la strophe Channels contient un attribut **ServerConnectionParms** , le canal qu'il définit est utilisé. Pour plus d'informations, voir Fichier de configurationIBM MQ MQI client , `mqclient.ini` et Strophe Channels du fichier de configuration client.
4. Si les variables d'environnement **MQCHLLIB** et **MQCHLTAB** sont définies, la table de définition de canal du client vers laquelle elles pointent est utilisée. Sinon, la variable d'environnement **MQCCDTURL** offre la capacité équivalente à la définition d'une combinaison des variables d'environnement **MQCHLLIB** et **MQCHLTAB** . Si **MQCCDTURL** est défini, la table de définition de canal du client vers laquelle il pointe est utilisée. Pour plus d'informations, voir Accès par URL à la table de définition de canal du client.
5. Si un fichier `mqclient.ini` est défini et que la strophe Channels contient des attributs **ChannelDefinitionDirectory** et **ChannelDefinitionFile** , ces attributs sont utilisés pour localiser la table de définition de canal du client. Pour plus d'informations, voir Fichier de configurationIBM MQ MQI client , `mqclient.ini` et Strophe Channels du fichier de configuration client.

6. Enfin, si les variables d'environnement ne sont pas définies, le client recherche une table de définition de canal du client avec un chemin et un nom établis à partir de l'attribut **DefaultPrefix** de la strophe de gestionnaires AllQueuedans le fichier `mqs.ini` . Pour plus d'informations, voir la section AllQueueManagers du fichier mqs.ini.

   Si la recherche d'une table de définition de canal du client échoue, le client utilise les chemins suivants:

   - Linux AIX Sous AIX and Linux : `/var/mqm/AMQCLCHL.TAB`
   - Windows Sous Windows : `C:\Program Files\IBM\MQ\amqclchl.tab`
   - IBM i Sous IBM i : `/QIBM/UserData/mqm/@ipcc`
   - MQ Appliance Sous IBM MQ Appliance: *QMname*`_AMQCLCHL.TAB`. Ils apparaissent sous `mqbackup:// URI`.

La première des options décrites dans la liste précédente (à l'aide des zones `ClientConnOffset` ou `ClientConnPtr` de MQCNO) est prise en charge uniquement par l'appel MQCONNX. Si l'application utilise MQCONN au lieu de MQCONNX, les informations de canal sont recherchées dans les cinq autres manières, dans l'ordre indiqué dans la liste. Si le client ne parvient pas à trouver les informations de canal, l'appel MQCONN ou MQCONNX échoue.

Le nom de canal (pour la connexion client) doit correspondre au nom de canal de connexion serveur défini sur le serveur pour que l'appel MQCONN ou MQCONNX aboutisse.

Concepts associés

Accès Web adressable à la table de définition de canal du client

Tâches associées

Configuration des connexions entre le serveur et le client

Référence associée

Table de définition de canal du client

MQCNO-Options de connexion

Connexion d'applications client à des gestionnaires de files d'attente à l'aide de variables d'environnement
Les informations de canal client peuvent être fournies à une application exécutée dans un environnement client par des variables d'environnement.

Une application s'exécutant dans un environnement IBM MQ MQI client peut se connecter à un gestionnaire de files d'attente à l'aide des variables d'environnement suivantes:

MQSERVER
: La variable d'environnement **MQSERVER** est utilisée pour définir un canal minimal. **MQSERVER** indique l'emplacement du serveur IBM MQ et la méthode de communication à utiliser.

MQCHLLIB
: La variable d'environnement **MQCHLLIB** indique le chemin de répertoire du fichier contenant la table de définition de canal du client (CCDT). Le fichier est créé sur le serveur, mais il peut être copié sur le poste de travail IBM MQ MQI client .

MQCHLTAB
: La variable d'environnement **MQCHLTAB** indique le nom du fichier contenant la table de définition de canal du client (CCDT).

La variable d'environnement **MQCCDTURL** offre la capacité équivalente à la définition d'une combinaison des variables d'environnement **MQCHLLIB** et **MQCHLTAB** . **MQCCDTURL** vous permet de fournir un fichier, ftp ou une URL http sous la forme d'une valeur unique à partir de laquelle une table de définition de canal du client peut être obtenue. Pour plus d'informations, voir Accès Web adressable à la table de définitions de canaux client (CCDT).

Connexion d'applications client à des gestionnaires de files d'attente à l'aide de la structure MQCNO
Vous pouvez spécifier la définition du canal dans une structure de définition de canal (MQCD), qui est fournie à l'aide de la structure MQCNO de l'appel MQCONNX.

Pour plus d'informations, voir Création d'un canal de connexion client sur le IBM MQ MQI client à l'aide de MQCNO.

Connexion d'applications client à des gestionnaires de files d'attente à l'aide d'une table de définition de canal du client
Si vous utilisez la commande MQSC DEFINE CHANNEL, les détails que vous fournissez sont placés dans la table de définition de canal du client (ccdt). Le contenu du paramètre **QMgrName** de l'appel MQCONN ou MQCONNX détermine à quel gestionnaire de files d'attente le client se connecte.

Ce fichier est accessible par le client pour déterminer le canal qu'une application utilisera. Lorsqu'il existe plusieurs définitions de canal appropriées, le choix du canal est influencé par les attributs de pondération de canal client (CLNTWGHT) et d'affinité de connexion (AFFINITY).

Utilisation de la reconnexion automatique du client
Vous pouvez faire en sorte que vos applications client se reconnectent automatiquement, sans écrire de code supplémentaire, en configurant un certain nombre de composants.

La reconnexion automatique du client est *intégrée*. La connexion est restaurée automatiquement à tout moment dans le programme d'application client et les descripteurs permettant d'ouvrir les objets sont tous restaurés.

Par contre, la reconnexion manuelle requiert que l'application client recrée une connexion avec MQCONN ou MQCONNX et qu'elle rouvre les objets. La reconnexion automatique du client est adaptée pour de nombreuses applications client, mais pas toutes.

Pour plus d'informations, voir Reconnexion automatique du client.

Rôle de la table de définition de canal du client
La table de définition de canal du client (CCDT) contient les définitions des canaux de connexion client. Il est particulièrement utile si vos applications client doivent se connecter à un certain nombre de gestionnaires de files d'attente alternatifs.

La table de définition de canal du client est créée lorsque vous définissez un gestionnaire de files d'attente. Le même fichier peut être utilisé par plusieurs clients IBM MQ .

Une application client peut utiliser une table de définition de canal du client de différentes manières. La table de définition de canal du client peut être copiée sur l'ordinateur client. Vous pouvez copier la table de définition de canal du client dans un emplacement partagé par plusieurs clients. Vous pouvez rendre la table de définition de canal du client accessible au client en tant que fichier partagé, alors qu'elle reste sur le serveur.

La table de définition de canal du client peut être hébergée dans un emplacement central accessible via un URI, ce qui élimine la nécessité de mettre à jour individuellement la table de définition de canal du client pour chaque client déployé.

Concepts associés

Accès Web adressable à la table de définition de canal du client

Tâches associées

Accès aux définitions de canal de connexion client

Référence associée

Table de définition de canal du client

Groupes de gestionnaires de files d'attente dans CCDT
Vous pouvez définir un ensemble de connexions dans la table de définition de canal du client (CCDT) en tant que *groupe de gestionnaires de files d'attente*. Vous pouvez connecter une application à un gestionnaire de files d'attente faisant partie d'un groupe de gestionnaires de files d'attente. Pour ce faire, vous pouvez préfixer le nom du gestionnaire de files d'attente sur un appel MQCONN ou MQCONNX à l'aide d'un astérisque.

Vous pouvez choisir de définir des connexions à plusieurs serveurs pour les raisons suivantes:

- Vous souhaitez connecter un client à l'un des ensembles de gestionnaires de files d'attente en cours d'exécution, afin d'améliorer la disponibilité.
- Vous souhaitez reconnecter un client au gestionnaire de files d'attente auquel il s'est connecté pour la dernière fois, mais vous devez vous connecter à un autre gestionnaire de files d'attente en cas d'échec de la connexion.
- Vous souhaitez pouvoir relancer une connexion client à un autre gestionnaire de files d'attente en cas d'échec de la connexion, en émettant à nouveau la commande `MQCONN` dans le programme client.
- Vous souhaitez reconnecter automatiquement une connexion client à un autre gestionnaire de files d'attente en cas d'échec de la connexion, sans écrire de code client.
- Vous souhaitez reconnecter automatiquement une connexion client à une autre instance d'un gestionnaire de files d'attente multi-instance si une instance de secours prend le relais, sans écrire de code client.
- Vous souhaitez équilibrer vos connexions client entre un certain nombre de gestionnaires de files d'attente, avec un plus grand nombre de clients se connectant à certains gestionnaires de files d'attente que d'autres.
- Vous souhaitez répartir la reconnexion de nombreuses connexions client sur plusieurs gestionnaires de files d'attente et dans le temps, en cas d'échec du volume élevé de connexions.
- Vous souhaitez pouvoir déplacer vos gestionnaires de files d'attente sans changer de code d'application client.
- Vous souhaitez écrire des programmes d'application client qui n'ont pas besoin de connaître les noms de gestionnaire de files d'attente.

Il n'est pas toujours approprié de se connecter à différents gestionnaires de files d'attente. Un client transactionnel étendu ou un client Java dans WebSphere Application Server, par exemple, peut avoir besoin de se connecter à une instance de gestionnaire de files d'attente prévisible. La reconnexion client automatique n'est pas prise en charge par IBM MQ classes for Java.

Un groupe de gestionnaires de files d'attente est un ensemble de connexions défini dans la table de définition de canal du client (CCDT). L'ensemble est défini par ses membres ayant la même valeur de l'attribut **QMNAME** dans leurs définitions de canal.

Figure 97, à la page 950 est une représentation graphique d'une table de connexion client, présentant trois groupes de gestionnaires de files d'attente, deux groupes de gestionnaires de files d'attente nommés écrits dans la table de définition de canal du client en tant que **QMNAME** `(QM1)` et **QMNAME** `(QMGrp1)`, et un groupe vide ou par défaut écrit en tant que **QMNAME** `('')`.

1. Le groupe de gestionnaires de files d'attente `QM1` possède trois canaux de connexion client, qui sont connectés aux gestionnaires de files d'attente `QM1` et `QM2`. `QM1` peut être un gestionnaire de files d'attente multi-instance situé sur deux serveurs différents.
2. Le groupe de gestionnaires de files d'attente par défaut comporte six canaux de connexion client qui le connectent à tous les gestionnaires de files d'attente.
3. `QMGrp1` dispose de canaux de connexion client vers deux gestionnaires de files d'attente, `QM4` et `QM5`.

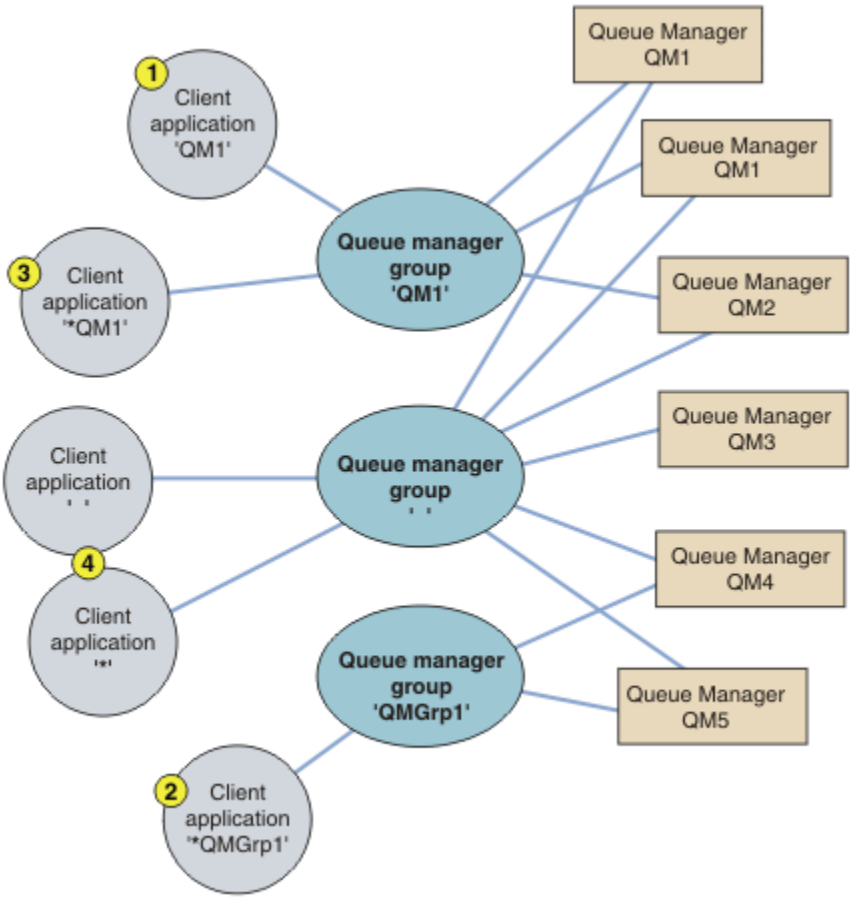


Figure 97. Groupes de gestionnaires de files d'attente

Quatre exemples d'utilisation de cette table de connexion client sont décrits à l'aide des applications client numérotées dans Figure 97, à la page 950.

1. Dans le premier exemple, l'application client transmet un nom de gestionnaire de files d'attente, `QM1`, en tant que paramètre **`QmgrName`** à son appel `MQCONN` ou `MQCONNX` MQI. Le code client IBM MQ sélectionne le groupe de gestionnaires de files d'attente correspondant, QM1. Le groupe contient trois canaux de connexion et IBM MQ MQI client tente de se connecter à `QM1` en utilisant chacun de ces canaux à tour de rôle jusqu'à ce qu'il trouve un programme d'écoute IBM MQ pour la connexion à un gestionnaire de files d'attente en cours d'exécution appelé `QM1`.

   L'ordre des tentatives de connexion dépend de la valeur de l'attribut AFFINITY de la connexion client et des pondérations du canal client. Dans ces contraintes, l'ordre des tentatives de connexion est aléatoire, à la fois sur les trois connexions possibles, et dans le temps, afin de répartir la charge de la réalisation des connexions.

   L'appel `MQCONN` ou `MQCONNX` émis par l'application client aboutit lorsqu'une connexion est établie à une instance en cours d'exécution de `QM1`.
2. Dans le deuxième exemple, l'application client transmet un nom de gestionnaire de files d'attente précédé d'un astérisque, `*QMGrp1` comme paramètre **`QmgrName`** à son appel `MQCONN` ou `MQCONNX` MQI. Le client IBM MQ sélectionne le groupe de gestionnaires de files d'attente correspondant, QMGrp1. Ce groupe contient deux canaux de connexion client et IBM MQ MQI client tente de se connecter à *tout gestionnaire de files d'attente* en utilisant chaque canal à tour de rôle. Dans cet exemple, IBM MQ MQI client doit établir une connexion réussie ; le nom du gestionnaire de files d'attente auquel il se connecte n'a pas d'importance.

   La règle pour l'ordre d'établissement des tentatives de connexion est la même que précédemment. La seule différence est qu'en ajoutant un astérisque au nom du gestionnaire de files d'attente, le client indique que le nom du gestionnaire de files d'attente n'est pas pertinent.

L'appel MQCONN ou MQCONNX émis par l'application client aboutit lorsqu'une connexion est établie à une instance en cours d'exécution d'un gestionnaire de files d'attente connecté par les canaux du groupe de gestionnaires de files d'attente QMGrp1 .

3. Le troisième exemple est essentiellement le même que le second, car le paramètre **QmgrName** est précédé d'un astérisque, *QM1. L'exemple illustre que vous ne pouvez pas déterminer à quel gestionnaire de files d'attente une connexion de canal client va se connecter en examinant l'attribut QMNAME dans une définition de canal par lui-même. Le fait que l'attribut **QMNAME** de la définition de canal soit QM1n'est pas suffisant pour exiger une connexion à un gestionnaire de files d'attente appelé QM1. Si votre application client préfixe son paramètre **QmgrName** avec un astérisque, tout gestionnaire de files d'attente est une cible de connexion possible.

   Dans ce cas, les appels MQCONN ou MQCONNX émis par l'application client aboutissent lorsqu'une connexion est établie à une instance en cours d'exécution de QM1 ou de QM2.

4. Le quatrième exemple illustre l'utilisation du groupe par défaut. Dans ce cas, l'application client transmet un astérisque, '*', ou un blanc ' ', comme paramètre **QmgrName** à son appel MQI MQCONN ou MQCONNX . Par convention dans la définition de canal du client, un attribut **QMNAME** vide indique le groupe de gestionnaires de files d'attente par défaut et un blanc ou un astérisque **QmgrName** correspond à un attribut **QMNAME** vide.

   Dans cet exemple, le groupe de gestionnaires de files d'attente par défaut possède des connexions de canal client à tous les gestionnaires de files d'attente. En sélectionnant le groupe de gestionnaires de files d'attente par défaut, l'application peut être connectée à n'importe quel gestionnaire de files d'attente du groupe.

   L'appel MQCONN ou MQCONNX émis par l'application client aboutit lorsqu'une connexion est établie à une instance en cours d'exécution d'un gestionnaire de files d'attente.

   Remarque : Le groupe par défaut est différent d'un gestionnaire de files d'attente par défaut, bien qu'une application utilise un paramètre **QmgrName** vide pour se connecter au groupe de gestionnaires de files d'attente par défaut ou au gestionnaire de files d'attente par défaut. Le concept de groupe de gestionnaires de files d'attente par défaut n'est pertinent que pour une application client et un gestionnaire de files d'attente par défaut pour une application serveur.

Définissez vos canaux de connexion client sur un seul gestionnaire de files d'attente, y compris les canaux qui se connectent à un deuxième ou à un troisième gestionnaire de files d'attente. Ne les définissez pas sur deux gestionnaires de files d'attente, puis essayez de fusionner les deux tables de définition de canal du client. Une seule table de définition de canal du client est accessible par le client.

Exemples

Examinez à nouveau la liste des raisons de l'utilisation des groupes de gestionnaires de files d'attente au début de la rubrique. Comment l'utilisation d'un groupe de gestionnaires de files d'attente offre-t-elle ces fonctions?

Connectez-vous à l'un des ensembles de gestionnaires de files d'attente.
: Définissez un groupe de gestionnaires de files d'attente avec des connexions à tous les gestionnaires de files d'attente de l'ensemble et connectez-vous au groupe à l'aide du paramètre **QmgrName** précédé d'un astérisque.

Reconnectez-vous au même gestionnaire de files d'attente, mais connectez-vous à un autre, si le gestionnaire de files d'attente connecté à la dernière fois n'est pas disponible.
: Définissez un groupe de gestionnaires de files d'attente comme précédemment, mais définissez l'attribut **AFFINITY** (PREFERRED) sur chaque définition de canal du client.

Faites une nouvelle tentative de connexion à un autre gestionnaire de files d'attente en cas d'échec d'une connexion.
: Connectez-vous à un groupe de gestionnaires de files d'attente et émettez à nouveau l'appel MQCONN ou MQCONNX MQI si la connexion est interrompue ou si le gestionnaire de files d'attente échoue.

Se reconnecter automatiquement à un autre gestionnaire de files d'attente en cas d'échec d'une connexion.

Connectez-vous à un groupe de gestionnaires de files d'attente à l'aide de l'option MQCONNX **MQCNO** MQCNO_RECONNECT.

Se reconnecte automatiquement à une autre instance d'un gestionnaire de files d'attente multi-instance.

Procédez de la même manière que dans l'exemple précédent. Dans ce cas, si vous souhaitez restreindre le groupe de gestionnaires de files d'attente pour qu'il se connecte aux instances d'un gestionnaire de files d'attente multi-instance particulier, définissez le groupe avec des connexions uniquement aux instances de gestionnaire de files d'attente multi-instance.

Vous pouvez également demander à l'application client d'émettre son appel MQCONN ou MQCONNX MQI sans astérisque comme préfixe du paramètre **QmgrName** . Ainsi, l'application client ne peut se connecter qu'au gestionnaire de files d'attente nommé. Enfin, vous pouvez définir l'option **MQCNO** sur MQCNO_RECONNECT_Q_MGR. Cette option accepte les reconnexions au même gestionnaire de files d'attente que celui précédemment connecté. Vous pouvez également utiliser cette valeur pour restreindre les reconnexions à la même instance d'un gestionnaire de files d'attente normal.

Equilibrez les connexions client entre les gestionnaires de files d'attente, avec plus de clients connectés à certains gestionnaires de files d'attente que d'autres.

Définissez un groupe de gestionnaires de files d'attente et définissez l'attribut **CLNTWGHT** sur chaque définition de canal du client pour répartir les connexions de manière inégale.

Répartissez la charge de reconnexion du client de manière inégale et répartissez cette charge dans le temps, après une défaillance de la connexion ou du gestionnaire de files d'attente.

Procédez de la même manière que dans l'exemple précédent. IBM MQ MQI client randomise les reconnexions entre les gestionnaires de files d'attente et répartit les reconnexions dans le temps.

Déplacez vos gestionnaires de files d'attente sans modifier le code client.

La table de définition de canal du client isole votre application client de l'emplacement du gestionnaire de files d'attente. La table de définition de canal du client est un fichier de données qui peut être défini sur le client, lu à partir d'un emplacement partagé ou extrait à partir d'un serveur Web. Pour plus d'informations, voir Table de définition de canal du client.

Ecrivez une application client qui ne connaît pas les noms des gestionnaires de files d'attente.

Utilisez les noms de groupe de gestionnaires de files d'attente et établissez une convention de dénomination pour les noms de groupe de gestionnaires de files d'attente qui soit pertinente pour vos applications client dans votre organisation et qui reflète l'architecture de vos solutions plutôt que la dénomination des gestionnaires de files d'attente.

z/OS *Connecting to queue sharing groups*

You can connect your application to a queue manager that is part of a queue sharing group. This can be done by using the queue sharing group name instead of the queue manager name on the MQCONN or MQCONNX call.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

The client channel definition should use the queue sharing group generic interface to connect to an available queue manager in the group. For more information, see Connecting a client to a queue sharing group. A check is made to ensure that the queue manager the listener connects to is a member of the queue sharing group.

For more information on shared queues, see Shared queues and queue sharing groups.

Exemples de pondération et d'affinité de canal

Ces exemples illustrent la façon dont les canaux de connexion client sont sélectionnés lorsque des ClientChannelPoids différents de zéro sont utilisés.

Les attributs de canal ClientChannelWeight et ConnectionAffinity contrôlent la manière dont les canaux de connexion client sont sélectionnés lorsque plusieurs canaux appropriés sont disponibles pour une connexion. Ces canaux sont configurés pour se connecter à différents gestionnaires de files d'attente

afin de fournir une plus grande disponibilité, un équilibrage de charge ou les deux. Les appels MQCONN qui peuvent aboutir à une connexion à l'un des gestionnaires de files d'attente doivent préfixer le nom du gestionnaire de files d'attente avec un astérisque, comme décrit dans: Exemples d'appels MQCONN: Exemple 1. Le nom du gestionnaire de files d'attente inclut un astérisque (*).

Les canaux candidats applicables pour une connexion sont ceux où l'attribut QMNAME correspond au nom de gestionnaire de files d'attente spécifié dans l'appel MQCONN. Si tous les canaux applicables pour une connexion ont un `ClientChannelWeight` égal à zéro (valeur par défaut), ils sont sélectionnés dans l'ordre alphabétique comme dans l'exemple suivant: Exemples d'appels MQCONN: Exemple 1. Le nom du gestionnaire de files d'attente inclut un astérisque (*).

Les exemples suivants illustrent ce qui se passe lorsque des pondérations `ClientChannel` différentes de zéro sont utilisées. Notez que, puisque cette fonction implique une sélection de canal pseudo-aléatoire, les exemples montrent une séquence d'actions qui peuvent se produire plutôt que ce qui sera définitivement.

Exemple 1. Sélection de canaux lorsque ConnectionAffinity est défini sur PREFERRED
Cet exemple illustre la façon dont un IBM MQ MQI client sélectionne un canal à partir d'une table de définition de canal du client, où ConnectionAffinity est défini sur PREFERRED.

Dans cet exemple, un certain nombre de machines client utilisent une table de définition de canal du client (CCDT) fournie par un gestionnaire de files d'attente. La table de définition de canal du client inclut des canaux de connexion client avec les attributs suivants (affichés à l'aide de la syntaxe de la commande 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)
```

L'application émet MQCONN (*CORE)

Le canal A n'est pas candidat pour cette connexion, car l'attribut QMNAME ne correspond pas. Les canaux B, C et D sont identifiés comme candidats, et sont placés dans un ordre de préférence en fonction de leur pondération. Dans cet exemple, l'ordre peut être C, B, D. Le client tente de se connecter au gestionnaire de files d'attente à l'adresse core2.ops.company.example. Le nom du gestionnaire de files d'attente à cette adresse n'est pas vérifié car l'appel MQCONN incluait un astérisque dans le nom du gestionnaire de files d'attente.

Il est important de noter que, avec `AFFINITY(PREFERRED)`, chaque fois que cette machine client particulière se connecte, elle place les canaux dans le même ordre de préférence initial. Cela s'applique même lorsque les connexions proviennent de processus différents ou à des moments différents.

Dans cet exemple, le gestionnaire de files d'attente à l'adresse core.2.ops.company.example est inaccessible. Le client tente de se connecter à core1.ops.company.example car le canal B est le suivant dans l'ordre de préférence. En outre, le canal C est rétrogradé pour devenir le moins préféré.

Un deuxième appel MQCONN (*CORE) est émis par la même application. Le canal C a été rétrogradé par la connexion précédente, de sorte que le canal le plus préféré est maintenant B. Cette connexion est établie à core1.ops.company.example.

Une deuxième machine partageant la même table de définition de canal du client place les canaux dans un ordre de préférence initial différent. Par exemple, D, B, C. Dans des circonstances normales, avec tous les canaux fonctionnant, les applications de cette machine sont connectées à core3.ops.company.example tandis que celles de la première machine sont connectées à core2.ops.company.example. Cela permet d'équilibrer la charge de travail d'un grand nombre de clients sur plusieurs gestionnaires de files d'attente tout en permettant à chaque client individuel de se connecter au même gestionnaire de files d'attente s'il est disponible.

Exemple 2. Sélection de canaux lorsque ConnectionAffinity est défini sur NONE
Cet exemple illustre la façon dont un IBM MQ MQI client sélectionne un canal à partir d'une table de définition de canal du client (CCDT), où ConnectionAffinity est défini sur NONE.

Dans cet exemple, un certain nombre de clients utilisent une table de définition de canal du client (CCDT) fournie par un gestionnaire de files d'attente. La table de définition de canal du client inclut des canaux de connexion client avec les attributs suivants (affichés à l'aide de la syntaxe de la commande 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)
```

L'application émet MQCONN (*CORE). Comme dans l'exemple précédent, le canal A n'est pas pris en compte car QMNAME ne correspond pas. Les canaux B, C ou D sont sélectionnés en fonction de leur pondération, avec des probabilités de 50%, 30% ou 20%. Dans cet exemple, le canal B peut être sélectionné. Aucun ordre de préférence persistant n'a été créé.

Un deuxième appel MQCONN (*CORE) est effectué. Là encore, l'un des trois canaux applicables est sélectionné, avec les mêmes probabilités. Dans cet example, le canal C est choisi. Cependant, core2.ops.company.example ne répond pas, un autre choix est donc effectué entre les canaux candidats restants. Le canal B est sélectionné et l'application est connectée à core1.ops.company.example.

Avec AFFINITY (NONE), chaque appel MQCONN est indépendant des autres. Par conséquent, lorsque cet exemple d'application crée un troisième MQCONN (*CORE), il peut tenter une fois de plus de se connecter via le canal C rompu, avant de choisir l'un des suivants: B ou D.

Exemples d'appels MQCONN
Exemples d'utilisation de MQCONN pour la connexion à un gestionnaire de files d'attente spécifique ou à l'un des groupes de gestionnaires de files d'attente.

Dans chacun des exemples suivants, le réseau est le même ; une connexion est définie à deux serveurs à partir du même IBM MQ MQI client. (Dans ces exemples, l'appel MQCONNX peut être utilisé à la place de l'appel MQCONN.)

Deux gestionnaires de files d'attente sont en cours d'exécution sur les machines serveur, l'un nommé SALE et l'autre nommé SALE_BACKUP.

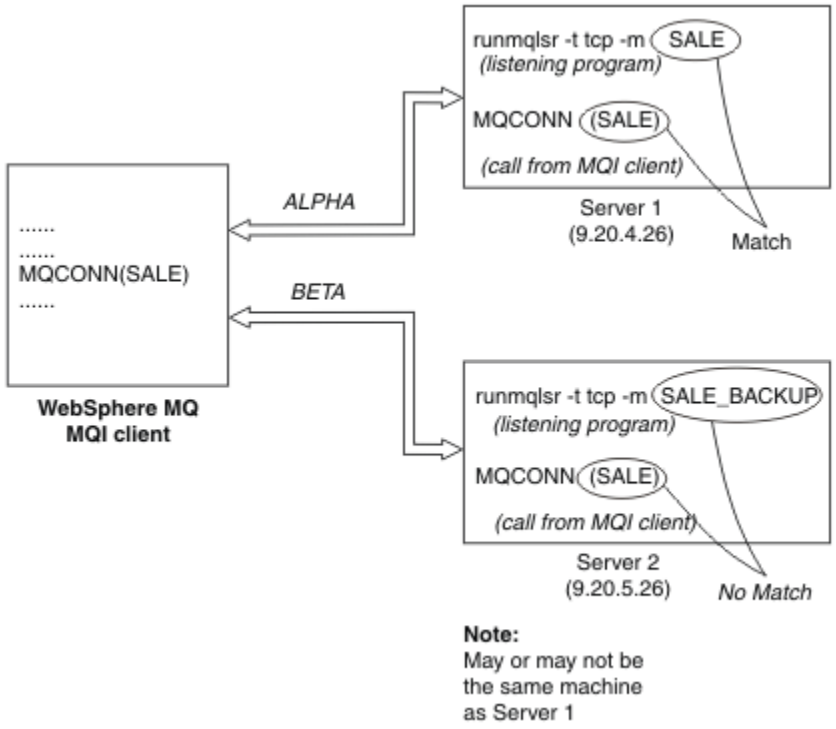


Figure 98. Exemple MQCONN

Les définitions des canaux dans ces exemples sont les suivantes:

Définitions de vente:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('IBM MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('IBM MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Définition SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to IBM MQ MQI client')
```

Les définitions de canal du client peuvent être résumées comme suit:

Nom	CHLTYPE	TRPTYPE	CONNAME	QMNAME
ALPHA	CLNTCONN	TCP	9.20.4.26	Vente
BETA	CLNTCONN	TCP	9.20.5.26	Vente

Exemples MQCONN
Les exemples illustrent l'utilisation de plusieurs gestionnaires de files d'attente comme système de sauvegarde.

Supposons que la liaison de communication avec le serveur 1 soit temporairement interrompue. L'utilisation de plusieurs gestionnaires de files d'attente en tant que système de sauvegarde est illustrée.

Chaque exemple couvre un appel MQCONN différent et explique ce qui se passe dans l'exemple spécifique présenté, en appliquant les règles suivantes:

1. La table de définition de canal du client (CCDT) est analysée dans l'ordre alphabétique des noms de canal pour un nom de gestionnaire de files d'attente (zone QMNAME) correspondant à celui indiqué dans l'appel MQCONN.
2. Si une correspondance est trouvée, la définition de canal est utilisée.
3. Tentative de démarrage du canal vers la machine identifiée par le nom de connexion (CONNAME). Si cette opération aboutit, l'application se poursuit. Il requiert:
   - Un programme d'écoute à exécuter sur le serveur.
   - Programme d'écoute à connecter au même gestionnaire de files d'attente que celui auquel le client souhaite se connecter (si spécifié).
4. Si la tentative de démarrage du canal échoue et qu'il y a plus d'une entrée dans la table de définition de canal du client (dans cet exemple, il y a deux entrées), la recherche d'une autre correspondance est effectuée dans le fichier. Si une correspondance est trouvée, le traitement se poursuit à l'étape 1.
5. Si aucune correspondance n'est trouvée ou qu'il n'y a plus d'entrées dans la table de définition de canal du client et que le canal n'a pas pu démarrer, l'application ne peut pas se connecter. Un code anomalie et un code achèvement appropriés sont renvoyés dans l'appel MQCONN. L'application peut prendre des mesures en fonction du motif et des codes d'achèvement renvoyés.

Exemple 1. Le nom du gestionnaire de files d'attente inclut un astérisque (*)
Dans cet exemple, l'application ne se préoccupe pas du gestionnaire de files d'attente auquel elle se connecte. L'application émet un appel MQCONN pour un nom de gestionnaire de files d'attente incluant un astérisque. Un canal approprié est choisi.

Les problèmes de l'application sont les suivants:

```
MQCONN (*SALE)
```

En suivant les règles, voici ce qui se passe dans cette instance:

1. La table de définition de canal du client (CCDT) est analysée pour le nom de gestionnaire de files d'attente SALE, qui correspond à l'appel MQCONN de l'application.
2. Des définitions de canal ont été trouvées pour ALPHA et BETA .
3. Si un canal a une valeur CLNTWGHT de 0, ce canal est sélectionné. Si les deux ont une valeur CLNTWGHT de 0, le canal ALPHA est sélectionné car il est le premier dans l'ordre alphabétique. Si les deux canaux ont une valeur CLNTWGHT non nulle, un canal est sélectionné aléatoirement, en fonction de sa pondération.
4. Une tentative de démarrage du canal est effectuée.
5. Si le canal BETA a été sélectionné, la tentative de démarrage a abouti.
6. Si le canal ALPHA a été sélectionné, la tentative de démarrage n'aboutit PAS car la liaison de communication est rompue. Les étapes suivantes s'appliquent ensuite:
   a. Le seul autre canal pour le nom de gestionnaire de files d'attente SALE est BETA.
   b. Une tentative de démarrage de ce canal a été effectuée-cette opération a abouti.
7. Une vérification indiquant qu'un programme d'écoute est en cours d'exécution indique qu'un programme d'écoute est en cours d'exécution. Il n'est pas connecté au gestionnaire de files d'attente SALE , mais comme le paramètre d'appel MQI comporte un astérisque (*), aucune vérification n'est effectuée. L'application est connectée au gestionnaire de files d'attente SALE_BACKUP et poursuit le traitement.

Exemple 2. Nom de gestionnaire de files d'attente spécifié
Dans cet exemple, l'application doit se connecter à un gestionnaire de files d'attente particulier. L'application émet un appel MQCONN pour ce nom de gestionnaire de files d'attente. Un canal approprié est choisi.

L'application requiert une connexion à un gestionnaire de files d'attente spécifique, nommé SALE, comme indiqué dans l'appel MQI:

```
MQCONN (SALE)
```

En suivant les règles, voici ce qui se passe dans cette instance:

1. La table de définition de canal du client (CCDT) est analysée dans la séquence alphabétique des noms de canal, pour le nom de gestionnaire de files d'attente SALE, correspondant à l'appel MQCONN de l'application.
2. La première définition de canal trouvée est ALPHA.
3. Une tentative de démarrage du canal a été effectuée. Cette opération a échoué car la liaison de communication est interrompue.
4. La table de définition de canal du client est à nouveau analysée pour le nom de gestionnaire de files d'attente SALE et le nom de canal BETA est trouvé.
5. Une tentative de démarrage du canal a été effectuée. Cette opération a abouti.
6. Une vérification indiquant qu'un programme d'écoute est en cours d'exécution indique qu'un programme d'écoute est en cours d'exécution, mais qu'il n'est pas connecté au gestionnaire de files d'attente SALE .
7. Il n'y a pas d'autres entrées dans la table de définition de canal du client. L'application ne peut pas continuer et reçoit le code retour MQRC_Q_MGR_NOT_AVAILABLE.

Exemple 3. Le nom du gestionnaire de files d'attente est vide ou un astérisque (*)
Dans cet exemple, l'application ne se préoccupe pas du gestionnaire de files d'attente auquel elle se connecte. L'application émet un MQCONN spécifiant un nom de gestionnaire de files d'attente vide ou un astérisque. Un canal approprié est choisi.

Cette opération est traitée de la même manière que «Exemple 1. Le nom du gestionnaire de files d'attente inclut un astérisque (*)», à la page 956.

Remarque : Si cette application s'exécute dans un environnement autre qu'un environnement IBM MQ MQI clientet que le nom est vide, elle tente de se connecter au gestionnaire de files d'attente par défaut. Ce n'est pas le cas lorsqu'il est exécuté à partir d'un environnement client ; le gestionnaire de files d'attente utilisé est celui associé au programme d'écoute auquel le canal se connecte.

Les problèmes de l'application sont les suivants:

```
MQCONN ("")
```

ou

```
MQCONN (*)
```

En suivant les règles, voici ce qui se passe dans cette instance:

1. La table de définition de canal du client (CCDT) est analysée dans la séquence de noms de canal alphabétique, à la recherche d'un nom de gestionnaire de files d'attente vide, correspondant à l'appel MQCONN de l'application.
2. L'entrée du nom de canal ALPHA a un nom de gestionnaire de files d'attente dans la définition de SALE. Cela ne correspond pas au paramètre d'appel MQCONN, qui requiert que le nom du gestionnaire de files d'attente soit vide.
3. L'entrée suivante concerne le nom de canal BETA.
4. `queue manager name` dans la définition est SALE. Une fois de plus, cela ne correspond pas au paramètre d'appel MQCONN, qui requiert que le nom du gestionnaire de files d'attente soit vide.
5. Il n'y a pas d'autres entrées dans la table de définition de canal du client. L'application ne peut pas continuer et reçoit le code retour MQRC_Q_MGR_NOT_AVAILABLE.

Déclenchement dans l'environnement client

Les messages envoyés par les applications IBM MQ s'exécutant sous IBM MQ MQI clients contribuent au déclenchement exactement de la même manière que les autres messages et peuvent être utilisés pour déclencher des programmes sur le serveur et le client.

Le déclenchement est expliqué en détail dans la rubrique Canaux de déclenchement.

Le moniteur de déclenchement et l'application à démarrer doivent se trouver sur le même système.

Les caractéristiques par défaut de la file d'attente de déclenchement sont les mêmes que celles de l'environnement serveur. En particulier, si aucune option de contrôle de point de synchronisation MQPMO n'est spécifiée dans une application client qui place des messages dans une file d'attente déclenchée locale d'un gestionnaire de files d'attente z/OS , les messages sont insérés dans une unité d'oeuvre. Si la condition de déclenchement est alors remplie, le message de déclenchement est inséré dans la file d'attente d'initialisation au sein de la même unité d'oeuvre et ne peut pas être extrait par le moniteur de déclenchement tant que l'unité d'oeuvre n'est pas terminée. Le processus à déclencher n'est pas lancé tant que l'unité de travail n'est pas arrêtée.

Définition de processus

Vous devez définir la définition de processus sur le serveur, car elle est associée à la file d'attente sur laquelle le déclenchement est défini.

L'objet de processus définit ce qui doit être déclenché. Si le client et le serveur ne sont pas en cours d'exécution sur la même plateforme, tous les processus démarrés par le moniteur de déclenchement doivent définir *ApplType*, sinon le serveur prend ses définitions par défaut (c'est-à-dire le type d'application normalement associé à la machine du serveur) et provoque un échec.

Par exemple, si le moniteur de déclenchement s'exécute sur un IBM MQ MQI client et souhaite envoyer une demande à un serveur sur un autre système d'exploitation, MQAT_WINDOWS_NT doit être défini, sinon l'autre système d'exploitation utilise ses définitions par défaut et le processus échoue.

Multi *moniteur de déclenchement*

Le moniteur de déclenchement fourni par IBM MQ for Multiplatforms s'exécute dans les environnements client pour les systèmes Multiplatforms.

Pour exécuter le moniteur de déclenchement, exécutez l'une des commandes suivantes:

- IBM i Sous IBM i :

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ALW Sur les plateformes AIX, Linux, and Windows :

```
runmqtmc [-m QMgrName] [-q InitQ]
```

La file d'attente d'initialisation par défaut est SYSTEM.DEFAULT.INITIATION.QUEUE sur le gestionnaire de files d'attente par défaut. La file d'attente d'initialisation est l'endroit où le moniteur de déclenchement recherche les messages de déclenchement. Il appelle ensuite les programmes pour les messages de déclenchement appropriés. Ce moniteur de déclenchement prend en charge le type d'application par défaut et est identique à `runmqtrm` , sauf qu'il lie les bibliothèques client.

La chaîne de commande, générée par le moniteur de déclenchement, est la suivante:

1. Le *ApplicId* de la définition de processus appropriée. *ApplicId* est le nom du programme à exécuter, tel qu'il est entré sur la ligne de commande.
2. La structure MQTMC2 , placée entre guillemets, obtenue à partir de la file d'attente d'initialisation. Une chaîne de commande est démarrée avec cette chaîne, exactement comme elle est fournie, entre guillemets afin que la commande système l'accepte comme un paramètre.
3. Le *EnvrData* de la définition de processus appropriée.

Le moniteur de déclenchement ne recherche pas s'il existe un autre message dans la file d'attente d'initialisation tant que l'application qu'il a démarrée n'est pas terminée. Si l'application a beaucoup de traitement à effectuer, le moniteur de déclenchement risque de ne pas suivre le nombre de messages de déclenchement qui arrivent. Il existe deux façons de faire face à cette situation:

1. Avoir plus de moniteurs de déclenchement en cours d'exécution

   Si vous choisissez d'exécuter plus de moniteurs de déclenchement, vous pouvez contrôler le nombre maximal d'applications pouvant être exécutées simultanément.

2. Exécuter les applications démarrées en arrière-plan

   Si vous choisissez d'exécuter des applications en arrière-plan, IBM MQ n'impose aucune restriction sur le nombre d'applications pouvant être exécutées.

Pour exécuter l'application démarrée en arrière-plan sur les systèmes AIX and Linux , vous devez placer une & (perluète) à la fin du *EnvrData* de la définition de processus.

Applications CICS (nonz/OS)
Un programme d'application nonz/OS CICS qui émet un appel `MQCONN` ou `MQCONNX` doit être défini sur CEDA en tant que RErésidente. Si vous reliez une application serveur CICS en tant que client, vous risquez de perdre la prise en charge des points de synchronisation.

Un programme d'application nonz/OS CICS qui émet un appel `MQCONN` ou `MQCONNX` doit être défini sur CEDA en tant que RErésidente. Pour rendre le code résident aussi petit que possible, vous pouvez créer un lien vers un programme distinct pour émettre l'appel `MQCONN` ou `MQCONNX` .

Si la variable d'environnement MQSERVER est utilisée pour définir la connexion client, elle doit être spécifiée dans le fichier `CICSENV.CMD` .

Les applications IBM MQ peuvent être exécutées dans un environnement de serveur IBM MQ ou sur un client IBM MQ sans changer de code. Toutefois, dans un environnement de serveur IBM MQ , CICS peut agir en tant que coordinateur de point de synchronisation et vous utilisez EXEC CICS SYNCPOINT et EXEC CICS SYNCPOINT ROLLBACK plutôt que **MQCMIT** et **MQBACK**. Si une application CICS est simplement relayée en tant que client, la prise en charge des points de synchronisation est perdue. **MQCMIT** et **MQBACK** doivent être utilisés pour l'application qui s'exécute sur un IBM MQ MQI client.

ALW Préparation et exécution des applications CICS et Tuxedo

Pour exécuter les applications CICS et Tuxedo en tant qu'applications client, vous utilisez des bibliothèques différentes de celles que vous utilisez avec les applications serveur. L'ID utilisateur sous lequel l'application s'exécute est également différent.

Pour préparer les applications CICS et Tuxedo à s'exécuter en tant qu'applications IBM MQ MQI client , suivez les instructions de la rubrique Configuration d'un client transactionnel étendu.

Notez toutefois que les informations traitant spécifiquement de la préparation des applications CICS et Tuxedo, y compris les exemples de programme fournis avec IBM MQ, supposent que vous préparez des applications à exécuter sur un système serveur IBM MQ . Par conséquent, les informations font uniquement référence aux bibliothèques IBM MQ destinées à être utilisées sur un système serveur. Lorsque vous préparez vos applications client, vous devez effectuer les opérations suivantes:

- Utilisez la bibliothèque système client appropriée pour les liaisons de langage utilisées par votre application. Exemple :
  - Linux AIX Pour les applications écrites en C sous AIX and Linux, utilisez la bibliothèque libmqic à la place de libmqm.
  - Windows Sur les systèmes Windows , utilisez la bibliothèque mqic.lib à la place de mqm.lib.
- Au lieu des bibliothèques système du serveur indiquées dans les Tableau 134, à la page 960 et Tableau 135, à la page 960, utilisez les bibliothèques système du client équivalentes. Si une bibliothèque système de serveur n'est pas répertoriée dans ces tables, utilisez la même bibliothèque sur un système client.

Tableau 134. Bibliothèques système client sous AIX and Linux	
Bibliothèque pour un système de serveur IBM MQ	**Bibliothèque équivalente à utiliser sur un système client IBM MQ**
libmqmxa	libmqcxa
V 9.4.0 V 9.4.0 libmqmxa64	V 9.4.0 V 9.4.0 libmqcxa64

Tableau 135. Bibliothèques système client sur les systèmes Windows	
Bibliothèque pour un système de serveur IBM MQ	**Bibliothèque équivalente à utiliser sur un système client IBM MQ**
mqmxa.lib	mqcxa.lib
mqmtux.lib	mqcxa.lib
mqmenc.lib	mqcxa.lib
mqmcics4.lib	mqccics4.lib

ID utilisateur utilisé par une application client

Lorsque vous exécutez une application serveur IBM MQ sous CICS, elle passe normalement de l'utilisateur CICS à l'ID utilisateur de la transaction. Toutefois, lorsque vous exécutez une application IBM MQ MQI client sous CICS, elle conserve les droits d'accès privilégiés CICS .

ALW *Exemples de programmes CICS et Tuxedo*

Exemples de programmes CICS et Tuxedo à utiliser sur les systèmes AIX, Linux, and Windows .

Le Tableau 136, à la page 960 répertorie les exemples de programmes CICS et Tuxedo qui sont fournis pour être utilisés sur les systèmes client AIX and Linux . Le Tableau 137, à la page 961 répertorie les informations équivalentes pour les systèmes client Windows . Les tableaux répertorient également les fichiers utilisés pour la préparation et l'exécution des programmes. Pour obtenir une description des exemples de programme, voir «Exemple de transaction CICS», à la page 1111 et «Utilisation des exemples TUXEDO sur AIX, Linux, and Windows», à la page 1157.

Tableau 136. Exemples de programmes pour les systèmes client AIX and Linux		
Description	**Source**	**Module exécutable**
CICS programme	amqscic0.ccs	amqscicc
Fichier d'en-tête du programme CICS	amqscih0.h	-
Programme client Tuxedo pour insérer des messages	amqstxpx.c	-
Programme client Tuxedo pour obtenir des messages	amqstxgx.c	-
Programme serveur Tuxedo pour les deux programmes client	amqstxsx.c	-
Fichier UBBCONFIG pour les programmes Tuxedo	ubbstxcx.cfg	-
Fichier de table de zones pour les programmes Tuxedo	amqstxvx.flds	-
Afficher le fichier de description des programmes Tuxedo	amqstxvx.v	-

Tableau 137. Exemples de programmes pour les systèmes client Windows		
Description	**Source**	**Module exécutable**
CICS transaction	amqscic0.ccs	amqscicc
Fichier d'en-tête de la transaction CICS	amqscih0.h	-
Programme client Tuxedo pour insérer des messages	amqstxpx.c	-
Programme client Tuxedo pour obtenir des messages	amqstxgx.c	-
Programme serveur Tuxedo pour les deux programmes client	amqstxsx.c	-
Fichier UBBCONFIG pour les programmes Tuxedo	ubbstxcx.cfg	-
Fichier de table de zones pour les programmes Tuxedo	amqstxvx.fld	-
Afficher le fichier de description des programmes Tuxedo	amqstxvx.v	-
Fichier makefile pour les programmes Tuxedo	amqstxmc.mak	-
Fichier ENVFILE pour les programmes Tuxedo	amqstxen.env	-

ALW *Message d'erreur AMQ5203, tel que modifié pour les applications CICS et Tuxedo*

Lorsque vous exécutez des applications CICS ou Tuxedo qui utilisent un client transactionnel étendu, vous pouvez voir des messages de diagnostic standard. L'un de ces éléments a été modifié pour être utilisé avec un client transactionnel étendu

Les messages que vous pouvez voir dans les fichiers journaux d'erreurs IBM MQ sont documentés dans Messages de diagnostic: AMQ4000-9999. Le message AMQ5203 a été modifié pour être utilisé avec un client transactionnel étendu. Voici le texte du message modifié:

AMQ5203: Une erreur s'est produite lors de l'appel de l'interface XA.

Explication

Le numéro d'erreur est & 2, où la valeur 1 indique que la valeur d'indicateurs fournie, & 1, n'était pas valide, 2 indique qu'une tentative d'utilisation de bibliothèques à unités d'exécution et non à unités d'exécution a été effectuée dans le même processus, 3 indique qu'une erreur s'est produite avec le nom de gestionnaire de files d'attente fourni'& 3', 4 indique que l'ID de gestionnaire de ressources, & 1, n'était pas valide, 5 indique qu'une tentative d'utilisation d'un second gestionnaire de files d'attente appelé'& 3'lorsqu'un autre gestionnaire de files d'attente était déjà connecté, 6 indique que le gestionnaire de transactions a été appelé lorsque l'application n'est pas connectée à un gestionnaire de files d'attente, 7 indique que l'appel XA a été effectué alors qu'un autre appel était en cours, 8 indique que la chaîne xa_info'& 4'de l'appel xa_open contenait une valeur de paramètre non valide pour le nom de paramètre'& 5', et 9 indique que la chaîne xa_info'& 4'de l'appel xa_open ne contient pas un paramètre obligatoire, le nom de paramètre'& 5'.

Intervention de l'utilisateur

Corrigez l'erreur et renouvelez l'opération.

Windows Préparation et exécution des applications Microsoft Transaction Server

Pour préparer une application MTS à s'exécuter en tant qu'application IBM MQ MQI client , suivez ces instructions en fonction de votre environnement.

Pour des informations générales sur le développement d'applications Microsoft Transaction Server (MTS) qui accèdent aux ressources IBM MQ , voir la section relative à MTS dans le centre d'aide IBM MQ .

Pour préparer une application MTS à s'exécuter en tant qu'application IBM MQ MQI client , effectuez l'une des opérations suivantes pour chaque composant de l'application:

- Si le composant utilise les liaisons de langage C pour l'interface MQI, suivez les instructions de la rubrique «Préparation des programmes C dans Windows», à la page 1044 mais liez le composant à la bibliothèque mqicxa.lib à la place de mqic.lib.
- Si le composant utilise les classes C++ IBM MQ , suivez les instructions de la rubrique «Génération de programmes C++ sur Windows», à la page 563 mais liez le composant à la bibliothèque imqx23vn.lib à la place de imqc23vn.lib.
- Si le composant utilise les liaisons de langage Visual Basic pour l'interface MQI, suivez les instructions de la rubrique «Préparation des programmes Visual Basic dans Windows», à la page 1047 , mais lorsque vous définissez le projet Visual Basic, entrez `MqType=3` dans la zone **Arguments de compilation conditionnels** .

Préparation et exécution des applications IBM MQJMS

Vous pouvez exécuter des applications IBM MQ JMS en mode client, avec WebSphere Application Server comme gestionnaire de transactions. Certains messages d'avertissement peuvent s'afficher.

Pour préparer et exécuter des applications IBM MQ JMS en mode client, avec WebSphere Application Server comme gestionnaire de transactions, suivez les instructions de la rubrique «Utilisation de IBM MQ classes for JMS/Jakarta Messaging», à la page 85.

Lorsque vous exécutez une application client IBM MQ JMS , les messages d'avertissement suivants peuvent s'afficher:

MQJE080
Unités de licence insuffisantes-exécutez setmqcap

MQJE081
Le fichier contenant les informations d'unité de licence est dans un format incorrect-exécutez setmqcap

MQJE082
Fichier contenant les informations d'unité de licence introuvable-exécutez setmqcap

Exits utilisateur, exits API et services optionnels d'IBM MQ

Cette rubrique contient des liens vers des informations sur l'utilisation et le développement de ces programmes.

Pour savoir comment utiliser les exits utilisateur, les exits API et les services installables pour étendre les fonctions du gestionnaire de files d'attente, voir Extension des fonctions du gestionnaire de files d'attente.

Pour plus d'informations sur l'écriture et la compilation des exits et des services installables, voir les sous-rubriques.

Concepts associés
Programmes d'exit de canal pour les canaux MQI
Référence associée
Référence d'exit API
Informations de référence de l'interface des services installables
IBM i Informations de référence de l'interface des services installables sur IBM i

ALW Ecriture d'exits et de services installables sous AIX, Linux, and Windows

Vous pouvez écrire et compiler des exits sans liaison à des bibliothèques IBM MQ sur AIX, Linux, and Windows.

Pourquoi et quand exécuter cette tâche

Cette rubrique s'applique uniquement aux systèmes AIX, Linux, and Windows . Pour plus de détails sur l'écriture des exits et des services installables pour d'autres plateformes, voir les rubriques spécifiques à la plateforme.

Si IBM MQ est installé dans un emplacement autre que celui par défaut, vous devez écrire et compiler vos exits sans lien vers des bibliothèques IBM MQ .

Vous pouvez écrire et compiler des exits sur des systèmes AIX, Linux, and Windows sans lier les bibliothèques IBM MQ suivantes:

- mqmzf
- mqm
- mqmvx
- mqmvxd
- mqic
- mqutl

Les exits existants qui sont liés à ces bibliothèques continuent de fonctionner, à condition que sur les systèmes AIX and Linux , IBM MQ soit installé à l'emplacement par défaut.

Procédure

1. Incluez le fichier d'en-tête cmqec.h .

   L'inclusion de ce fichier d'en-tête inclut automatiquement les fichiers d'en-tête cmqc.h, cmqxc.h et cmqzc.h .
2. Ecrivez l'exit de sorte que les appels MQI et DCI soient effectués via la structure MQIEP. Pour plus d'informations sur la structure MQIEP, voir Structure MQIEP.
   - Services optionnels
     - Utilisez le paramètre **Hconfig** pour pointer vers l'appel MQZEP.
     - Vous devez vérifier que les 4 premiers octets de **Hconfig** correspondent au **StrucId** de la structure MQIEP avant d'utiliser le paramètre **Hconfig** .
     - Pour plus d'informations sur l'écriture des composants de service installables, voir MQIEP.
   - Exits API
     - Utilisez le paramètre **Hconfig** pour pointer vers l'appel MQXEP.
     - Vous devez vérifier que les 4 premiers octets de **Hconfig** correspondent au **StrucId** de la structure MQIEP avant d'utiliser le paramètre **Hconfig** .
     - Pour plus d'informations sur l'écriture des exits d'API, voir «Ecriture des exits API», à la page 981.
   - Exits de canal
     - Utilisez le paramètre **pEntryPoints** de la structure MQCXP pour pointer vers les appels MQI et DCI.
     - Vous devez vérifier que le numéro de version de MQCXP correspond à la version 8 ou à une version ultérieure avant d'utiliser **pEntryPoints**.
     - Pour plus d'informations sur l'écriture des exits de canal, voir «Ecriture de programmes d'exit de canal», à la page 991.
   - Exits de conversion de données
     - Utilisez le paramètre **pEntryPoints** de la structure MQDXP pour pointer vers les appels MQI et DCI.
     - Vous devez vérifier que le numéro de version de MQDXP est à la version 2 ou supérieure avant d'utiliser **pEntryPoints**.

- Vous pouvez utiliser la commande **crtmqcvx** et le fichier source amqsvfc0.c pour créer un code de conversion de données qui utilise le paramètre **pEntryPoints** . Voir «Ecriture d'un exit de conversion de données pour IBM MQ for Windows», à la page 1018 et «Ecriture d'un exit de conversion de données pour les systèmes IBM MQ for AIX or Linux», à la page 1015.
- Si vous disposez d'exits de conversion de données existants qui ont été générés à l'aide de la commande **crtmqcvx** , vous devez régénérer l'exit à l'aide de la commande mise à jour.
- Pour plus d'informations sur l'écriture des exits de conversion de données, voir «Ecriture des exits de conversion de données», à la page 1010.

- Exits de préconnexion
  - Utilisez le paramètre **pEntryPoints** de la structure MQNXP pour pointer vers les appels MQI et DCI.
  - Vous devez vérifier que le numéro de version de MQNXP est à la version 2 ou supérieure avant d'utiliser **pEntryPoints**.
  - Pour plus d'informations sur l'écriture des exits de préconnexion, voir «Référencement des définitions de connexion à l'aide d'un exit de préconnexion à partir d'un référentiel», à la page 1020.
- Exits de publication
  - Utilisez le paramètre **pEntryPoints** de la structure MQPSXP pour pointer vers les appels MQI et DCI.
  - Vous devez vérifier que le numéro de version de MQPSXP est à la version 2 ou supérieure avant d'utiliser **pEntryPoints**.
  - Pour plus d'informations sur l'écriture des exits de publication, voir «Ecriture et compilation des exits de publication», à la page 1022.
- Exits de charge de travail de cluster
  - Utilisez le paramètre **pEntryPoints** de la structure MQWXP pour pointer vers les appels MQXCLWLN.
  - Vous devez vérifier que le numéro de version de MQWXP est la version 4 ou une version ultérieure avant d'utiliser **pEntryPoints**.
  - Pour plus d'informations sur l'écriture des exits de charge de travail de cluster, voir «Ecriture et compilation des exits de charge de travail de cluster», à la page 1024.

Par exemple, dans un exit de canal appelant MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Vous trouverez d'autres exemples dans le «Utilisation des exemples de programmes procéduraux IBM MQ», à la page 1085.

3. Compilez l'exit:
   - N'établissez pas de lien vers les bibliothèques IBM MQ .
   - N'incluez pas de chemin RPath intégré à des bibliothèques IBM MQ dans votre exit.
   - Pour plus d'informations sur la compilation de votre exit, voir l'une des rubriques suivantes:
     - Exits API: «Compilation des exits API», à la page 983.
     - Exits de canal, exits de publication, exits de charge de travail de cluster: «Compilation des programmes d'exit de canal sur les systèmes AIX, Linux, and Windows», à la page 1009.
     - Exits de conversion de données: «Ecriture des exits de conversion de données», à la page 1010.

4. Placez l'exit dans l'un des emplacements suivants:
   - Chemin de votre choix que vous qualifiez complètement lors de la configuration de l'exit
   - Chemin d'exit par défaut, dans un répertoire d'installation spécifique. Par exemple, *MQ_DATA_PATH*/exits/installation2.
   - Chemin d'exit par défaut

     Le chemin d'exit par défaut est *MQ_DATA_PATH*/exits pour les exits 32 bits et *MQ_DATA_PATH*/exits64 pour les exits 64 bits. Vous pouvez modifier ces chemins dans le fichier qm.ini ou mqclient.ini . Pour plus d'informations, voir Chemin d'exit. Sous Windows et Linux, vous pouvez utiliser l'explorateur IBM MQ pour modifier le chemin d'accès:

     a. Cliquez avec le bouton droit de la souris sur le nom du gestionnaire
     b. Cliquez sur **Propriétés ...**
     c. Cliquez sur **Exits**
     d. Dans la zone du chemin d'accès par défaut des exits, indiquez le nom de chemin du répertoire contenant le programme d'exit.

   Si un exit est placé à la fois dans un répertoire d'installation spécifique et dans le répertoire de chemin par défaut, l'exit de répertoire d'installation spécifique est utilisé par l'installation de IBM MQ nommé dans le chemin. Par exemple, l'exit est placé dans /exits/installation2 et dans /exits, mais pas dans /exits/installation1. L' IBM MQ installation installation2 utilise l'exit de /exits/installation2. L' IBM MQ installation installation1 utilise l'exit du répertoire /exits .
5. Si nécessaire, configurez l'exit:
   - Services installables: «Configuration des services et des composants», à la page 974.
   - Exits API: «Configuration des exits API», à la page 985.
   - Exits de canal: «Configuration des exits de canal», à la page 1010.
   - Exits de publication: «Configuration des exits de publication», à la page 1023.
   - Exits de préconnexion: section PreConnect du fichier de configuration client.

ALW *Exits API non liés à une bibliothèque MQI*

Dans certaines circonstances, vous devez lier votre exit API existant, qui ne peut pas être recodé pour utiliser les pointeurs de fonction MQIEP, à une bibliothèque d'API IBM MQ .

Cette opération est nécessaire pour que votre exit API existant puisse être chargé, par l'éditeur de liens d'exécution de votre système, dans des programmes pour lesquels les pointeurs de fonction ne sont pas déjà chargés.

Remarque : Ces informations sont limitées aux exits API existants qui effectuent directement des appels MQI. C'est-à-dire les exits qui n'utilisent pas, MQIEP. Dans la mesure du possible, vous devez prévoir de recoder l'exit pour utiliser les points d'entrée MQIEP à la place.

runmqsc est un exemple de programme qui n'est pas directement lié à une bibliothèque MQI.

Par conséquent, un exit API qui n'a pas été lié à sa bibliothèque d'API IBM MQ requise, ou qui a été recodé pour utiliser le MQIEP, ne parvient pas à se charger dans **runmqsc**.

Vous voyez des erreurs dans le journal des erreurs du gestionnaire de files d'attente, par exemple, AMQ6175: Le système n'a pas pu charger dynamiquement la bibliothèque partagée, avec du texte qualifié tel que undefined symbol: MQCONN.

et AMQ7214: Le module de l'exit API'myexitname'n'a pas pu être chargé.

Tâches associées

«Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962
Vous pouvez écrire et compiler des exits sans liaison à des bibliothèques IBM MQ sur AIX, Linux, and Windows.

ALW Services et composants installables pour AIX, Linux, and Windows

Cette section présente les services installables ainsi que les fonctions et les composants qui leur sont associés. L'interface de ces fonctions est documentée afin que vous, ou les fournisseurs de logiciels, puissiez fournir des composants.

Les principales raisons pour lesquelles vous fournissez des services installables IBM MQ sont les suivantes:

- Pour vous offrir la flexibilité de choisir d'utiliser les composants fournis par les produits IBM MQ ou de les remplacer ou de les augmenter avec d'autres.
- Permettre aux fournisseurs de participer, en fournissant des composants qui peuvent utiliser de nouvelles technologies, sans apporter de modifications internes aux produits IBM MQ .
- Permettre à IBM MQ d'exploiter les nouvelles technologies plus rapidement et moins cher, et ainsi de fournir des produits plus tôt et à des prix plus bas.

Les *services optionnels* et les *composants de service* font partie de la structure du produit IBM MQ . Au centre de cette structure se trouve la partie du gestionnaire de files d'attente qui implémente la fonction et les règles associées à l'interface MQI (Message Queue Interface). Cette partie centrale requiert un certain nombre de fonctions de service, appelées *services installables*, pour effectuer son travail. Les services installables sont les suivants:

- Serveur d'autorisation
- Service annuaire

Chaque service installable est un ensemble connexe de fonctions implémentées à l'aide d'un ou de plusieurs *composants de service*. Chaque composant est appelé à l'aide d'une interface correctement structurée et accessible au public. Cela permet aux fournisseurs de logiciels indépendants et à d'autres tiers de fournir des composants installables pour augmenter ou remplacer ceux fournis par les produits IBM MQ . Le Tableau 138, à la page 966 récapitule les services et les composants qui peuvent être utilisés.

Tableau 138. Récapitulatif des composants de service installables

fonction installable	Composant fourni	Fonction	Exigences
Serveur d'autorisation	gestionnaire des droits d'accès aux objets (OAM)	Permet de vérifier les autorisations sur les commandes et les appels MQI. Les utilisateurs peuvent écrire leur propre composant pour augmenter ou remplacer la méthode d'accès aux objets (OAM). Par exemple, pour vérifier qu'un ID utilisateur est autorisé à ouvrir une file d'attente.	(Les installations d'autorisation de plateforme appropriées sont supposées)
Service annuaire	Aucun	Fournit une prise en charge au gestionnaire de files d'attente pour la recherche du nom du gestionnaire de files d'attente qui possède une file d'attente spécifiée. • Défini par l'utilisateur	• Un gestionnaire de noms tiers ou écrit par l'utilisateur

L'interface des services installables est décrite dans Informations de référence sur l'interface des services installables.

Tâches associées

Configuration des services installables

Ecriture d'un composant de service

Cette section décrit la relation entre les services, les composants, les points d'entrée et les codes retour.

Fonctions et composants

Chaque service se compose d'un ensemble de fonctions associées. Par exemple, le service annuaire contient une fonction pour:

- Recherche d'un nom de file d'attente et renvoi du nom du gestionnaire de files d'attente dans lequel la file d'attente est définie
- Insertion d'un nom de file d'attente dans le répertoire du service
- Suppression d'un nom de file d'attente dans le répertoire du service

Il contient également des fonctions d'initialisation et d'arrêt.

Un service installable est fourni par un ou plusieurs composants de service. Chaque composant peut exécuter certaines ou toutes les fonctions définies pour ce service. Par exemple, dans IBM MQ for AIX, le composant de service d'autorisation fourni, la méthode d'accès aux objets (OAM), exécute toutes les fonctions disponibles. Pour plus d'informations, voir «Interface de service d'autorisation», à la page 971. Le composant est également responsable de la gestion des ressources ou des logiciels sous-jacents (par exemple, un annuaire LDAP) dont il a besoin pour implémenter le service. Les fichiers de configuration fournissent un moyen standard de charger le composant et de déterminer les adresses des routines fonctionnelles qu'il fournit.

Le Figure 99, à la page 967 montre comment les services et les composants sont liés:

- Un service est défini dans un gestionnaire de files d'attente par des sections dans un fichier de configuration.
- Chaque service est pris en charge par le code fourni dans le gestionnaire de files d'attente. Les utilisateurs ne peuvent pas modifier ce code et ne peuvent donc pas créer leurs propres services.
- Chaque service est implémenté par un ou plusieurs composants ; ceux-ci peuvent être fournis avec le produit ou écrits par l'utilisateur. Plusieurs composants d'un service peuvent être appelés, chacun prenant en charge des fonctions différentes au sein du service.
- Les points d'entrée connectent les composants de service au code de prise en charge dans le gestionnaire de files d'attente.

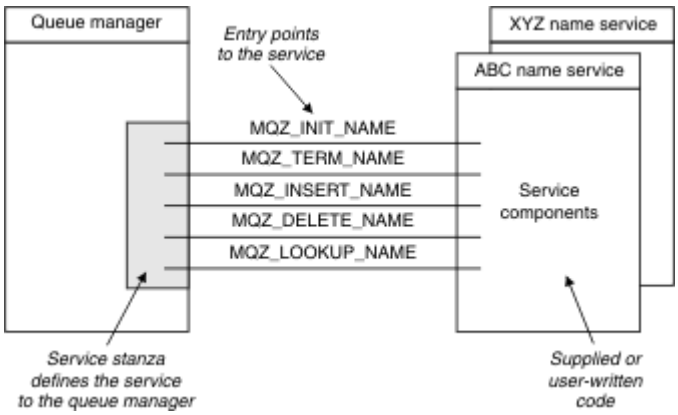


Figure 99. Description des services, des composants et des points d'entrée

Points d'entrée

Chaque composant de service est représenté par une liste des adresses de point d'entrée des routines qui prennent en charge un service installable particulier. Le service installable définit la fonction à exécuter par chaque routine.

L'ordre des composants de service lorsqu'ils sont configurés définit l'ordre dans lequel les points d'entrée sont appelés afin de satisfaire une demande pour le service.

Dans le fichier d'en-tête fourni `cmqzc.h`, les points d'entrée fournis à chaque service ont un préfixe MQZID_.

Si les services sont présents, ils sont chargés dans un ordre prédéfini. La liste suivante montre les services et l'ordre dans lequel ils sont initialisés.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` est le seul service configuré par défaut. Configurez `NameService` et `UserIdentifierService` manuellement si vous souhaitez les utiliser.

Les services et les composants de service ont un mappage un à un ou un à plusieurs. Plusieurs composants de service peuvent être définis pour chaque service. Sur les systèmes AIX and Linux , la valeur Service de la section `ServiceComponent` doit correspondre à la valeur Nom de la section Service dans le fichier qm.ini . Sous Windows, la valeur de la clé de registre du service `ServiceComponent` doit correspondre à la valeur de la clé de registre du nom et est définie comme suit: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\`*`MQ_INSTALLATION_NAME`*`\Configuration\QueueManager\`*`qmname`*`\` où *qmname* est le nom du gestionnaire de files d'attente.

Pour les systèmes AIX and Linux , les composants de service sont démarrés dans l'ordre dans lequel ils sont définis dans le fichier qm.ini . Sous Windows, comme le registre Windows est utilisé, IBM MQ émet un appel **RegEnumKey** qui renvoie les valeurs par ordre alphabétique. Par conséquent, sous Windows , les services sont appelés par ordre alphabétique, car ils sont définis dans le registre.

L'ordre des définitions `ServiceComponent` est significatif. Cet ordre dicte l'ordre dans lequel les composants sont exécutés pour un service donné. Par exemple, `AuthorizationService` sous Windows est configuré avec le composant OAM par défaut nommé `MQSeries.WindowsNT.auth.service`. Des composants supplémentaires peuvent être définis pour ce service afin de remplacer la méthode d'accès aux objets (OAM) par défaut. Sauf si `MQCACF_SERVICE_COMPONENT` est spécifié, le premier composant rencontré dans l'ordre alphabétique est utilisé pour traiter la demande et le nom de ce composant est utilisé.

Codes retour

Les composants de service fournissent des codes retour au gestionnaire de files d'attente pour générer des rapports sur diverses conditions. Ils signalent la réussite ou l'échec de l'opération et indiquent si le gestionnaire de files d'attente doit passer au composant de service suivant. Un paramètre *Continuation* distinct contient cette indication.

Données de composant

Un seul composant de service peut nécessiter le partage de données entre ses différentes fonctions. Les services installables fournissent une zone de données facultative à transmettre à chaque appel d'un composant de service. Cette zone de données est destinée à l'utilisation exclusive du composant de service. Il est partagé par tous les appels d'une fonction particulière, même s'ils sont effectués à partir d'espaces adresse ou de processus différents. Il est garanti qu'il est adressable à partir du composant de service chaque fois qu'il est appelé. Vous devez déclarer la taille de cette zone dans la section *`ServiceComponent`* .

Initialisation et arrêt des composants
Utilisation des options d'initialisation et d'arrêt des composants.

Lorsque la routine d'initialisation de composant est appelée, elle doit appeler la fonction **MQZEP** du gestionnaire de files d'attente pour chaque point d'entrée pris en charge par le composant. **MQZEP** définit un point d'entrée pour le service. Tous les points d'exit non définis sont supposés être NULL.

Un composant est toujours appelé une fois avec l'option d'initialisation principale, avant d'être appelé d'une autre manière.

Un composant peut être appelé avec l'option d'initialisation secondaire sur certaines plateformes. Par exemple, il peut être appelé une fois pour chaque processus, unité d'exécution ou tâche du système d'exploitation par lequel le service est accédé.

Si l'initialisation secondaire est utilisée:

- Le composant peut être appelé plusieurs fois pour l'initialisation secondaire. Pour chaque appel de ce type, un appel correspondant pour la terminaison secondaire est émis lorsque le service n'est plus nécessaire.

  Pour les services de nommage, il s'agit de l'appel MQZ_TERM_NAME.

  Pour les services d'autorisation, il s'agit de l'appel MQZ_TERM_AUTHORITY.
- Les points d'entrée doivent être redéfinis (en appelant MQZEP) chaque fois que le composant est appelé pour l'initialisation principale et secondaire.
- Une seule copie des données de composant est utilisée pour le composant ; il n'existe pas de copie différente pour chaque initialisation secondaire.
- Le composant n'est pas appelé pour d'autres appels au service (à partir du processus du système d'exploitation, de l'unité d'exécution ou de la tâche, selon le cas) avant que l'initialisation secondaire ait été effectuée.
- Le composant doit définir le paramètre **Version** sur la même valeur pour l'initialisation principale et secondaire.

Le composant est toujours appelé avec l'option de terminaison principale une fois, lorsqu'il n'est plus nécessaire. Aucun autre appel n'est effectué à ce composant.

Le composant est appelé avec l'option d'arrêt secondaire, s'il a été appelé pour l'initialisation secondaire.

Gestionnaire des droits d'accès aux objets (OAM)
Le composant de service d'autorisation fourni avec les produits IBM MQ est appelé Object Authority Manager (OAM).

Par défaut, la méthode d'accès aux objets (OAM) est active et fonctionne avec les commandes de contrôle **dspmqaut** (droit d'affichage), **dmpmqaut** (droit de vidage) et **setmqaut** (droit de définition ou de réinitialisation).

La syntaxe de ces commandes et leur utilisation sont décrites dans la rubrique Administration d' IBM MQ for Multiplatforms à l'aide de commandes de contrôle.

La méthode d'accès aux objets (OAM) fonctionne avec l' *entité* d'un principal ou d'un groupe:

- Linux AIX Sur les systèmes AIX and Linux , un principal est un ID utilisateur ou un ID associé à un programme d'application exécuté pour le compte d'un utilisateur ; un groupe est une collection de principaux définie par le système.
- Windows Sur les systèmes Windows , un principal est un ID utilisateur Windows ou un ID associé à un programme d'application exécuté pour le compte d'un utilisateur ; un groupe est un groupe Windows .

Les autorisations peuvent être accordées ou révoquées au niveau du principal ou du groupe.

Lorsqu'une demande MQI est émise ou qu'une commande est émise, la méthode d'accès aux objets (OAM) vérifie si l'entité associée à l'opération est autorisée à effectuer l'opération demandée et à accéder aux ressources de gestionnaire de files d'attente spécifiées.

Le service d'autorisation vous permet d'augmenter ou de remplacer la vérification des droits d'accès fournie pour les gestionnaires de files d'attente en écrivant votre propre composant de service d'autorisation.

Service annuaire
Le service annuaire est un service installable qui fournit une prise en charge au gestionnaire de files d'attente pour la recherche du nom du gestionnaire de files d'attente propriétaire d'une file d'attente spécifiée. Aucun autre attribut de file d'attente ne peut être extrait d'un service annuaire.

Le service annuaire permet à une application d'ouvrir des files d'attente distantes pour la sortie comme s'il s'agissait de files d'attente locales. Un service annuaire n'est pas appelé pour des objets autres que des files d'attente.

Remarque : L'attribut **Scope** des files d'attente éloignées doit être défini sur CELL.

Lorsqu'une application ouvre une file d'attente, elle recherche d'abord le nom de la file d'attente dans le répertoire du gestionnaire de files d'attente. S'il ne l'y trouve pas, il recherche dans autant de services de nom que ceux qui ont été configurés, jusqu'à ce qu'il en trouve un qui reconnaisse le nom de la file d'attente. Si aucun ne reconnaît le nom, l'ouverture échoue.

Le service annuaire renvoie le gestionnaire de files d'attente propriétaire de cette file d'attente. Le gestionnaire de files d'attente poursuit ensuite la demande MQOPEN comme si la commande avait indiqué le nom de la file d'attente et du gestionnaire de files d'attente dans la demande d'origine.

L'interface de service annuaire (NSI) fait partie de l'infrastructure IBM MQ .

Fonctionnement du service annuaire

Si une définition de file d'attente spécifie l'attribut **Scope** comme gestionnaire de files d'attente, c'est-à-dire, SCOPE (QMGR) dans MQSC, la définition de file d'attente (ainsi que tous les attributs de file d'attente) est stockée dans le répertoire du gestionnaire de files d'attente uniquement. Il ne peut pas être remplacé par un service installable.

Si une définition de file d'attente spécifie l'attribut **Scope** en tant que cellule, c'est-à-dire, SCOPE (CELL) dans MQSC, la définition de file d'attente est à nouveau stockée dans le répertoire du gestionnaire de files d'attente, avec tous les attributs de file d'attente. Toutefois, la file d'attente et le nom du gestionnaire de files d'attente sont également stockés dans un service annuaire. Si aucun service n'est disponible pour stocker ces informations, il n'est pas possible de définir une file d'attente avec la cellule *Scope* .

Le répertoire dans lequel les informations sont stockées peut être géré par le service, ou le service peut utiliser un service sous-jacent, par exemple un annuaire LDAP, à cette fin. Dans les deux cas, les définitions stockées dans le répertoire doivent être conservées, même après l'arrêt du composant et du gestionnaire de files d'attente, jusqu'à ce qu'elles soient explicitement supprimées.

Remarque :

1. Pour envoyer un message à la définition de file d'attente locale d'un hôte distant (avec une portée CELL) sur un gestionnaire de files d'attente différent dans une cellule de répertoire de nommage, vous devez définir un canal.
2. Vous ne pouvez pas obtenir de messages directement à partir de la file d'attente éloignée, même lorsqu'elle a une portée CELL.
3. Aucune définition de file d'attente éloignée n'est requise lors de l'envoi à une file d'attente dont la portée est CELL.
4. Le service de nommage définit de manière centralisée la file d'attente de destination, bien que vous ayez toujours besoin d'une file d'attente de transmission pour le gestionnaire de files d'attente de destination et d'une paire de définitions de canal. En outre, la file d'attente de transmission sur le système local doit avoir le même nom que le gestionnaire de files d'attente propriétaire de la file d'attente cible, avec la portée de la cellule, sur le système distant.

   Par exemple, si le gestionnaire de files d'attente éloignées a le nom QM01, la file d'attente de transmission sur le système local doit également avoir le nom QM01.

Interface de service d'autorisation
Le service d'autorisation fournit des points d'entrée à utiliser par le gestionnaire de files d'attente.

Les points d'entrée sont les suivants:

UTILISATEUR MQZ_AUTHENTICATE_USER
Authentifie un ID utilisateur et un mot de passe et peut définir des zones de contexte d'identité.

MQZ_CHECK_AUTHORITY
Vérifie si une entité est autorisée à effectuer une ou plusieurs opérations sur un objet spécifié.

MQZ_CHECK_PRIVILEGED
Vérifie si un utilisateur spécifié est un utilisateur privilégié.

MQZ_COPY_ALL_AUTHORITY
Copie toutes les autorisations en cours qui existent pour un objet référencé dans un autre objet.

MQZ_DELETE_AUTHORITY
Supprime toutes les autorisations associées à un objet spécifié.

MQZ_ENUMERATE_AUTHORITY_DATA
Extrait toutes les données de droits d'accès qui correspondent aux critères de sélection indiqués.

Utilisateur_FREE_MQZ
Libère les ressources allouées associées.

MQZ_GET_AUTHORITY
Obtient les droits dont dispose une entité pour accéder à un objet spécifié.

MQZ_GET_EXPLICIT_AUTHORITY
Obtient soit les droits dont dispose un groupe nommé pour accéder à un objet spécifié (mais sans les droits supplémentaires du groupe **personne**), soit les droits dont dispose le groupe principal du principal nommé pour accéder à un objet spécifié.

MQZ_XX_ENCODE_CASE_ONE autorite_instance
Initialise le composant de service d'autorisation.

MQZ_INQUIRE
Interroge la fonctionnalité prise en charge du service d'autorisation.

MQZ_CACHE d'actualisation
Actualisez toutes les autorisations.

MQZ_SET_AUTHORITY
Définit les droits dont dispose une entité sur un objet spécifié.

MQZ_TERM_AUTHORITY
Arrête le composant de service d'autorisation.

En outre, sous IBM MQ for Windows, le service d'autorisation fournit les points d'entrée suivants à utiliser par le gestionnaire de files d'attente:

- **MQZ_CHECK_AUTHORITY_2**
- **MQZ_GET_AUTHORITY_2**
- **MQZ_GET_EXPLICIT_AUTHORITY_2**
- **MQZ_SET_AUTHORITY_2**

Ces points d'entrée prennent en charge l'utilisation de l'identificateur de sécurité Windows (NT SID).

Ces noms sont définis en tant que **typedef** s, dans le fichier d'en-tête `cmqzc.h`, qui peut être utilisé pour prototyper les fonctions de composant.

La fonction d'initialisation (**MQZ_INIT_AUTHORITY**) doit être le point d'entrée principal du composant. Les autres fonctions sont appelées via l'adresse de point d'entrée que la fonction d'initialisation a ajoutée dans le vecteur de point d'entrée de composant.

Interface de service annuaire
Un service annuaire fournit des points d'entrée à utiliser par le gestionnaire de files d'attente.

Les points d'entrée suivants sont fournis:

NOM_INIT_MQZ
Initialisez le composant de service annuaire.

NOM_MQZ
Arrêtez le composant de service annuaire.

NOM_RECHERCHE_MQZ
Recherchez le nom du gestionnaire de files d'attente pour la file d'attente spécifiée.

NOM_INSERT_MQZ
Insérez une entrée contenant le nom du gestionnaire de files d'attente propriétaire de la file d'attente spécifiée dans le répertoire utilisé par le service.

NOM_DELET_MQZ
Supprimez l'entrée de la file d'attente indiquée du répertoire utilisé par le service.

Si plusieurs services d'annuaire sont configurés:

- Pour la recherche, la fonction MQZ_LOOKUP_NAME est appelée pour chaque service de la liste jusqu'à ce que le nom de la file d'attente soit résolu (sauf si un composant indique que la recherche doit s'arrêter).
- Pour l'insertion, la fonction MQZ_INSERT_NAME est appelée pour le premier service de la liste qui prend en charge cette fonction.
- Pour la suppression, la fonction MQZ_DELETE_NAME est appelée pour le premier service de la liste qui prend en charge cette fonction.

Ne disposez pas de plus d'un composant prenant en charge les fonctions d'insertion et de suppression. Toutefois, un composant qui ne prend en charge que la recherche est faisable et peut être utilisé, par exemple, comme dernier composant de la liste pour résoudre un nom qui n'est pas connu par un autre composant de service de nom dans un gestionnaire de files d'attente sur lequel le nom peut être défini.

Dans le langage de programmation C, les noms sont définis en tant que types de données de fonction à l'aide de l'instruction typedef. Ils peuvent être utilisés pour prototyper les fonctions de service, afin de s'assurer que les paramètres sont corrects.

Le fichier d'en-tête qui contient tous les éléments spécifiques aux services installables est `cmqzc.h` pour le langage C.

Outre la fonction d'initialisation (MQZ_INIT_NAME), qui doit être le point d'entrée principal du composant, les fonctions sont appelées par l'adresse de point d'entrée ajoutée par la fonction d'initialisation, à l'aide de l'appel MQZEP.

Utilisation de plusieurs composants de service
Vous pouvez installer plusieurs composants pour un service. Cela permet aux composants de fournir uniquement des implémentations partielles du service et de s'appuyer sur d'autres composants pour fournir les fonctions restantes.

Exemple d'utilisation de plusieurs composants

Supposons que vous créiez deux composants de services de nom appelés `ABC_name_serv` et `XYZ_name_serv`.

ABC_name_serv
Ce composant prend en charge l'insertion ou la suppression d'un nom dans le répertoire de service, mais ne prend pas en charge la recherche d'un nom de file d'attente.

XYZ_name_serv
Ce composant prend en charge la recherche d'un nom de file d'attente, mais ne prend pas en charge l'insertion ou la suppression d'un nom dans le répertoire de service.

Le composant `ABC_name_serv` contient une base de données de noms de file d'attente et utilise deux algorithmes simples pour insérer ou supprimer un nom dans le répertoire de service.

Le composant XYZ_name_serv utilise un algorithme simple qui renvoie un nom de gestionnaire de files d'attente fixe pour tout nom de file d'attente avec lequel il est appelé. Il ne contient pas de base de données de noms de file d'attente et ne prend donc pas en charge les fonctions d'insertion et de suppression.

Les composants sont installés sur le même gestionnaire de files d'attente. Les sections *ServiceComponent* sont ordonnées de sorte que le composant ABC_name_serv soit appelé en premier. Tous les appels d'insertion ou de suppression d'une file d'attente dans un répertoire de composant sont gérés par le composant ABC_name_serv ; c'est le seul qui implémente ces fonctions. Toutefois, un appel de recherche que le composant ABC_name_serv ne peut pas résoudre est transmis au composant de recherche uniquement, XYZ_name_serv. Ce composant fournit un nom de gestionnaire de files d'attente à partir de son algorithme simple.

Omission de points d'entrée lors de l'utilisation de plusieurs composants

Si vous décidez d'utiliser plusieurs composants pour fournir un service, vous pouvez concevoir un composant de service qui n'implémente pas certaines fonctions. L'infrastructure des services installables n'impose aucune restriction que vous pouvez omettre. Toutefois, pour des services installables spécifiques, l'omission d'une ou de plusieurs fonctions peut être logiquement incompatible avec l'objectif du service.

Exemple de points d'entrée utilisés avec plusieurs composants

La Tableau 139, à la page 973 illustre un exemple de service de nom installable pour lequel les deux composants ont été installés. Chacun prend en charge un ensemble différent de fonctions associées à ce service installable particulier. Pour la fonction d'insertion, le point d'entrée du composant ABC est appelé en premier. Points d'entrée qui n'ont pas été définis pour le service (à l'aide de **MQZEP**) sont supposés être NULL. Un point d'entrée pour l'initialisation est fourni dans la table, mais cela n'est pas obligatoire car l'initialisation est effectuée par le point d'entrée principal du composant.

Lorsque le gestionnaire de files d'attente doit utiliser un service installable, il utilise les points d'entrée définis pour ce service (les colonnes dans Tableau 139, à la page 973). Pour chaque composant, le gestionnaire de files d'attente détermine l'adresse de la routine qui implémente la fonction requise. Il appelle ensuite la routine, si elle existe. Si l'opération aboutit, les résultats et les informations de statut sont utilisés par le gestionnaire de files d'attente.

Tableau 139. Exemple de points d'entrée pour un service installable

Numéro de fonction	**Composant de service de nom ABC**	**Composant de service annuaire XYZ**
MQZID_INIT_NAME (initialisation)	ABC_initialize ()	XYZ_initialize ()
MQZID_TERM_NAME (Terminate)	ABC_terminate ()	XYZ_terminate ()
MQZID_INSERT_NAME (insertion)	ABC_Insert ()	NULL
MQZID_DELETE_NAME (Supprimer)	ABC_Delete ()	NULL
MQZID_LOOKUP_NAME (recherche)	NULL	XYZ_Lookup ()

Si la routine n'existe pas, le gestionnaire de files d'attente répète ce processus pour le composant suivant de la liste. En outre, si la routine existe mais renvoie un code indiquant qu'elle n'a pas pu effectuer l'opération, la tentative se poursuit avec le composant disponible suivant. Les routines des composants de service peuvent renvoyer un code indiquant qu'aucune autre tentative d'exécution de l'opération ne doit être effectuée.

Configuration des services et des composants

Vous configurez les composants de service à l'aide des fichiers de configuration du gestionnaire de files d'attente, sauf sur les systèmes Windows , où chaque gestionnaire de files d'attente possède sa propre section dans le registre.

Procédure

1. Ajoutez des sections au fichier de configuration du gestionnaire de files d'attente, `qm.ini`, pour définir le service dans le gestionnaire de files d'attente et spécifier l'emplacement du module:
   - Chaque service utilisé doit comporter une section `Service` qui définit le service dans le gestionnaire de files d'attente. Pour plus d'informations, voir Service stanza of qm.ini file.
   - Pour chaque composant d'un service, il doit y avoir une section `ServiceComponent` . Cette section identifie le nom et le chemin du module contenant le code de ce composant. Pour plus d'informations, voir la section ServiceComponent du fichier qm.ini.

   Le composant de service d'autorisation, appelé Object Authority Manager (OAM), est fourni avec le produit. Lorsque vous créez un gestionnaire de files d'attente, le fichier de configuration du gestionnaire de files d'attente (ou le registre sur les systèmes Windows) est automatiquement mis à jour pour inclure les sections appropriées pour le service d'autorisation et pour le composant par défaut (OAM). Pour les autres composants, vous devez configurer le fichier de configuration du gestionnaire de files d'attente manuellement.

   Le code de chaque composant de service est chargé dans le gestionnaire de files d'attente lors du démarrage du gestionnaire de files d'attente, à l'aide de la liaison dynamique, qui est prise en charge sur la plateforme.
2. Arrêtez et redémarrez le gestionnaire de files d'attente pour activer le composant.

Référence associée

Section de service du fichier qm.ini

Section ServiceComponent du fichier qm.ini

Actualisation de la méthode d'accès aux objets (OAM) après la modification de l'autorisation d'un utilisateur

Dans IBM MQ, vous pouvez actualiser les informations du groupe d'autorisations de la méthode d'accès aux objets immédiatement après avoir modifié l'appartenance à un groupe d'autorisations d'un utilisateur, en reflétant les modifications apportées au niveau du système d'exploitation, sans avoir à arrêter et à redémarrer le gestionnaire de files d'attente. Pour ce faire, exécutez la commande **REFRESH SECURITY** .

Remarque : Lorsque vous modifiez les autorisations à l'aide de la commande `setmqaut` , la méthode d'accès aux objets (OAM) implémente ces modifications immédiatement.

Les gestionnaires de files d'attente stockent les données d'autorisation dans une file d'attente locale appelée SYSTEM.AUTH.DATA.QUEUE. Ces données sont gérées par **amqzfuma.exe**.

Référence associée

REFRESH SECURITY

IBM i Services et composants installables sous IBM i

Utilisez ces informations pour en savoir plus sur les services installables et les fonctions et composants qui leur sont associés. L'interface de ces fonctions est documentée afin que vous, ou les fournisseurs de logiciels, puissiez fournir des composants.

Les principales raisons pour lesquelles vous fournissez des services installables IBM MQ sont les suivantes:

- Pour vous offrir la flexibilité de choisir d'utiliser les composants fournis par IBM MQ for IBM i, ou de les remplacer ou de les augmenter avec d'autres.
- Permettre aux fournisseurs de participer, en fournissant des composants qui peuvent utiliser de nouvelles technologies, sans apporter de modifications internes à IBM MQ for IBM i.

- Permettre à IBM MQ d'exploiter les nouvelles technologies plus rapidement et moins cher, et ainsi de fournir des produits plus tôt et à des prix plus bas.

Les *services optionnels* et les *composants de service* font partie de la structure du produit IBM MQ . Au centre de cette structure se trouve la partie du gestionnaire de files d'attente qui implémente la fonction et les règles associées à l'interface MQI (Message Queue Interface). Cette partie centrale requiert un certain nombre de fonctions de service, appelées *services installables*, pour effectuer son travail. Le service installable disponible dans IBM MQ for IBM i est le service d'autorisation.

Chaque service installable est un ensemble connexe de fonctions implémentées à l'aide d'un ou de plusieurs *composants de service*. Chaque composant est appelé à l'aide d'une interface correctement structurée et accessible au public. Cela permet aux éditeurs de logiciels indépendants et à d'autres tiers de fournir des composants installables pour augmenter ou remplacer ceux fournis par IBM MQ for IBM i. Le Tableau 140, à la page 975 récapitule la prise en charge du service d'autorisation.

Tableau 140. Récapitulatif des composants du service d'autorisation

Composant fourni	**Fonction**	**Exigences**
Gestionnaire des droits d'accès aux objets (OAM)	Permet de vérifier les autorisations sur les commandes et les appels MQI. Les utilisateurs peuvent écrire leur propre composant pour augmenter ou remplacer la méthode d'accès aux objets (OAM).	(Les installations d'autorisation de plateforme appropriées sont supposées)
Composant de service de nom DCE **Remarque :** DCE n'est pris en charge que sur les versions de IBM MQ antérieures à V6.0.	• Permet aux gestionnaires de files d'attente de partager des files d'attente ou • Défini par l'utilisateur **Remarque :** L'attribut **Scope** des files d'attente partagées doit être défini sur CELL.	• DCE est requis pour le composant fourni, ou • Un gestionnaire de noms tiers ou écrit par l'utilisateur

IBM i *Fonctions et composants sous IBM i*

Utilisez ces informations pour comprendre les fonctions et les composants, les points d'entrée, les codes retour et les données de composant que vous pouvez utiliser dans IBM MQ for IBM i.

Chaque service se compose d'un ensemble de fonctions associées. Par exemple, le service annuaire contient une fonction pour:

- Recherche d'un nom de file d'attente et renvoi du nom du gestionnaire de files d'attente dans lequel la file d'attente est définie
- Insertion d'un nom de file d'attente dans le répertoire du service
- Suppression d'un nom de file d'attente dans le répertoire du service

Il contient également des fonctions d'initialisation et d'arrêt.

Un service installable est fourni par un ou plusieurs composants de service. Chaque composant peut exécuter certaines ou toutes les fonctions définies pour ce service. Le composant est également responsable de la gestion des ressources ou des logiciels sous-jacents dont il a besoin pour implémenter le service. Les fichiers de configuration fournissent un moyen standard de charger le composant et de déterminer les adresses des routines fonctionnelles qu'il fournit.

Les services et les composants sont liés comme suit:

- Un service est défini dans un gestionnaire de files d'attente par des sections dans un fichier de configuration.
- Chaque service est pris en charge par le code fourni dans le gestionnaire de files d'attente. Les utilisateurs ne peuvent pas modifier ce code et ne peuvent donc pas créer leurs propres services.

- Chaque service est implémenté par un ou plusieurs composants ; ceux-ci peuvent être fournis avec le produit ou écrits par l'utilisateur. Plusieurs composants d'un service peuvent être appelés, chacun prenant en charge des fonctions différentes au sein du service.
- Les points d'entrée connectent les composants de service au code de prise en charge dans le gestionnaire de files d'attente.

Points d'entrée

Chaque composant de service est représenté par une liste des adresses de point d'entrée des routines qui prennent en charge un service installable particulier. Le service installable définit la fonction à exécuter par chaque routine. L'ordre des composants de service lorsqu'ils sont configurés définit l'ordre dans lequel les points d'entrée sont appelés afin de satisfaire une demande pour le service. Dans le fichier d'en-tête fourni `cmqzc.h`, les points d'entrée fournis à chaque service ont un préfixe MQZID_.

Codes retour

Les composants de service fournissent des codes retour au gestionnaire de files d'attente pour générer des rapports sur diverses conditions. Ils signalent la réussite ou l'échec de l'opération et indiquent si le gestionnaire de files d'attente doit passer au composant de service suivant. Un paramètre *Continuation* distinct contient cette indication.

Données de composant

Un seul composant de service peut nécessiter le partage de données entre ses différentes fonctions. Les services installables fournissent une zone de données facultative à transmettre à chaque appel d'un composant de service particulier. Cette zone de données est destinée à l'utilisation exclusive du composant de service. Il est partagé par tous les appels d'une fonction donnée, même s'ils sont effectués à partir d'espaces adresse ou de processus différents. Il est garanti qu'il est adressable à partir du composant de service chaque fois qu'il est appelé. Vous devez déclarer la taille de cette zone dans la section *`ServiceComponent`* .

IBM i *Initialisation sur IBM i*

Lorsque la routine d'initialisation de composant est appelée, elle doit appeler la fonction MQZEP du gestionnaire de files d'attente pour chaque point d'entrée pris en charge par le composant. MQZEP définit un point d'entrée pour le service. Tous les points d'exit non définis sont supposés être NULL.

Initialisation principale
Un composant est toujours appelé avec cette option une fois, avant d'être appelé d'une autre manière.

Initialisation secondaire
Un composant peut être appelé avec cette option sur certaines plateformes. Par exemple, il peut être appelé une fois pour chaque processus, unité d'exécution ou tâche du système d'exploitation par lequel le service est accédé.

Si l'initialisation secondaire est utilisée:

- Le composant peut être appelé plusieurs fois pour l'initialisation secondaire. Pour chaque appel de ce type, un appel correspondant pour la terminaison secondaire est émis lorsque le service n'est plus nécessaire.

  Pour les services d'autorisation, il s'agit de l'appel MQZ_TERM_AUTHORITY.
- Les points d'entrée doivent être redéfinis (en appelant MQZEP) chaque fois que le composant est appelé pour l'initialisation principale et secondaire.
- Une seule copie des données de composant est utilisée pour le composant ; il n'existe pas de copie différente pour chaque initialisation secondaire.
- Le composant n'est pas appelé pour d'autres appels au service (à partir du processus du système d'exploitation, de l'unité d'exécution ou de la tâche, selon le cas) avant que l'initialisation secondaire ait été effectuée.

- Le composant doit définir le paramètre **Version** sur la même valeur pour l'initialisation principale et secondaire.

Arrêt principal
Le composant est toujours démarré avec cette option une fois, lorsqu'il n'est plus nécessaire. Aucun autre appel n'est effectué à ce composant.

Arrêt secondaire
Le composant est démarré avec cette option, s'il a été démarré pour l'initialisation secondaire.

IBM i *Configuration de services et de composants sous IBM i*

Vous configurez les composants de service à l'aide des fichiers de configuration du gestionnaire de files d'attente.

Procédure

1. Ajoutez des sections au fichier de configuration du gestionnaire de files d'attente, `qm.ini`, pour définir le service dans le gestionnaire de files d'attente et spécifier l'emplacement du module:
   - Chaque service utilisé doit comporter une section `Service` qui définit le service dans le gestionnaire de files d'attente. Pour plus d'informations, voir Service stanza of qm.ini file.
   - Pour chaque composant d'un service, il doit y avoir une section `ServiceComponent` . Cette section identifie le nom et le chemin du module contenant le code de ce composant. Pour plus d'informations, voir la section ServiceComponent du fichier qm.ini.

   Le composant de service d'autorisation, appelé Object Authority Manager (OAM), est fourni avec le produit. Lorsque vous créez un gestionnaire de files d'attente, le fichier de configuration du gestionnaire de files d'attente est automatiquement mis à jour pour inclure les sections appropriées pour le service d'autorisation et pour le composant par défaut (la méthode d'accès aux objets (OAM)). Pour les autres composants, vous devez configurer le fichier de configuration du gestionnaire de files d'attente manuellement.

   Le code de chaque composant de service est chargé dans le gestionnaire de files d'attente lors du démarrage du gestionnaire de files d'attente, à l'aide de la liaison dynamique, qui est prise en charge sur la plateforme.
2.

IBM i *Création de votre propre composant de service sous IBM i*

Utilisez ces informations pour apprendre à créer un composant de service sur IBM MQ for IBM i.

Pour créer votre propre composant de service:

- Vérifiez que le fichier d'en-tête `cmqzc.h` est inclus dans votre programme.
- Créez la bibliothèque partagée en compilant le programme et en le liant aux bibliothèques partagées `libmqm*` et `libmqmzf*`.

  Remarque : Etant donné que l'agent peut s'exécuter dans un environnement à unités d'exécution, vous devez générer la méthode d'accès aux objets (OAM) pour qu'elle s'exécute dans un environnement à unités d'exécution. Cela inclut l'utilisation des versions à unités d'exécution de `libmqm` et `libmqmzf`.
- Ajoutez des sections au fichier de configuration du gestionnaire de files d'attente pour définir le service dans le gestionnaire de files d'attente et spécifier l'emplacement du module.
- Arrêtez et redémarrez le gestionnaire de files d'attente pour activer le composant.

IBM i *Service d'autorisation sur IBM i*

Le service d'autorisation est un service installable qui permet aux gestionnaires de files d'attente d'appeler des fonctions d'autorisation, par exemple, en vérifiant qu'un ID utilisateur est autorisé à ouvrir une file d'attente.

Ce service est un composant de l'interface d'activation de la sécurité (SEI) IBM MQ , qui fait partie de l'infrastructure IBM MQ . Les sujets suivants sont abordés:

Gestionnaire des droits d'accès aux objets (OAM)

Le composant de service d'autorisation fourni avec les produits IBM MQ est appelé gestionnaire des droits d'accès aux objets (OAM). Par défaut, la méthode d'accès aux objets (OAM) est active et fonctionne avec les commandes de contrôle suivantes:

- **WRKMQMAUT** gestion des droits d'accès
- **WRKMQMAUTD** gestion des données de droits d'accès
- Droits sur les objets écran **DSPMQMAUT**
- **GRTMQMAUT** accorder des droits sur les objets
- **RVKMQMAUT** révoquer les droits sur les objets
- **RFRMQMAUT** Actualiser la sécurité

La syntaxe de ces commandes et leur utilisation sont décrites dans l'aide sur les commandes CL. La méthode d'accès aux objets (OAM) fonctionne avec l' *entité* d'un principal ou d'un groupe.

Lorsqu'une demande MQI est émise ou qu'une commande est émise, la méthode d'accès aux objets (OAM) vérifie l'autorisation de l'entité associée à l'opération pour voir si elle peut effectuer les actions suivantes:

- Effectuez l'opération demandée.
- Accédez aux ressources de gestionnaire de files d'attente spécifiées.

Le service d'autorisation vous permet d'augmenter ou de remplacer la vérification des droits d'accès fournie pour les gestionnaires de files d'attente en écrivant votre propre composant de service d'autorisation.

Définition du service sur le système d'exploitation

Les sections du service d'autorisation dans le fichier de configuration du gestionnaire de files d'attente `qm.ini` définissent le service d'autorisation dans le gestionnaire de files d'attente. Pour plus d'informations sur les types de section, voir «Configuration de services et de composants sous IBM i», à la page 977 .

Configuration des sections de service d'autorisation

Sous IBM MQ for IBM i :

Principale
Est un profil utilisateur système IBM i .

Groupe
Est un profil de groupe de systèmes IBM i .

Les autorisations ne peuvent être accordées ou révoquées qu'au niveau du groupe. Une demande d'octroi ou de révocation des droits d'un utilisateur met à jour le groupe principal de cet utilisateur.

Chaque gestionnaire de files d'attente possède son propre fichier de configuration de gestionnaire de files d'attente. Par exemple, le chemin d'accès et le nom de fichier par défaut du fichier de configuration du gestionnaire de files d'attente QMNAME est `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

La section *`Service`* et la section *`ServiceComponent`* du composant d'autorisation par défaut sont ajoutées à `qm.ini` automatiquement, mais peuvent être remplacées par `WRKENVVAR`. Toute autre strophe *`ServiceComponent`* doit être ajoutée manuellement.

Par exemple, les sections suivantes du fichier de configuration du gestionnaire de files d'attente définissent deux composants de service d'autorisation:

```
Service:
    Name=AuthorizationService
    EntryPoints=7

ServiceComponent:
    Service=AuthorizationService
    Name=MQ.UNIX.authorization.service
    Module=QMQM/AMQZFU
    ComponentDataSize=0

ServiceComponent:
    Service=AuthorizationService
    Name=user.defined.authorization.service
    Module=LIBRARY/SERVICE PROGRAM NAME
    ComponentDataSize=96
```

Figure 100. Sections de service d'autorisation dans qm.ini sur IBM i

La première section de composant de service `MQ.UNIX.authorization.service` définit le composant de service d'autorisation par défaut, OAM. Si vous supprimez cette section et redémarrez le gestionnaire de files d'attente, la méthode d'accès aux objets (OAM) est désactivée et aucune vérification d'autorisation n'est effectuée.

IBM i *Interface de service d'autorisation sur IBM i*

L'interface de service d'autorisation fournit plusieurs points d'entrée à utiliser par le gestionnaire de files d'attente.

UTILISATEUR MQZ_AUTHENTICATE_USER
Authentifie un ID utilisateur et un mot de passe et peut définir des zones de contexte d'identité.

MQZ_CHECK_AUTHORITY
Vérifie si une entité est autorisée à effectuer une ou plusieurs opérations sur un objet spécifié.

MQZ_COPY_ALL_AUTHORITY
Copie toutes les autorisations en cours qui existent pour un objet référencé dans un autre objet.

MQZ_DELETE_AUTHORITY
Supprime toutes les autorisations associées à un objet spécifié.

MQZ_ENUMERATE_AUTHORITY_DATA
Extrait toutes les données de droits d'accès qui correspondent aux critères de sélection indiqués.

Utilisateur_FREE_MQZ
Libère les ressources allouées associées.

MQZ_GET_AUTHORITY
Obtient les droits dont dispose une entité pour accéder à un objet spécifié.

MQZ_GET_EXPLICIT_AUTHORITY
Obtient soit les droits dont dispose un groupe nommé pour accéder à un objet spécifié (mais sans les droits supplémentaires du groupe **personne**), soit les droits dont dispose le groupe principal du principal nommé pour accéder à un objet spécifié.

MQZ_XX_ENCODE_CASE_ONE autorite_instance
Initialise le composant de service d'autorisation.

MQZ_INQUIRE
Interroge la fonctionnalité prise en charge du service d'autorisation.

MQZ_CACHE d'actualisation
Actualisez toutes les autorisations.

MQZ_SET_AUTHORITY
Définit les droits dont dispose une entité sur un objet spécifié.

MQZ_TERM_AUTHORITY
Arrête le composant de service d'autorisation.

Ces points d'entrée prennent en charge l'utilisation de l'identificateur de sécurité Windows (NT SID).

Ces noms sont définis en tant que **typedef** s, dans le fichier d'en-tête `cmqzc.h`, qui peut être utilisé pour prototyper les fonctions de composant.

La fonction d'initialisation (**MQZ_INIT_AUTHORITY**) doit être le point d'entrée principal du composant. Les autres fonctions sont appelées via l'adresse de point d'entrée que la fonction d'initialisation a ajoutée dans le vecteur de point d'entrée de composant.

Pour plus d'informations, voir «Création de votre propre composant de service sous IBM i», à la page 977.

Multi Ecriture et compilation des exits API sur Multiplatforms

Les exits API permettent d'écrire du code modifiant le comportement des appels API IBM MQ, tels que MQPUT et MQGET, puis d'insérer le code immédiatement avant ou après ces appels.

Remarque : z/OS Non pris en charge sur IBM MQ for z/OS.

Pourquoi utiliser les exits API?

Chacune de vos applications a un travail spécifique à effectuer et son code doit effectuer cette tâche aussi efficacement que possible. A un niveau supérieur, vous pouvez appliquer des normes ou des processus métier à un gestionnaire de files d'attente particulier pour toutes les applications qui utilisent ce gestionnaire de files d'attente. Il est plus efficace de le faire au-delà du niveau des applications individuelles, et donc sans avoir à modifier le code de chaque application affectée.

Voici quelques suggestions de domaines dans lesquels les exits API peuvent être utiles:

Sécurité
Pour des raisons de sécurité, vous pouvez fournir une authentification en vérifiant que les applications sont autorisées à accéder à une file d'attente ou à un gestionnaire de files d'attente. Vous pouvez également contrôler l'utilisation de l'API par les applications, en authentifiant les appels d'API individuels, ou même les paramètres qu'elles utilisent.

Flexible
Pour plus de flexibilité, vous pouvez réagir aux changements rapides de votre environnement métier sans changer les applications qui s'appuient sur les données de cet environnement. Vous pouvez, par exemple, avoir des exits API qui répondent à des changements de taux d'intérêt, de taux de change ou de prix de composants dans un environnement de fabrication.

Surveillance de l'utilisation d'une file d'attente ou d'un gestionnaire de files d'attente
Pour surveiller l'utilisation d'une file d'attente ou d'un gestionnaire de files d'attente, vous pouvez tracer le flux des applications et des messages, consigner des erreurs dans les appels API, configurer des traces d'audit à des fins de comptabilité ou collecter des statistiques d'utilisation à des fins de planification.

Que se passe-t-il lorsqu'un exit d'API s'exécute?

Une fois que vous avez écrit un programme d'exit et que vous l'avez identifié dans IBM MQ, le gestionnaire de files d'attente appelle automatiquement votre code d'exit aux points enregistrés.

Les routines d'exit d'API à exécuter sont identifiées dans des sections sur Multiplatforms. Cette rubrique décrit les sections des fichiers de configuration `mqs.ini` et `qm.ini`.

La définition des routines peut se produire à trois endroits:

1. ApiExitCommon, dans le fichier mqs.ini , identifie les routines, pour l'ensemble de IBM MQ, appliquées au démarrage des gestionnaires de files d'attente. Ils peuvent être remplacés par des routines définies pour des gestionnaires de files d'attente individuels (voir l'élément «3», à la page 981 dans cette liste).
2. Le modèle ApiExit, dans le fichier mqs.ini , identifie les routines, pour l'ensemble de IBM MQ, copiées dans l'ensemble local ApiExit(voir l'élément «3», à la page 981 dans cette liste) lorsqu'un nouveau gestionnaire de files d'attente est créé.

3. ApiExitLocal, dans le fichier qm.ini , identifie les routines qui s'appliquent à un gestionnaire de files d'attente particulier.

Lorsqu'un nouveau gestionnaire de files d'attente est créé, les définitions de modèle ApiExitdans mqs.ini sont copiées dans les définitions locales ApiExitdans qm.ini du nouveau gestionnaire de files d'attente. Lorsqu'un gestionnaire de files d'attente est démarré, les définitions ApiExitCommon et ApiExitLocal sont utilisées. Les définitions locales ApiExitremplacent les définitions communes ApiExitsi les deux identifient une routine du même nom. L'attribut Sequence , décrit dans «Configuration des exits API», à la page 985 , détermine l'ordre dans lequel les routines définies dans les sections s'exécutent.

Utilisation des exits API dans plusieurs installations d' IBM MQ

Assurez-vous que les exits API écrits pour la version antérieure de IBM MQ sont utilisés pour fonctionner avec toutes les versions car les modifications apportées aux exits dans IBM WebSphere MQ 7.1 risquent de ne pas fonctionner avec une version antérieure. Pour plus d'informations sur les modifications apportées aux exits, voir «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962.

Les exemples fournis pour les exits API amqsaem et amqsaxe reflètent les modifications requises lors de l'écriture des exits. L'application client doit s'assurer que les bibliothèques IBM MQ appropriées qui correspondent à l'installation du gestionnaire de files d'attente auquel l'application est associée lui sont liées avant le lancement de l'application.

Multi *Ecriture des exits API*

Vous pouvez écrire des exits pour chaque appel d'API à l'aide du langage de programmation C.

Exits disponibles

Les exits sont disponibles pour chaque appel d'API, comme suit:

- MQCB, pour réenregistrer un rappel pour le descripteur d'objet spécifié et contrôler l'activation et les modifications du rappel
- MQCTL, pour effectuer des actions de contrôle sur les descripteurs d'objet ouverts pour une connexion
- MQCONN/MQCONNX, pour fournir un descripteur de connexion de gestionnaire de files d'attente à utiliser lors des appels API suivants
- MQDISC, pour se déconnecter d'un gestionnaire de files d'attente
- MQBEGIN, pour démarrer une unité de travail globale (UOW)
- MQBACK, pour l'exécution d'une unité de travail
- MQCMIT, pour valider une unité de travail
- MQOPEN, pour ouvrir une ressource IBM MQ pour un accès ultérieur
- MQCLOSE, pour fermer une ressource IBM MQ précédemment ouverte pour accès
- MQGET, pour extraire un message d'une file d'attente précédemment ouverte pour accès
- MQPUT1, pour placer un message dans une file d'attente
- MQPUT, pour placer un message dans une file d'attente précédemment ouverte pour accès
- MQINQ, pour interroger les attributs d'une ressource IBM MQ qui a été précédemment ouverte pour l'accès
- MQSET, pour définir les attributs d'une file d'attente précédemment ouverte pour accès
- MQSTAT, pour extraire les informations de statut
- MQSUB, pour enregistrer l'abonnement des applications à une rubrique particulière
- MQSUBRQ, pour effectuer une demande d'abonnement

MQ_CALLBACK_EXIT fournit une fonction d'exit à exécuter avant et après le traitement du rappel. Pour plus d'informations, voir Callback-MQ_CALLBACK_EXIT.

Ecriture des exits API

Dans les exits API, les appels prennent la forme générale suivante:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

où *call* est le nom de l'appel MQI sans le préfixe MQ ; par exemple, PUT, GET. Le *parameters* contrôle la fonction de l'exit, en fournissant principalement la communication entre l'exit et les blocs de contrôle externes MQAXP (structure de paramètres d'exit d'API) et MQAXC (structure de contexte d'exit d'API). *context* décrit le contexte dans lequel l'exit API a été appelé et *ApiCallParameters* représente les paramètres de l'appel MQI.

Pour vous aider à écrire votre exit API, un exemple d'exit, amqsaxe0.c, est fourni ; cet exit génère des entrées de trace dans un fichier que vous spécifiez. Vous pouvez utiliser cet exemple comme point de départ lors de l'écriture des exits. Pour plus d'informations sur l'utilisation de l'exemple d'exit, voir «Exemple de programme d'exit API», à la page 1106.

Pour plus d'informations sur les appels d'exit d'API, les blocs de contrôle externes et les rubriques associées, voir Référence d'exit d'API.

Pour obtenir des informations générales sur l'écriture, la compilation et la configuration d'un exit, voir «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962.

Utilisation de descripteurs de message dans les exits API

Vous pouvez contrôler les propriétés de message auxquelles un exit API a accès. Les propriétés sont associées à un descripteur ExitMsg. Les propriétés définies dans un exit d'insertion sont définies dans le message en cours d'insertion, mais les propriétés extraites dans un exit d'extraction ne sont pas renvoyées à l'application.

Lorsque vous enregistrez une fonction d'exit MQ_INIT_EXIT à l'aide de l'appel MQI MQXEP avec **Function** défini sur MQXF_INIT et **ExitReason** défini sur MQXR_CONNECTION, vous transmettez une structure MQXEPO en tant que paramètre **ExitOpts** . La structure MQXEPO contient la zone ExitProperties , qui spécifie l'ensemble de propriétés à mettre à la disposition de l'exit. Elle est indiquée sous la forme d'une chaîne de caractères représentant le préfixe des propriétés, qui correspond à un nom de dossier MQRFH2 .

Chaque exit API reçoit une structure MQAXP contenant une zone de descripteur ExitMsg. Cette zone est définie sur une valeur générée par IBM MQ et est spécifique à une connexion. Le descripteur est donc inchangé entre les exits API de mêmes types ou de types différents sur la même connexion.

Dans une instruction MQ_PUT_EXIT ou MQ_PUT1_EXIT avec un **ExitReason** de MQXR_BEFORE, c'est-à-dire un exit d'API exécuté avant d'insérer un message, toutes les propriétés (autres que les propriétés de descripteur de message) associées à l'instruction ExitMsgHandle lorsque l'exit est terminé sont définies sur le message en cours d'insertion. Pour éviter que cela ne se produise, définissez le descripteur ExitMsgsur MQHM_NONE. Vous pouvez également fournir un autre descripteur de message.

Dans MQ_GET_EXIT et MQ_CALLBACK_EXIT, le descripteur ExitMsgest effacé des propriétés et rempli avec les propriétés spécifiées dans la zone ExitProperties lors de l'enregistrement de MQ_INIT_EXIT, à l'exception des propriétés de descripteur de message. Ces propriétés ne sont pas mises à la disposition de l'application d'obtention. Si l'application d'extraction a spécifié un descripteur de message dans la zone MQGMO (Get message options), toutes les propriétés associées à ce descripteur, y compris les propriétés de descripteur de message, sont disponibles pour l'exit API. Pour éviter que le descripteur ExitMsgne soit rempli avec des propriétés, définissez-le sur MQHM_NONE.

Un exemple de programme, amqsaem0.c, est fourni pour illustrer l'utilisation des descripteurs de message dans les exits API.

Référence associée

Références des exits utilisateur, des exits API et des services installables

Multi *Compilation des exits API*

Une fois que vous avez écrit un exit, vous le compilez et le liez comme suit.

Les exemples suivants présentent les commandes utilisées pour l'exemple de programme décrit dans «Exemple de programme d'exit API», à la page 1106. Pour les plateformes autres que les systèmes Windows , vous trouverez l'exemple de code d'exit d'API dans *MQ_INSTALLATION_PATH*/samp et la bibliothèque partagée compilée et liée dans *MQ_INSTALLATION_PATH*/samp/bin.

Windows Pour les systèmes Windows , vous trouverez l'exemple de code d'exit d'API dans *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ a été installé.

Remarque : Les conseils relatifs à la programmation des applications 64 bits sont répertoriés dans la rubrique Normes de codage sur les plateformes 64 bits.

Pour les clients multidiffusion, les exits API et les exits de conversion de données doivent pouvoir s'exécuter côté client car certains messages risquent de ne pas passer par le gestionnaire de files d'attente. Les bibliothèques suivantes font partie des packages client ainsi que des packages serveur:

Tableau 141. Bibliothèques qui se trouvent dans les packages client et serveur

Système d'exploitation	Bibliothèques
AIX AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
IBM i IBM i	LIBMQM & LIBMQM_R
Linux Linux	32 bits & 64 bits: libmqm.so & libmqm_r.so
Windows Windows	32 bits & 64 bits: mqm.dll & mqm.pdb

Linux AIX *Compilation des exits API sur les systèmes AIX and Linux*

Exemples de compilation des exits API sur les systèmes AIX and Linux .

Sur toutes les plateformes, le point d'entrée du module est MQStart.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

sur AIX

AIX

Compilez le code source de l'exit API en exécutant l'une des commandes suivantes:

Applications 32 bits

Non unités d'exécution

```
cc -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
cc -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
amqsaxe0.c -I MQ_INSTALLATION_PATH/inc
```

sur Linux

Linux

Compilez le code source de l'exit API en exécutant l'une des commandes suivantes:

Applications 31 bits

Non unités d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Applications 32 bits

Non unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -I MQ_INSTALLATION_PATH/inc
```

Windows *Compilation des exits API sur les systèmes Windows*

Compilez et liez l'exemple de programme d'exit API, `amqsaxe0.c`, sous Windows

Un fichier manifeste est un document XML facultatif contenant la version ou toute autre information pouvant être imbriquée dans une application compilée ou une DLL.

Si vous n'avez pas de document de ce type, omettez le paramètre -manifest *manifest.file* dans la commande **mt** .

Adaptez les commandes des exemples dans Figure 101, à la page 985 ou Figure 102, à la page 985 pour compiler et lier `amqsaxe0.c` sous Windows. Les commandes fonctionnent avec Microsoft Visual Studio 2008, 2010 ou 2012. Les exemples supposent que le répertoire `C:\Program Files\IBM\MQ\tools\c\samples` est le répertoire en cours.

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
```

Figure 101. Compilez et liez amqsaxe0.c sur Windows 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
```

Figure 102. Compilez et liez amqsaxe0.c sur un système Windows 64 bits

Concepts associés

«Exemple de programme d'exit API», à la page 1106
L'exemple d'exit API génère une trace MQI dans un fichier spécifié par l'utilisateur avec un préfixe défini dans la variable d'environnement **MQAPI_TRACE_LOGFILE** .

IBM i *Exits d'API de conformité sous IBM i*
Compilation des exits API sous IBM i.

Un exit est créé comme suit (pour un exemple de langage C):

1. Créez un module à l'aide de la commande CRTCMOD. Compilez-le pour utiliser l'espace mémoire à téraoctets en incluant le paramètre TERASPACE(*YES *TSIFC).
2. Créez un programme de service à partir du module à l'aide de la commande CRTSRVPGM. Vous devez le lier au programme de service QMQM/LIBMQMZF_R pour les exits API à unités d'exécution multiples.

Configuration des exits API

Vous configurez IBM MQ pour activer les exits API en modifiant les informations de configuration.

Pour modifier les informations de configuration, vous devez modifier les sections qui définissent les routines d'exit et la séquence dans laquelle elles s'exécutent. Ces informations peuvent être modifiées comme suit:

- Windows Linux Utilisation de IBM MQ Explorer sous Windows et Linux (plateformesx86 et x86-64).
- Windows Utilisation de la commande **amqmdain** sous Windows.

- Multi Utilisation des fichiers `mqs.ini` et `qm.ini` directement sur Multiplatforms.

  Le fichier `mqs.ini` contient des informations relatives à tous les gestionnaires de files d'attente d'un noeud particulier. Vous pouvez le trouver dans les emplacements suivants:

  – Linux AIX Dans le répertoire `/var/mqm` sous AIX and Linux.

  – Windows Dans le chemin WorkPath spécifié dans la clé `HKLM\SOFTWARE\IBM\WebSphere MQ` sur les systèmes Windows .

  – IBM i Dans le répertoire `/QIBM/UserData/mqm` sous IBM i.

  Le fichier `qm.ini` contient des informations relatives à un gestionnaire de files d'attente spécifique. Il existe un fichier de configuration de gestionnaire de files d'attente pour chaque gestionnaire de files d'attente, stocké à la racine de l'arborescence de répertoires occupée par le gestionnaire de files d'attente. Par exemple, le chemin et le nom d'un fichier de configuration pour un gestionnaire de files d'attente appelé QMNAME sont les suivants:

  IBM i Sur les systèmes IBM i :

  ```
  /QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini
  ```

  Linux AIX Sur les systèmes AIX and Linux :

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows Sur les systèmes Windows :

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  Avant d'éditer un fichier de configuration, sauvegardez ce fichier afin de disposer d'une copie à laquelle vous pouvez revenir si nécessaire.

  Vous pouvez éditer les fichiers de configuration de l'une des manières suivantes:

  – Automatiquement, à l'aide de commandes qui modifient la configuration des gestionnaires de files d'attente sur le noeud.

  – Manuellement, à l'aide d'un éditeur de texte standard.

  Si vous définissez une valeur incorrecte pour un attribut de fichier de configuration, la valeur est ignorée et un message d'opérateur est émis pour indiquer le problème. L'effet est identique à l'absence complète de l'attribut.

Sections à configurer

Les sections qui doivent être modifiées sont les suivantes:

ApiExitCommon

Défini dans `mqs.ini` et dans IBM MQ Explorer sur la page de propriétés IBM MQ , sous Exits.

Lorsqu'un gestionnaire de files d'attente démarre, les attributs de cette section sont lus, puis remplacés par les exits API définis dans `qm.ini`.

ApiExitTemplate

Défini dans `mqs.ini` et dans IBM MQ Explorer sur la page de propriétés IBM MQ , sous Exits.

Lorsqu'un gestionnaire de files d'attente est créé, les attributs de cette section sont copiés dans le fichier qm.ini nouvellement créé sous la section locale ApiExit.

ApiExitLocal

Défini dans `qm.ini` et dans IBM MQ Explorer sur la page des propriétés du gestionnaire de files d'attente, sous Exits.

Lorsque le gestionnaire de files d'attente démarre, les exits API définis ici remplacent les valeurs par défaut définies dans `mqs.ini`.

Attributs pour les strophes

- Nommez l'exit API à l'aide de l'attribut suivant:

  Name=nom_exit_pix
  Nom descriptif de l'exit API qui lui a été transmis dans la zone ExitInfoNom de la structure MQAXP.

  Ce nom doit être unique, ne pas dépasser 48 caractères et contenir uniquement des caractères valides pour les noms d'objets IBM MQ (par exemple, les noms de file d'attente).

- Identifiez le module et le point d'entrée du code d'exit API à exécuter à l'aide des attributs suivants:

  Function=nom_fonction
  Nom du point d'entrée de fonction dans le module contenant le code d'exit d'API. Ce point d'entrée est la fonction MQ_INIT_EXIT.

  La longueur de cette zone est limitée à MQ_EXIT_NAME_LENGTH.

  Module=nom_module
  Module contenant le code d'exit d'API.

  Ce champ est utilisé tel quel s'il contient le chemin d'accès complet au module.

  Si cette zone contient uniquement le nom du module, le module est localisé à l'aide de l'attribut `ExitsDefaultPath` dans le fichier `ExitPath` dans qm.ini.

  Sur les plateformes qui prennent en charge des bibliothèques à unités d'exécution distinctes, vous devez fournir à la fois une version non à unités d'exécution et une version à unités d'exécution du module d'exit API. La version à unités d'exécution doit avoir un suffixe `_r` . La version à unités d'exécution du module de remplacement d'application IBM MQ ajoute implicitement `_r` au nom de module donné avant son chargement.

  La longueur de cette zone est limitée à la longueur de chemin maximale prise en charge par la plateforme.

- Vous pouvez éventuellement transmettre des données avec l'exit à l'aide de l'attribut suivant:

  Data=nom_données
  Données à transmettre à l'exit API dans la zone ExitData de la structure MQAXP.

  Si vous incluez cet attribut, les blancs de début et de fin sont supprimés, la chaîne restante est tronquée à 32 caractères et le résultat est transmis à l'exit. Si vous omettez cet attribut, la valeur par défaut de 32 blancs est transmise à l'exit.

  La longueur maximale de cette zone est de 32 caractères.

- Identifiez la séquence de cet exit par rapport aux autres exits à l'aide de l'attribut suivant:

  Sequence=numéro_séquence
  Séquence dans laquelle cet exit d'API est appelé par rapport à d'autres exits d'API. Un exit avec un numéro de séquence faible est appelé avant un exit avec un numéro de séquence plus élevé. Il n'est pas nécessaire que la numérotation de séquence des exits soit contiguë. Une séquence de 1, 2, 3 a le même résultat qu'une séquence de 7, 42, 1096. Si deux exits ont le même numéro de séquence, le gestionnaire de files d'attente décide lequel appeler en premier. Vous pouvez déterminer qui a été appelé après l'événement en plaçant l'heure ou un marqueur dans la zone ExitChainindiquée par ExitChainAreaPtr dans MQAXP ou en écrivant votre propre fichier journal.

  Cet attribut est une valeur numérique non signée.

Exemples de strophes

L'exemple de fichier mqs.ini contient les strophes suivantes:

ApiExitTemplate
Cette section définit un exit avec le nom descriptif `OurPayrollQueueAuditor`, le nom de module `auditor`et le numéro de séquence 2. Une valeur de données de 123 est transmise à l'exit.

ApiExitCommon
Cette section définit un exit avec le nom descriptif `MQPoliceman`, le nom du module `tmqp`et le numéro de séquence 1. Les données transmises sont une instruction (`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
```

L'exemple de fichier qm.ini suivant contient une définition locale ApiExitd'un exit avec le nom descriptif `ClientApplicationAPIchecker`, le nom du module `ClientAppChecker`et le numéro de séquence 3.

```
qm.ini

   ApiExitLocal:
      Name=ClientApplicationAPIchecker
      Sequence=3
      Function=EntryPoint
      Module=/usr/Dev/ClientAppChecker
      Data=9.20.176.20
```

Programmes d'exit de canal pour les canaux de messagerie

Cette collection de rubriques contient des informations sur les programmes d'exit de canal IBM MQ pour les canaux de messagerie.

Les agents MCA (Message Channel Agent) peuvent également appeler des exits de conversion de données. Pour plus d'informations sur l'écriture des exits de conversion de données, voir «Ecriture des exits de conversion de données», à la page 1010.

Certaines de ces informations s'appliquent également aux exits sur les canaux MQI, qui connectent IBM MQ MQI clients aux gestionnaires de files d'attente. Pour plus d'informations, voir Programmes d'exit de canal pour les canaux MQI.

Les programmes d'exit de canal sont appelés à des emplacements définis dans le traitement effectué par les programmes MCA.

Certains de ces programmes d'exit utilisateur fonctionnent par paires complémentaires. Par exemple, si un programme d'exit utilisateur est appelé par l'agent MCA émetteur pour chiffrer les messages à transmettre, le processus complémentaire doit fonctionner à l'extrémité réceptrice pour inverser le processus.

Le Tableau 142, à la page 989 présente les types d'exit de canal disponibles pour chaque type de canal.

Tableau 142. Exits de canal disponibles pour chaque type de canal

Type de canal	Exit de message	Exit de relance de message	Exit de réception	Exit de sécurité	Exit d'émission	Exit de définition automatique
Canal émetteur	Oui		Oui	Oui	Oui	
Canal serveur	Oui		Oui	Oui	Oui	
Canal émetteur de cluster	Oui		Oui	Oui	Oui	Oui
Canal récepteur	Oui	Oui	Oui	Oui	Oui	Oui
Canal demandeur	Oui	Oui	Oui	Oui	Oui	
Canal récepteur de cluster	Oui	Oui	Oui	Oui	Oui	Oui
Canal de connexion client			Oui	Oui	Oui	
Canal de connexion serveur			Oui	Oui	Oui	Oui

Remarques : z/OS

1. Sous z/OS, l'exit de définition automatique s'applique uniquement aux canaux émetteurs et récepteurs de cluster.

Si vous prévoyez d'exécuter des exits de canal sur un client, vous ne pouvez pas utiliser la variable d'environnement MQSERVER. A la place, créez et référencez une table de définition de canal du client (CCDT) comme décrit dans Table de définition de canal du client.

Présentation du traitement

Présentation de la façon dont les agents MCA utilisent les programmes d'exit de canal.

Au démarrage, les agents MCA échangent une boîte de dialogue de démarrage pour synchroniser le traitement. Ensuite, ils basculent vers un échange de données qui inclut les exits de sécurité. Ces exits doivent se terminer correctement pour que la phase de démarrage se termine et pour permettre le transfert des messages.

La phase de contrôle de sécurité est une boucle, comme illustré dans la Figure 103, à la page 990.

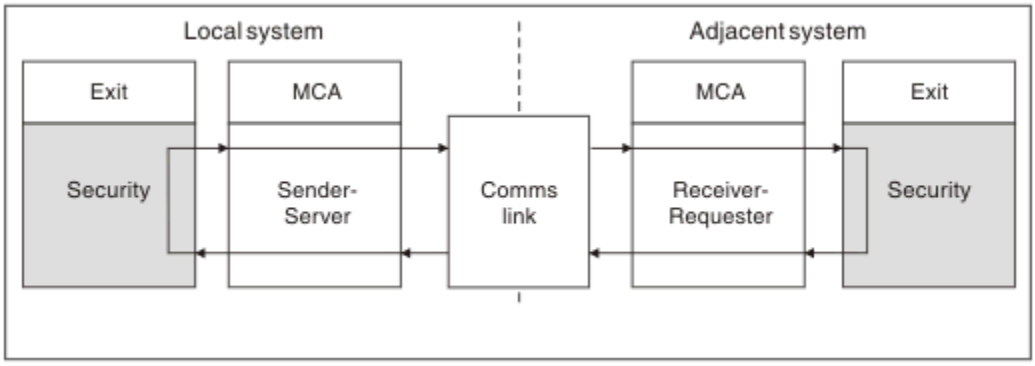


Figure 103. Boucle d'exit de sécurité

Au cours de la phase de transfert de message, l'agent MCA émetteur extrait des messages d'une file d'attente de transmission, appelle l'exit de message, appelle l'exit d'émission, puis envoie le message à l'agent MCA récepteur, comme illustré dans la Figure 104, à la page 990.

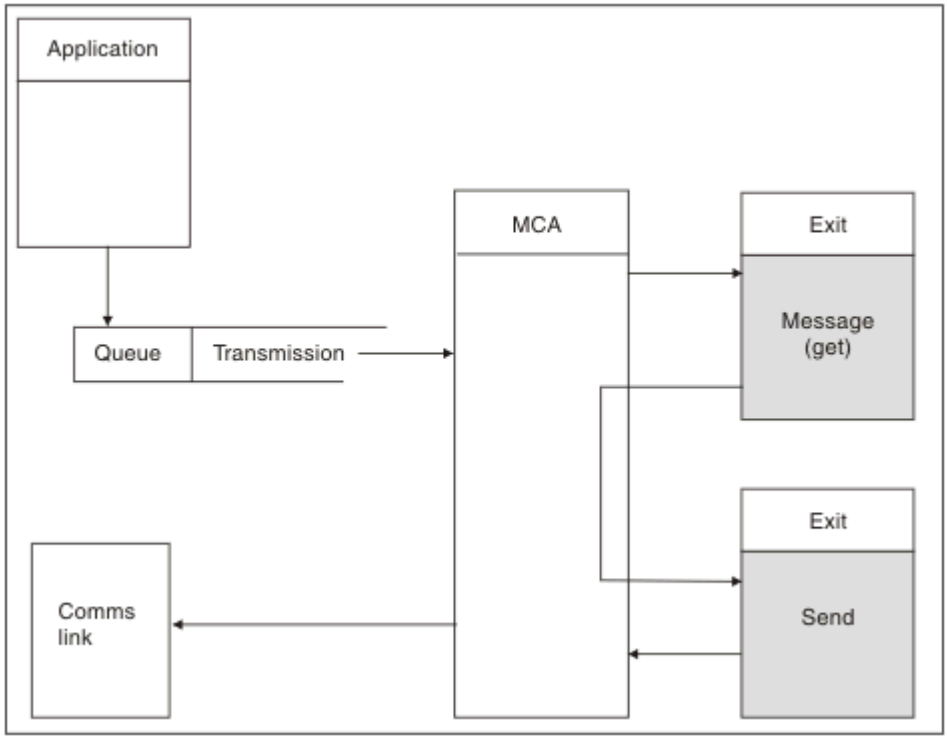


Figure 104. Exemple d'exit d'émission à l'extrémité émettrice du canal de transmission de messages

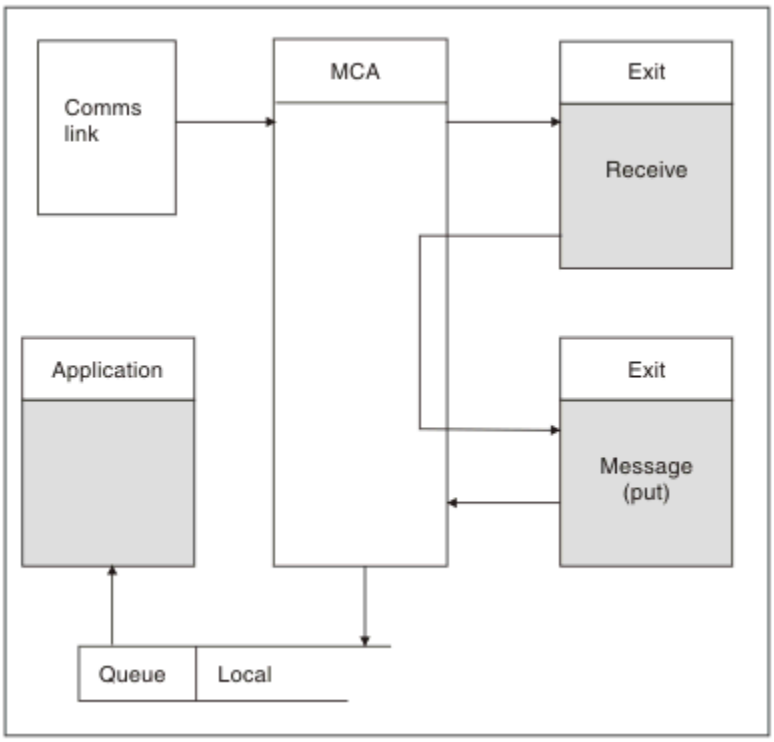


Figure 105. Exemple d'exit de réception à l'extrémité réceptrice du canal de transmission de messages

L'agent MCA récepteur reçoit un message de la liaison de communication, appelle l'exit de réception, appelle l'exit de message, puis place le message dans la file d'attente locale, comme illustré dans la Figure 105, à la page 991. (L'exit de réception peut être appelé plusieurs fois avant l'appel de l'exit de message.)

Ecriture de programmes d'exit de canal

Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal.

Les exits utilisateur et les programmes d'exit de canal peuvent utiliser tous les appels MQI, sauf comme indiqué dans les sections suivantes. Pour MQ V7 et versions ultérieures, la structure MQCXP version 7 et ultérieure contient le descripteur de connexion hConn, qui peut être utilisé à la place de l'émission de MQCONN. Pour les versions antérieures, afin d'obtenir le descripteur de connexion, un MQCONN doit être émis, même si un avertissement MQRC_ALREADY_CONNECTED est renvoyé car le canal lui-même est connecté au gestionnaire de files d'attente.

Notez que l'exit de canal doit être autorisant les unités d'exécution multiples.

Pour les exits sur les canaux de connexion client, le gestionnaire de files d'attente auquel l'exit tente de se connecter dépend de la manière dont l'exit a été lié. Si l'exit a été lié à MQM.LIB (ou QMQM/LIBMQM sous IBM i) et que vous n'indiquez pas de nom de gestionnaire de files d'attente dans l'appel MQCONN, l'exit tente de se connecter au gestionnaire de files d'attente par défaut sur votre système. Si l'exit a été lié à MQM.LIB (ou QMQM/LIBMQM sous IBM i) et que vous spécifiez le nom du gestionnaire de files d'attente qui a été transmis à l'exit via la zone QMgrName de MQCD, l'exit tente de se connecter à ce gestionnaire de files d'attente. Si l'exit a été lié à MQIC.LIB ou toute autre bibliothèque, l'appel MQCONN échoue que vous indiquiez un nom de gestionnaire de files d'attente ou non.

Vous devez éviter de modifier l'état de la transaction associée au hConn transmis dans un exit de canal ; vous ne devez pas utiliser les instructions MQCMIT, MQBACK ou MQDISC avec le canal hConnet vous ne pouvez pas utiliser l'instruction MQBEGIN en spécifiant le canal hConn.

Si MQCONNX est utilisé en spécifiant MQCNO_HANDLE_SHARE_BLOCK ou MQCNO_HANDLE_SHARE_NO_BLOCK pour créer une nouvelle connexion IBM MQ , vous devez vous assurer que la connexion est correctement gérée et qu'elle se déconnecte correctement du gestionnaire

de files d'attente. Par exemple, un exit de canal qui crée une nouvelle connexion au gestionnaire de files d'attente à chaque appel sans se déconnecter entraîne la création de descripteurs de connexion et une augmentation du nombre d'unités d'exécution d'agent.

Un exit s'exécute dans la même unité d'exécution que l'agent MCA lui-même et utilise le même descripteur de connexion. Il s'exécute donc dans la même unité de travail que l'agent MCA et tous les appels passés sous le point de synchronisation sont validés ou annulés par le canal à la fin du lot.

Par conséquent, un exit de message de canal peut envoyer des messages de notification qui ne sont validés dans cette file d'attente que lorsque le lot contenant le message d'origine est validé. Il est donc possible d'émettre des appels MQI de point de synchronisation à partir d'un exit de message de canal.

Un exit de canal peut modifier les zones de la base de données MQCD. Toutefois, ces modifications ne sont pas prises en compte, sauf dans les circonstances énumérées. Si un programme d'exit de canal modifie une zone dans la structure de données MQCD, la nouvelle valeur est ignorée par le processus de canal IBM MQ . Toutefois, la nouvelle valeur reste dans le MQCD et est transmise aux exits restants d'une chaîne d'exit et à toute conversation partageant l'instance de canal. Pour plus d'informations, voir Modification des zones MQCD dans un exit de canal

En outre, pour les programmes écrits en C, la fonction de bibliothèque C non réentrante ne doit pas être utilisée dans un programme d'exit de canal.

Si vous utilisez plusieurs bibliothèques d'exit de canal simultanément, des problèmes peuvent survenir sur certaines plateformes UNIX and Linux si le code de deux exits différents contient des fonctions portant le même nom. Lorsqu'un exit de canal est chargé, le chargeur dynamique résout les noms de fonction dans la bibliothèque d'exit en indiquant les adresses où la bibliothèque est chargée. Si deux bibliothèques d'exit définissent des fonctions distinctes qui ont des noms identiques, ce processus de résolution peut résoudre de manière incorrecte les noms de fonction d'une bibliothèque pour utiliser les fonctions d'une autre. Si ce problème se produit, indiquez à l'éditeur de liens qu'il doit uniquement exporter les fonctions d'exit et MQStart requises, car ces fonctions ne sont pas affectées. Les autres fonctions doivent bénéficier d'une visibilité locale afin qu'elles ne soient pas utilisées par des fonctions en dehors de leur propre bibliothèque d'exit. Pour plus d'informations, consultez la documentation de l'éditeur de liens.

Tous les exits sont appelés avec une structure de paramètres d'exit de canal (MQCXP), une structure de définition de canal (MQCD), une mémoire tampon de données préparée, un paramètre de longueur de données et un paramètre de longueur de mémoire tampon. La longueur de la mémoire tampon ne doit pas être dépassée:

- Pour les exits de message, vous devez autoriser l'envoi du message le plus volumineux sur le canal, plus la longueur de la structure MQXQH.
- Pour les exits d'émission et de réception, la mémoire tampon la plus importante que vous devez autoriser est la suivante:

  LU 6.2
  32 ko

  TCP:
  

  IBM i 16 ko

  Autres 32 Ko

  Remarque : La longueur maximale utilisable peut être inférieure de 2 octets à cette longueur. Vérifiez la valeur renvoyée dans **MaxSegmentLength** pour plus de détails. Pour plus d'informations sur **MaxSegmentLength**, voir MaxSegmentLength.

  NetBIOS:
  64 ko

  SPX:
  64 ko

Remarque : Les exits de réception sur les canaux émetteurs et les exits d'émission sur les canaux récepteurs utilisent des mémoires tampon de 2 Ko pour TCP.

- Pour les exits de sécurité, la fonction de mise en file d'attente répartie alloue une mémoire tampon de 4000 octets.

Il est permis à l'exit de renvoyer une autre mémoire tampon, ainsi que les paramètres pertinents. Pour plus d'informations sur les appels, voir «Programmes d'exit de canal pour les canaux de messagerie», à la page 988 .

z/OS *Writing channel exit programs on z/OS*

You can use the following information to help you write and compile channel-exit programs for z/OS.

The exits are started as if by a z/OS LINK, in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode

The link-edited modules must be placed in the data set specified by the CSQXLIB DD statement of the channel initiator address space procedure; the names of the load modules are specified as the exit names in the channel definition.

When writing channel exits for z/OS, the following rules apply:

- Exits must be written in assembler or C; if C is used, it must conform to the C systems programming environment for system exits, described in the z/OS C/C++ Programming Guide.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the channel initiator is running. The new version is used when the channel is restarted.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment, on return, to that at entry.
- Exits must free any storage obtained, or ensure that it is freed by a subsequent exit invocation.

  For storage that is to persist between invocations, use the z/OS STORAGE service, or the 4kmalc library function for System Programming C.

  For more information about this function, see 4kmalc() -- Allocate Page-Aligned Storage.
- All IBM MQ MQI calls except MQCMIT or CSQBCMT and MQBACK or CSQBBAK can be used. They must be contained after MQCONN (with a blank queue manager name). If these calls are used, the exit must be link-edited with the stub CSQXSTUB.

  The exception to this rule is that security channel exits can issue commit and backout MQI calls. To issue such calls, code the verbs CSQXCMT and CSQXBAK in place of MQCMIT or CSQBCMT and MQBACK or CSQBBAK.
- All exits that use stub CSQXSTUB from IBM WebSphere MQ 7.0 or later must be link-edited in a CSQXLIB load library with format PDS-E.
- Exits must not use any system services that cause a wait, because using system services would severely affect the handling of some or all the other channels. Many channels are run under a single TCB typically. If you do something in an exit that causes a wait and you do not use MQXWAIT, it causes all these channels to wait. Causing channels to wait does not give any functional problems, but might have an adverse effect on performance. Most SVCs involve waits, so you must avoid them, except for the following SVCs:
  - GETMAIN/FREEMAIN/STORAGE
  - LOAD/DELETE

In general, therefore, avoid SVCs, PCs, and I/O. Instead, use the MQXWAIT call.

- Exits do not issue ESTAEs or SPIEs, apart from in any subtasks they attach, because their error handling might interfere with the error handling performed by IBM MQ. This means that IBM MQ might not be able to recover from an error, or that your exit program might not receive all the error information.
- The MQXWAIT call (see MQXWAIT) provides a wait service that waits for I/O and other events; if this service is used, exits must not use the linkage stack.

  For I/O and other facilities that do not provide non-blocking facilities or an ECB to wait on, a separate subtask must be ATTACHed, and its completion waited for by MQXWAIT; because of the processing that this technique incurs, this facility must be used only by the security exit.

- The MQDISC MQI call does not cause an implicit commit to occur within the exit program. A commit of the channel process is performed only when the channel protocol dictates.

The following exit samples are provided with IBM MQ for z/OS:

CSQ4BAX0
This sample is written in assembler, and illustrates the use of MQXWAIT.

CSQ4BCX1 and CSQ4BCX2
These samples are written in C and illustrate how to access the parameters.

CSQ4BCX3 and CSQ4BAX3
These samples are written in C and assembler respectively.

The CSQ4BCX3 sample (which is pre-compiled into the SCSQAUTH LOADLIB, should function with no changes necessary on the exit itself. You can create a LOADLIB (for example, called MY.TEST.LOADLIB) and copy the SCSQAUTH(CSQ4BCX3) member to it.

To set up a security exit on a client connection, carry out the following procedure:

1. Establish a valid OMVS segment for the user ID that the channel initiator uses.

   This allows the IBM MQ for z/OS channel initiator to use TCP/IP with the z/OS UNIX System Services (z/OS UNIX) socket interface, in order to facilitate exit processing. Note that it is unnecessary to define an OMVS segment for the user ID of any connecting client.
2. Ensure that the exit code itself runs only in a program controlled environment.

   This means everything loaded into the CHINIT address space must be loaded from a program controlled library (meaning all libraries in the STEPLIB), and any libraries named on CSQXLIB and

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   To set a load library as program controlled, use a command similar to this example:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

   Then you can activate or refresh the program controlled environment by issuing the command:

   ```
   SETROPTS WHEN(PROGRAM) REFRESH
   ```
3. Add the exit LOADLIB to the CSQXLIB DD (in the CHINIT started procedure), by issuing the following command:

   ```
   ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
   ```

   This activates the exit for the named channel.
4. Your external security manager (ESM) lists any other libraries to be program controlled, but note that none of the ESM or C libraries needs to be under program control.

See IBM MQ for z/OS server connection channel for further information on setting up a security exit using the sample CSQ4BCX3.

CSQ4BCX4
This sample is written in C and demonstrates using the **RemoteProduct** and **RemoteVersion** fields in MQCXP.

Related concepts

“Ecriture de programmes d'exit de canal sous IBM i” on page 995
Vous pouvez utiliser les informations suivantes pour vous aider à écrire et à compiler des programmes d'exit de canal pour IBM i.

“Ecriture de programmes d'exit de canal sous AIX, Linux, and Windows” on page 996
Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal pour les systèmes AIX, Linux, and Windows .

Related reference

IBM MQ for z/OS server connection channel

IBM i *Ecriture de programmes d'exit de canal sous IBM i*

Vous pouvez utiliser les informations suivantes pour vous aider à écrire et à compiler des programmes d'exit de canal pour IBM i.

L'exit est un objet programme écrit en langage ILE C, ILE RPG ou ILE COBOL. Les noms des programmes d'exit et leurs bibliothèques sont nommés dans la définition de canal.

Respectez les conditions suivantes lors de la création et de la compilation d'un programme d'exit:

- Le programme doit être sécurisé par unité d'exécution et créé avec le compilateur ILE C, ILE RPG ou ILE COBOL. Pour ILE RPG, vous devez indiquer la spécification de contrôle THREAD (*SERIALIZE) et pour ILE COBOL, vous devez indiquer SERIALIZE pour l'option THREAD de l'instruction PROCESS. Les programmes doivent également être liés aux bibliothèques IBM MQ à unités d'exécution: QMQM/LIBMQM_R dans le cas d'ILE C et ILE RPG et AMQ0STUB_R dans le cas d'ILE COBOL. Pour plus d'informations sur la sécurisation des unités d'exécution des applications RPG ou COBOL, reportez-vous au manuel Programmer's Guide correspondant au langage.
- IBM MQ for IBM i requiert que les programmes d'exit soient activés pour la prise en charge de l'espace mémoire à téraoctets. (Teraspace est une forme de mémoire partagée introduite dans OS/400 V4R4.) Pour les compilateurs ILE RPG et COBOL, tous les programmes compilés sous OS/400 V4R4 ou version ultérieure sont activés. Pour C, les programmes doivent être compilés avec les options TERASPACE (*YES *TSIFC) indiquées dans les commandes CRTCMOD ou CRTBNDC.
- Un exit qui renvoie un pointeur vers son propre espace de mémoire tampon doit s'assurer que l'objet désigné existe au-delà de l'intervalle de temps du programme d'exit de canal. Le pointeur ne peut pas être l'adresse d'une variable de la pile du programme, ni d'une variable de la pile du programme. Au lieu de cela, le pointeur doit être obtenu à partir du système. Par exemple, un espace utilisateur créé dans l'exit utilisateur. Pour s'assurer que toute zone de données allouée par le programme d'exit de canal est toujours disponible pour l'agent MCA à l'arrêt du programme, l'exit de canal doit être exécuté dans le groupe d'activation de l'appelant ou dans un groupe d'activation nommé. Pour ce faire, définissez le paramètre ACTGRP de CRTPGM sur une valeur définie par l'utilisateur ou *CALLER. Si le programme est ainsi créé, le programme d'exit de canal peut allouer de la mémoire dynamique et transmettre un pointeur à cette mémoire à l'agent MCA.

Concepts associés

«Ecriture de programmes d'exit de canal sous AIX, Linux, and Windows», à la page 996
Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal pour les systèmes AIX, Linux, and Windows .

«Writing channel exit programs on z/OS », à la page 993
You can use the following information to help you write and compile channel-exit programs for z/OS.

ALW *Ecriture de programmes d'exit de canal sous AIX, Linux, and Windows*

Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal pour les systèmes AIX, Linux, and Windows .

Suivez les instructions décrites dans «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962. Utilisez les informations spécifiques à l'exit de canal suivantes, le cas échéant:

L'exit doit être écrit en C et il s'agit d'une DLL sous Windows.

Définissez une routine MQStart () factice dans l'exit et spécifiez `MQStart` comme point d'entrée dans la bibliothèque. Figure 106, à la page 996 montre comment configurer une entrée dans votre programme:

```
#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
}
```

Figure 106. Exemple de code source pour un exit de canal

Lorsque vous écrivez des exits de canal pour Windows à l'aide de Visual C + +, vous devez écrire votre propre fichier DEF . Un exemple de la façon dont il est présenté dans la Figure 107, à la page 996. Pour plus d'informations sur l'écriture de programmes d'exit de canal, voir «Ecriture de programmes d'exit de canal», à la page 991.

```
EXPORTS
ChannelExit
```

Figure 107. Exemple de fichier DEF pour Windows

Concepts associés

«Ecriture de programmes d'exit de canal sous IBM i», à la page 995
Vous pouvez utiliser les informations suivantes pour vous aider à écrire et à compiler des programmes d'exit de canal pour IBM i.

«Writing channel exit programs on z/OS », à la page 993
You can use the following information to help you write and compile channel-exit programs for z/OS.

Programmes d'exit de sécurité de canal

Vous pouvez utiliser des programmes d'exit de sécurité pour vérifier que le partenaire à l'autre extrémité d'un canal est authentique. C'est ce qu'on appelle l'authentification.

Pour indiquer qu'un canal doit utiliser un exit de sécurité, indiquez le nom de l'exit dans la zone **SCYEXIT** de la définition de canal.

Remarque : L'authentification peut également être effectuée à l'aide d'enregistrements d'authentification de canal. Les enregistrements d'authentification de canal offrent une grande souplesse pour empêcher l'accès aux gestionnaires de files d'attente à partir de certains utilisateurs et canaux et pour mapper les utilisateurs distants vers les identificateurs d'utilisateur IBM MQ . La prise en charge de TLS est également fournie par IBM MQ pour authentifier vos utilisateurs et pour fournir des contrôles de chiffrement et d'intégrité des données pour vos données. Pour plus d'informations sur TLS, voir Protocoles de sécurité TLS dans IBM MQ. Toutefois, si vous avez encore besoin de formes de traitement de sécurité plus sophistiquées (ou différentes) et d'autres types de contrôle et d'établissement de contexte de sécurité, pensez à écrire des exits de sécurité.

Les attributs de nom distinctif de sujet et d'émetteur apparaissent dans les attributs de statut de canal suivants:

- SSLPEER (sélecteur PCF MQCACH_SSL_SHORT_PEER_NAME)
- SSLCERTI (sélecteur PCF MQCACH_SSL_CERT_ISSUER_NAME)

Ces valeurs sont renvoyées par les commandes de statut de canal ainsi que les données transmises aux exits de sécurité de canal répertoriés, comme indiqué:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

Un exit de sécurité peut être écrit en C ou Java.

Les programmes d'exit de sécurité de canal sont appelés aux emplacements suivants du cycle de traitement d'un agent MCA:

- Au début et à la fin de l'agent MCA.
- Immédiatement après la fin de la négociation de données initiale au démarrage du canal. L'extrémité réceptrice ou serveur du canal peut initier un échange de messages de sécurité avec l'extrémité distante en fournissant un message à distribuer à l'exit de sécurité à l'extrémité distante. Elle pourrait également refuser de le faire. Le programme d'exit est redémarré pour traiter tout message de sécurité reçu de l'extrémité éloignée.
- Immédiatement après la fin de la négociation de données initiale au démarrage du canal. L'extrémité émettrice ou demandeur du canal traite un message de sécurité reçu de l'extrémité distante ou lance un échange de sécurité lorsque l'extrémité distante ne peut pas. Le programme d'exit est redémarré pour traiter tous les messages de sécurité suivants qui peuvent être reçus.

Un canal demandeur n'est jamais appelé avec MQXR_INIT_SEC. Le canal indique au serveur qu'il dispose d'un programme d'exit de sécurité, puis le serveur a la possibilité de lancer un exit de sécurité. S'il n'en a pas, il en informe le demandeur et un flux de longueur nulle est renvoyé au programme d'exit.

Remarque : Evitez d'envoyer des messages de sécurité de longueur nulle.

Des exemples de données échangées par les programmes d'exit de sécurité sont illustrés dans les figures Figure 108, à la page 998 à Figure 111, à la page 1000. Ces exemples montrent la séquence des événements qui se produisent impliquant l'exit de sécurité du récepteur et l'exit de sécurité de l'expéditeur. Des rangées successives sur les figures représentent le passage du temps. Dans certains cas, les événements au niveau du récepteur et de l'émetteur ne sont pas corrélés, et peuvent donc se produire en même temps ou à des moments différents. Dans d'autres cas, un événement au niveau d'un programme d'exit se traduit par un événement complémentaire qui se produit plus tard au niveau de l'autre programme d'exit. Par exemple, dans Figure 108, à la page 998:

1. Le récepteur et l'émetteur sont chacun appelés avec MQXR_INIT, mais ces appels ne sont pas corrélés et peuvent donc se produire en même temps ou à des moments différents.
2. Le récepteur est ensuite appelé avec MQXR_INIT_SEC, mais renvoie MQXCC_OK qui ne requiert aucun événement complémentaire à l'exit de l'expéditeur.
3. L'expéditeur est ensuite appelé avec MQXR_INIT_SEC. Ceci n'est pas corrélé avec l'appel du récepteur avec MQXR_INIT_SEC. L'émetteur renvoie MQXCC_SEND_SEC_MSG, ce qui provoque un événement complémentaire à l'exit du récepteur.
4. Le récepteur est ensuite appelé avec MQXR_SEC_MSG et renvoie MQXCC_SEND_SEC_MSG, ce qui provoque un événement complémentaire à l'exit de l'expéditeur.
5. L'émetteur est ensuite appelé avec MQXR_SEC_MSG et renvoie MQXCC_OK, qui ne requiert aucun événement complémentaire à l'exit du récepteur.

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Message transfer begins	

Figure 108. Echange initié par l'expéditeur avec accord

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

Figure 109. Echange initié par l'expéditeur sans accord

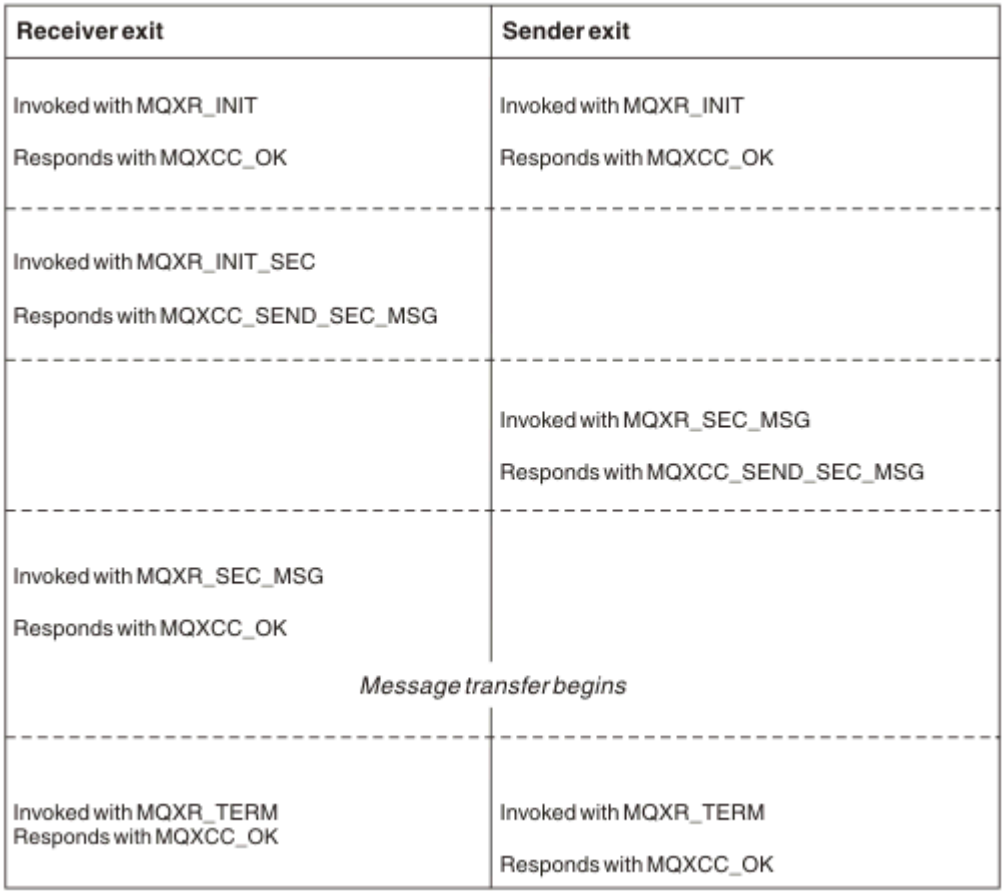


Figure 110. Echange initié par le récepteur avec accord

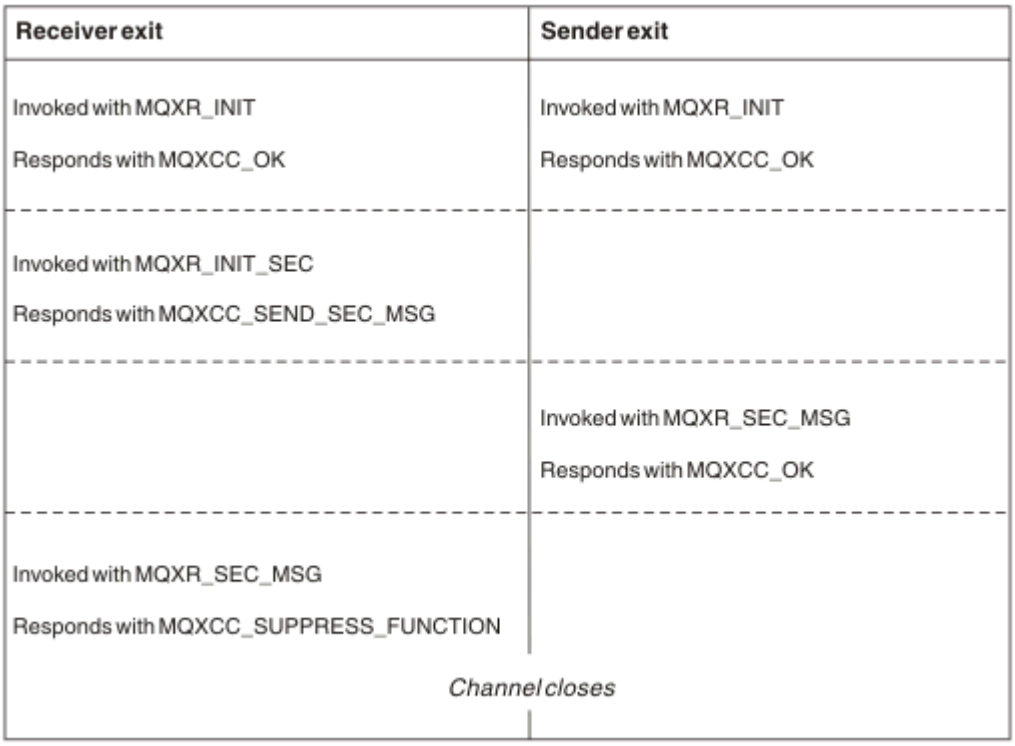


Figure 111. Echange initié par le récepteur sans accord

Le programme d'exit de sécurité de canal transmet une mémoire tampon d'agent contenant les données de sécurité, à l'exclusion des en-têtes de transmission, générées par l'exit de sécurité. Ces données peuvent être toutes les données appropriées pour que chaque extrémité du canal puisse effectuer une validation de sécurité.

Le programme d'exit de sécurité à l'extrémité émettrice et à l'extrémité réceptrice du canal de message peut renvoyer l'un ou l'autre des deux codes de réponse à n'importe quel appel:

- L'échange de sécurité s'est arrêté sans erreur
- Supprimer le canal et fermer

Remarque :

1. Les exits de sécurité de canal fonctionnent généralement par paires. Lorsque vous définissez les canaux appropriés, assurez-vous que les programmes d'exit compatibles sont nommés pour les deux extrémités du canal.
2. IBM i Dans IBM i , les programmes d'exit de sécurité qui ont été compilés avec `Use adopted authority` (USEADPAUT = *YES) peuvent adopter les droits QMQM ou QMQMADM. Prenez soin que l'exit n'utilise pas cette fonction pour présenter un risque de sécurité pour votre système.
3. Sur un canal TLS sur lequel l'autre extrémité du canal fournit un certificat, l'exit de sécurité reçoit le nom distinctif du sujet de ce certificat dans la zone MQCD accessible par SSLPeerNamePtr et le nom distinctif de l'émetteur dans la zone MQCXP accessible par SSLRemCertIssNamePtr. Les utilisations auxquelles ce nom peut être attribué sont les suivantes:
   - Pour restreindre l'accès via le canal TLS.
   - Pour modifier MQCD.MCAUserIdentifier basé sur le nom.

Concepts associés

Concepts TLS (Transport Layer Security)

Référence associée

Enregistrements d'authentification de canal

Ecriture d'un exit de sécurité

Vous pouvez écrire un exit de sécurité à l'aide du code de squelette d'exit de sécurité.

La Figure 112, à la page 1001 montre comment écrire un exit de sécurité.

```
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 */

}
```

Figure 112. Code squelette de l'exit de sécurité

Le point d'entrée IBM MQ MQStart standard doit exister, mais il n'est pas nécessaire pour exécuter une fonction. Le nom de la fonction (EntryPoint dans cet exemple) peut être modifié, mais la fonction doit être exportée lorsque la bibliothèque est compilée et liée. Comme dans l'exemple précédent, les pointeurs pChannelExitParms doivent être transtypés en PMQCXP et la définition pChanneldoit être transtypée en PMQCD. Pour obtenir des informations générales sur l'appel des exits de canal et l'utilisation des paramètres, voir MQ_CHANNEL_EXIT. Ces paramètres sont utilisés dans un exit de sécurité comme suit:

PMQVOID pChannelExitParms
d'entrée-sortie

Pointeur vers la structure MQCXP-transtypage vers PMQCXP pour accéder aux zones. Cette structure est utilisée pour communiquer entre l'exit et l'agent MCA. Les zones suivantes de MQCXP présentent un intérêt particulier pour les exits de sécurité:

ExitReason
Indique à l'exit de sécurité l'état en cours dans l'échange de sécurité et est utilisé pour décider de l'action à effectuer.

ExitResponse
Réponse à l'agent MCA qui détermine l'étape suivante de l'échange de sécurité.

ExitResponse2
Indicateurs de contrôle supplémentaires permettant de contrôler la façon dont l'agent MCA interprète la réponse de l'exit de sécurité.

Zone ExitUser
16 octets (maximum) de mémoire pouvant être utilisés par l'exit de sécurité pour maintenir l'état entre les appels.

ExitData
Contient les données indiquées dans la zone SCYDATA de la définition de canal (32 octets complétés à droite par des blancs).

Définition pMQVOID pChannel
d'entrée-sortie

Pointeur vers la structure MQCD-transtypage en PMQCD pour accéder aux zones. Ce paramètre contient la définition du canal. Les zones suivantes de MQCD présentent un intérêt particulier pour les exits de sécurité:

ChannelName
Nom du canal (20 octets complétés à droite par des blancs).

ChannelType
Code définissant le type de canal.

ID utilisateur MCA
Ce groupe de trois zones est initialisé à la valeur de la zone MCAUSER indiquée dans la définition de canal. Tout identificateur utilisateur spécifié par l'exit de sécurité dans ces zones est utilisé pour le contrôle d'accès (non applicable aux canaux SDR, SVR, CLNTCONN ou CLUSSDR).

MCAUserIdentifier
Les 12 premiers octets de l'identificateur sont complétés à droite par des blancs.

LongMCAUserIdPtr
Le pointeur vers une mémoire tampon contenant l'identificateur de longueur complète (non garanti comme terminé par une valeur nulle) est prioritaire sur MCAUserIdentifier.

LongMCAUserIdLength
Longueur de la chaîne pointée par LongMCAUserIdPtr -doit être définie si LongMCAUserIdPtr est défini.

Identificateur de l'utilisateur distant
S'applique uniquement aux paires de canaux CLNTCONN/SVRCONN. Si aucun exit de sécurité CLNTCONN n'est défini, ces trois zones sont initialisées par l'agent MCA client, de sorte qu'elles peuvent contenir un identificateur d'utilisateur provenant de l'environnement du client qui peut être utilisé par un exit de sécurité SVRCONN pour l'authentification et lors de la spécification de l'identificateur d'utilisateur MCA. Si un exit de sécurité CLNTCONN est défini, ces zones ne sont pas initialisées et peuvent être définies par l'exit de sécurité CLNTCONN, ou des messages de sécurité peuvent être utilisés pour transmettre un identificateur utilisateur du client au serveur.

Identificateur RemoteUser
Les 12 premiers octets de l'identificateur sont complétés à droite par des blancs.

LongRemoteUserIdPtr
Le pointeur vers une mémoire tampon contenant l'identificateur de longueur complète (non garanti comme terminé par une valeur nulle) est prioritaire sur l'identificateur RemoteUser.

LongRemoteUserIdLongueur
La longueur de la chaîne pointée par LongRemoteUserIdPtr-doit être définie si LongRemoteUserIdPtr est défini.

PMQLONG pDataLongueur
d'entrée-sortie

Pointeur vers MQLONG. Contient la longueur de tout exit de sécurité contenu dans AgentBuffer lors de l'appel de l'exit de sécurité. Doit être défini par un exit de sécurité sur la longueur de tout message envoyé dans AgentBuffer ou ExitBuffer.

PMQLONG pAgentBufferLength
entrée

Pointeur vers MQLONG. Longueur des données contenues dans AgentBuffer lors de l'appel de l'exit de sécurité.

Mémoire tampon pMQVOID pAgent
d'entrée-sortie

Lors de l'appel de l'exit de sécurité, il désigne tout message envoyé par l'exit partenaire. Si l'indicateur MQXR2_USE_AGENT_BUFFER est défini pour ExitResponse2 dans la structure MQCXP (valeur par défaut), un exit de sécurité doit définir ce paramètre pour pointer vers les données de message envoyées.

PMQLONG pExitBufferLength
d'entrée-sortie

Pointeur vers MQLONG. Ce paramètre est initialisé à 0 lors du premier appel d'un exit de sécurité et la valeur renvoyée est conservée entre les appels à l'exit de sécurité lors d'un échange de sécurité.

PMQPTR pExitBufferAddr
d'entrée-sortie

Ce paramètre est initialisé avec un pointeur null lors du premier appel d'un exit de sécurité et la valeur renvoyée est conservée entre les appels à l'exit de sécurité lors d'un échange de sécurité. Si l'indicateur MQXR2_USE_EXIT_BUFFER est défini dans ExitResponse2 de la structure MQCXP, un exit de sécurité doit définir ce paramètre pour pointer vers les données de message envoyées.

Différences de comportement entre les exits de sécurité définis sur les paires de canaux CLNTCONN/SVRCONN et les autres paires de canaux
Les exits de sécurité peuvent être définis sur tous les types de canal. Cependant, le comportement des exits de sécurité définis sur les paires de canaux CLNTCONN/SVRCONN est légèrement différent des exits de sécurité définis sur les autres paires de canaux.

Un exit de sécurité sur un canal CLNTCONN peut définir l'identificateur d'utilisateur distant dans la définition de canal pour le traitement par un exit SVRCONN partenaire, ou pour l'autorisation OAM si aucun exit de sécurité SVRCONN n'est défini et que la zone MCAUSER de SVRCONN n'est pas définie.

Si aucun exit de sécurité CLNTCONN n'est défini, l'ID utilisateur distant dans la définition de canal est défini sur un ID utilisateur provenant de l'environnement client (qui peut être vide) par l'agent MCA client.

Un échange de sécurité entre les exits de sécurité définis sur une paire de canaux CLNTCONN et SVRCONN aboutit lorsque l'exit de sécurité SVRCONN renvoie une réponse ExitResponse de MQXCC_OK. Un échange de sécurité entre d'autres paires de canaux aboutit lorsque l'exit de sécurité qui a lancé l'échange renvoie une ExitResponse de MQXCC_OK.

Toutefois, le code ExitResponse de MQXCC_SEND_AND_REQUEST_SEC_MSG peut être utilisé pour forcer la poursuite de l'échange de sécurité: si une ExitResponse de MQXCC_SEND_AND_REQUEST_SEC_MSG est renvoyée par un exit de sécurité CLNTCONN ou SVRCONN, l'exit partenaire doit répondre en envoyant un message de sécurité (et non MQXCC_OK ou une réponse nulle) ou si le canal s'arrête. Pour les exits de sécurité définis sur d'autres types de canal, une réponse ExitResponse de MQXCC_OK renvoyée en réponse à une réponse MQXCC_SEND_AND_REQUEST_SEC_MSG de l'exit de sécurité partenaire entraîne la poursuite de l'échange de sécurité comme si une réponse nulle était renvoyée et non l'arrêt du canal.

Exit de sécurité SSPI
IBM MQ for Windows fournit un exit de sécurité qui fournit l'authentification pour les canaux IBM MQ à l'aide de l'interface SSPI (Security Services Programming Interface). L'interface SSPI fournit les fonctions de sécurité intégrées de Windows.

Cet exit de sécurité concerne à la fois le client IBM MQ et le serveur IBM MQ .

Les modules de sécurité sont chargés à partir de security.dll ou de secur32.dll. Ces DLL sont fournies avec votre système d'exploitation.

L'authentification unidirectionnelle est fournie sur Windows, à l'aide des services d'authentification NTLM. L'authentification bidirectionnelle est fournie sur Windows 2000, à l'aide des services d'authentification Kerberos .

Le programme d'exit de sécurité est fourni au format source et objet. Vous pouvez utiliser le code objet tel qu'il est ou utiliser le code source comme point de départ pour créer vos propres programmes d'exit utilisateur. Pour plus d'informations sur l'utilisation de l'objet ou du code source de l'exit de sécurité SSPI, voir «Utilisation de l'exit de sécurité SSPI sous Windows», à la page 1168

Programmes d'exit d'émission et de réception de canal
Vous pouvez utiliser les exits d'envoi et de réception pour effectuer des tâches telles que la compression et la décompression des données. Vous pouvez spécifier une liste de programmes d'exit d'envoi et de réception à exécuter successivement.

Les programmes d'exit d'émission et de réception de canal sont appelés aux emplacements suivants du cycle de traitement d'un agent MCA:

- Les programmes d'exit d'émission et de réception sont appelés pour l'initialisation au début de l'agent MCA et pour l'arrêt à la fin de l'agent MCA.
- Le programme d'exit d'émission est appelé à l'une ou l'autre extrémité du canal, en fonction de l'extrémité à laquelle est envoyée une transmission pour un transfert de message, immédiatement avant l'envoi d'une transmission sur la liaison. La remarque 4 explique pourquoi les exits sont disponibles dans les deux directions, même si les canaux de messages envoient des messages dans une seule direction.
- Le programme d'exit de réception est appelé à l'une ou l'autre extrémité du canal, en fonction de la fin de réception d'une transmission pour un transfert de message, immédiatement après qu'une transmission a été effectuée à partir de la liaison. La remarque 4 explique pourquoi les exits sont disponibles dans les deux directions, même si les canaux de messages envoient des messages dans une seule direction.

Il peut y avoir de nombreuses transmissions pour un transfert de message, et il peut y avoir de nombreuses itérations des programmes d'exit d'émission et de réception avant qu'un message n'atteigne l'exit de message à la fin de la réception.

Les programmes d'exit d'émission et de réception de canal reçoivent une mémoire tampon d'agent contenant les données de transmission telles qu'elles sont envoyées ou reçues à partir de la liaison de communication. Pour les programmes d'exit d'émission, les 8 premiers octets de la mémoire tampon sont réservés à l'utilisation par l'agent MCA et ne doivent pas être modifiés. Si le programme renvoie une mémoire tampon différente, ces 8 premiers octets doivent exister dans la nouvelle mémoire tampon. Le format des données présentées aux programmes d'exit n'est pas défini.

Un code de réponse correct doit être renvoyé par les programmes d'exit d'envoi et de réception. Toute autre réponse entraîne une fin anormale de l'agent MCA (fin anormale).

Remarque : N'émettez pas d'appel MQGET, MQPUT ou MQPUT1 dans un point de synchronisation à partir d'un exit d'envoi ou de réception.

Remarque :

1. Les exits d'envoi et de réception fonctionnent généralement par paires. Par exemple, un exit d'émission peut compresser les données et un exit de réception peut les décompresser, ou un exit d'émission peut chiffrer les données et un exit de réception peut les déchiffrer. Lorsque vous

définissez les canaux appropriés, assurez-vous que les programmes d'exit compatibles sont nommés pour les deux extrémités du canal.

2. Si la compression est activée pour le canal, les exits sont transmis aux données compressées.
3. Les exits d'émission et de réception de canal peuvent être appelés pour des segments de message autres que des données d'application, par exemple des messages de statut. Ils ne sont pas appelés lors de la boîte de dialogue de démarrage, ni lors de la phase de vérification de la sécurité.
4. Bien que les canaux de message envoient des messages dans une seule direction, les données de contrôle de canal, telles que les pulsations cardiaques et la fin du traitement par lots, circulent dans les deux directions, et ces exits sont également disponibles dans les deux directions. Toutefois, certains des flux de données de démarrage de canal initiaux sont exemptés de traitement par l'un des exits.
5. Dans certaines circonstances, les exits d'envoi et de réception peuvent être appelés hors séquence ; par exemple, si vous exécutez une série de programmes d'exit ou si vous exécutez également des exits de sécurité. Ensuite, lorsque l'exit de réception est appelé pour la première fois pour traiter des données, il peut recevoir des données qui n'ont pas été transmises via l'exit d'émission correspondant. Si l'exit de réception vient d'effectuer l'opération, par exemple la décompression, sans vérifier au préalable qu'elle est nécessaire, les résultats seront inattendus.

   Vous devez coder vos exits d'émission et de réception de sorte que l'exit de réception puisse vérifier que les données qu'il reçoit ont été traitées par l'exit d'émission correspondant. La méthode recommandée consiste à coder vos programmes d'exit de sorte que:

   - L'exit d'émission met à 0 la valeur du neuvième octet de données et déplace toutes les données sur 1 octet, avant d'effectuer l'opération. (Les 8 premiers octets sont réservés à l'utilisation par l'agent MCA.)
   - Si l'exit de réception reçoit des données ayant un 0 dans l'octet 9, il sait que les données proviennent de l'exit d'émission. Il supprime le 0, effectue l'opération complémentaire et décale les données résultantes de 1 octet.
   - Si l'exit de réception reçoit des données dont la valeur est différente de 0 dans l'octet 9, il suppose que l'exit d'émission ne s'est pas exécuté et renvoie les données à l'appelant telles quelles.

   Lors de l'utilisation d'exits de sécurité, si le canal est arrêté par l'exit de sécurité, il est possible qu'un exit d'émission soit appelé sans l'exit de réception correspondant. Pour éviter ce problème, vous pouvez coder l'exit de sécurité pour définir un indicateur, dans MQCD.SecurityUserData ou MQCD.SendUserData, par exemple, lorsque l'exit décide d'arrêter le canal. Ensuite, l'exit d'émission doit vérifier cette zone et traiter les données uniquement si l'indicateur n'est pas défini. Cette vérification permet d'éviter que l'exit d'émission ne modifie inutilement les données et d'éviter ainsi toute erreur de conversion pouvant se produire si l'exit de sécurité a reçu des données modifiées.

Programmes d'exit d'émission de canal-réservation d'espace
Vous pouvez utiliser des exits d'envoi et de réception pour transformer les données avant la transmission. Les programmes d'exit d'émission de canal peuvent ajouter leurs propres données sur la transformation en réservant de l'espace dans la mémoire tampon de transmission.

Ces données sont traitées par le programme d'exit de réception, puis supprimées de la mémoire tampon. Par exemple, vous pouvez chiffrer les données et ajouter une clé de sécurité pour le déchiffrement.

Comment réserver de l'espace et l'utiliser

Lorsque le programme d'exit d'émission est appelé pour l'initialisation, définissez la zone *ExitSpace* de MQXCP sur le nombre d'octets à réserver. Pour plus d'informations, voir MQCXP . *ExitSpace* ne peut être défini que lors de l'initialisation, c'est-à-dire lorsque *ExitReason* a la valeur MQXR_INIT. Lorsque l'exit d'émission est appelé immédiatement avant la transmission, avec *ExitReason* défini sur MQXR_XMIT, *ExitSpace* octets sont réservés dans la mémoire tampon de transmission. *ExitSpace* n'est pas pris en charge sous z/OS.

L'exit d'émission n'a pas besoin d'utiliser tout l'espace réservé. Elle peut utiliser moins de *ExitSpace* octets ou, si la mémoire tampon de transmission n'est pas saturée, l'exit peut utiliser plus que la quantité

réservée. Lorsque vous définissez la valeur de *ExitSpace*, vous devez laisser au moins 1 ko pour les données de message dans la mémoire tampon de transmission. Les performances des canaux peuvent être affectées si l'espace réservé est utilisé pour de grandes quantités de données.

La mémoire tampon de transmission est normalement longue de 32KB . Toutefois, si le canal utilise TLS, la taille de la mémoire tampon de transmission est réduite à 15 352 octets afin de tenir compte de la longueur d'enregistrement maximale définie par RFC 6101 et de la famille de normes TLS associée. 1024 octets supplémentaires sont réservés pour une utilisation par IBM MQ, de sorte que l'espace maximal de la mémoire tampon de transmission utilisable par les exits d'émission est de 14 328 octets.

Que se passe-t-il à l'extrémité réceptrice du canal?

Les programmes d'exit de réception de canal doivent être configurés pour être compatibles avec les exits d'émission correspondants. Les exits de réception doivent connaître le nombre d'octets dans l'espace réservé et doivent supprimer les données de cet espace.

Exits d'émission multiples

Vous pouvez spécifier une liste de programmes d'exit d'envoi et de réception à exécuter successivement. IBM MQ gère un total pour l'espace réservé par tous les exits d'émission. Cet espace total doit laisser au moins 1 ko pour les données de message dans la mémoire tampon de transmission.

L'exemple suivant montre comment l'espace est alloué à trois exits d'émission, appelés successivement:

1. Lorsqu'il est appelé pour l'initialisation:
   - L'exit d'émission A réserve 1 ko.
   - L'exit d'émission B réserve 2 Ko.
   - L'exit d'émission C réserve 3 Ko.
2. La taille de transmission maximale est de 32 Ko et la longueur des données utilisateur est de 5 Ko.
3. L'exit A est appelé avec 5 Ko de données ; jusqu'à 27 Ko sont disponibles, car 5 Ko sont réservés pour les exits B et C. L'exit A ajoute 1 Ko, le montant qu'il a réservé.
4. L'exit B est appelé avec 6 Ko de données ; jusqu'à 29 Ko sont disponibles, car 3 Ko sont réservés pour l'exit C. L'exit B ajoute 1 ko, soit moins que les 2 ko qu'il a réservés.
5. L'exit C est appelé avec 7 Ko de données ; jusqu'à 32 Ko sont disponibles. L'exit C ajoute 10K, soit plus que les 3 ko qu'il a réservés. Cette quantité est valide, car la quantité totale de données, 17 ko, est inférieure à la quantité maximale de 32 ko.

La taille maximale de la mémoire tampon de transmission pour un canal utilisant TLS est de 15 352 octets, et non 32KB. En effet, les segments de transmission de socket sécurisé sous-jacents sont limités à 16KB et une partie de l'espace est requise pour les surcharges d'enregistrement TLS. 1024 octets supplémentaires sont réservés pour une utilisation par IBM MQ, de sorte que l'espace maximal de la mémoire tampon de transmission utilisable par les exits d'émission est de 14 328 octets.

Programmes d'exit de message de canal
Vous pouvez utiliser l'exit de message de canal pour effectuer des tâches telles que le chiffrement sur le lien, la validation ou la substitution des ID utilisateur entrants, la conversion des données de message, la journalisation et le traitement des messages de référence. Vous pouvez spécifier une liste de programmes d'exit de message à exécuter successivement.

Les programmes d'exit de message de canal sont appelés aux emplacements suivants du cycle de traitement de l'agent MCA:

- Au début et à la fin de l'agent MCA
- Immédiatement après qu'un agent MCA émetteur a émis un appel MQGET
- Avant que l'agent MCA récepteur n'émet un appel MQPUT

L'exit de message est transmis à une mémoire tampon d'agent contenant l'en-tête de file d'attente de transmission MQXQH et le texte du message d'application tel qu'il est extrait de la file d'attente. Le format de MQXQH est indiqué dans MQXQH-en-tête de file d'attente de transmission.

Multi Si vous utilisez des messages de référence (c'est-à-dire des messages qui ne contiennent qu'un en-tête pointant vers un autre objet à envoyer), l'exit de message reconnaît l'en-tête, MQRMH. Il identifie l'objet, l'extrait de la manière appropriée, l'ajoute à l'en-tête, puis le transmet à l'agent MCA pour transmission à l'agent MCA récepteur. Au niveau de l'agent MCA récepteur, un autre exit de message reconnaît que ce message est un message de référence, extrait l'objet et transmet l'en-tête à la file d'attente de destination. Voir «Messages de référence et transferts d'objets LOB», à la page 815 et «Exécution des exemples de message de référence», à la page 1138 pour plus d'informations sur les messages de référence et certains exemples d'exits de message qui les gèrent.

Les exits de message peuvent renvoyer les réponses suivantes:

- Envoyez le message (exit GET). Le message a peut-être été modifié par l'exit. (MQXCC_OK est renvoyé.)
- Placez le message dans la file d'attente (exit PUT). Le message a peut-être été modifié par l'exit. (MQXCC_OK est renvoyé.)
- Ne traitez pas le message. Le message est placé dans la file d'attente de rebut (file d'attente de messages non livrés) par l'agent MCA.
- Fermez le canal.
- Code retour incorrect, qui provoque la fin anormale de l'agent MCA.

Remarque :

1. Les exits de message sont appelés une fois pour chaque message complet transféré, même lorsque le message est divisé en parties.
2. Linux AIX Si vous fournissez un exit de message sous AIX ou Linux, la conversion automatique des ID utilisateur en caractères minuscules (décrite ici) ne fonctionne pas.
3. Un exit s'exécute dans la même unité d'exécution que l'agent MCA lui-même. Il s'exécute également dans la même unité de travail (UOW) que l'agent MCA car il utilise le même descripteur de connexion. Par conséquent, tous les appels effectués sous le point de synchronisation sont validés ou annulés par le canal à la fin du lot. Par exemple, un programme d'exit de message de canal peut envoyer des messages de notification à un autre programme et ces messages ne sont validés dans la file d'attente que lorsque le lot contenant le message d'origine est validé.

   Par conséquent, vous pouvez émettre des appels MQI de point de synchronisation à partir d'un programme d'exit de message de canal.

Conversion de message en dehors de l'exit de message
Avant d'appeler l'exit de message, l'agent MCA récepteur effectue des conversions sur le message. Cette rubrique décrit les algorithmes utilisés pour effectuer les conversions.

Les en-têtes qui sont traités

Une routine de conversion s'exécute dans l'agent MCA du récepteur avant l'appel de l'exit de message. La routine de conversion commence par l'en-tête MQXQH au début du message. La routine de conversion traite ensuite les en-têtes chaînés qui suivent le MQXQH, en effectuant la conversion si nécessaire. Les en-têtes chaînés peuvent s'étendre au-delà du décalage contenu dans le paramètre HeaderLength des données MQCXP transmises à l'exit de message du récepteur. Les en-têtes suivants sont convertis en interne:

- MQXQH (nom de format " `MQXMIT` ")
- MQMD (cet en-tête fait partie de MQXQH et n'a pas de nom de format)
- MQMDE (nom de format " `MQHMDE` ")
- MQDH (nom de format " `MQHDIST` ")
- MQWIH (nom de format " `MQHWIH` ")

Les en-têtes suivants ne sont pas convertis, mais sont remplacés à mesure que l'agent MCA continue de traiter les en-têtes chaînés:

- MQDLH (nom de format " MQDEAD ")
- tous les en-têtes dont les noms de format commencent par les trois caractères'MQH'(par exemple " MQHRF ") qui ne sont pas mentionnés autrement

Mode de traitement des en-têtes

Le paramètre Format de chaque en-tête IBM MQ est lu par l'agent MCA. Le paramètre Format est de 8 octets dans l'en-tête, qui est constitué de 8 caractères mono-octet contenant un nom.

L'agent MCA interprète ensuite les données qui suivent chaque en-tête comme étant du type nommé. Si le format est le nom d'un type d'en-tête éligible pour la conversion de données IBM MQ , il est converti. S'il s'agit d'un autre nom indiquant des données nonMQ (par exemple, MQFMT_NONE ou MQFMT_STRING), l'agent MCA arrête le traitement des en-têtes.

Qu'est-ce que MQCXP HeaderLength?

Le paramètre HeaderLength dans les données MQCXP fournies à un exit de message correspond à la longueur totale des en-têtes MQXQH (qui inclut MQMD), MQMDE et MQDH au début du message. Ces en-têtes sont chaînés à l'aide des noms et des longueurs'Format'.

MQWIH

Les en-têtes chaînés peuvent s'étendre au-delà de HeaderLength dans la zone de données utilisateur. L'en-tête MQWIH, s'il est présent, est l'un des en-têtes qui apparaissent au-delà de HeaderLength.

S'il existe un en-tête MQWIH dans les en-têtes chaînés, il est converti en place avant que l'exit de message du récepteur ne soit appelé.

Programme d'exit de relance de message de canal
L'exit de relance de message de canal est appelé lorsqu'une tentative d'ouverture de la file d'attente cible échoue. Vous pouvez utiliser l'exit pour déterminer dans quelles circonstances effectuer une nouvelle tentative, combien de fois effectuer une nouvelle tentative et à quelle fréquence.

Cet exit est également appelé à l'extrémité réceptrice du canal au démarrage et à l'arrêt de l'agent MCA.

L'exit de relance de message de canal est transmis à une mémoire tampon d'agent contenant l'en-tête de la file d'attente de transmission, MQXQH, et le texte du message d'application tel qu'il est extrait de la file d'attente. Le format de MQXQH est fourni dans Présentation de MQXQH.

L'exit est appelé pour tous les codes raison ; il détermine les codes raison pour lesquels il souhaite que l'agent MCA effectue une nouvelle tentative, le nombre de tentatives et les intervalles. (La valeur du nombre de relances de message définie lors de la définition du canal est transmise à l'exit dans le MQCD, mais l'exit peut ignorer cette valeur.)

La zone MsgRetryCount dans MQCXP est incrémentée par l'agent MCA chaque fois que l'exit est appelé et l'exit renvoie MQXCC_OK avec le temps d'attente contenu dans la zone MsgRetryInterval de MQCXP ou MQXCC_SUPPRESS_FUNCTION. Les nouvelles tentatives se poursuivent indéfiniment jusqu'à ce que l'exit renvoie MQXCC_SUPPRESS_FUNCTION dans la zone ExitResponse de MQCXP. Pour plus d'informations sur l'action effectuée par l'agent MCA pour ces codes achèvement, voir MQCXP .

Si toutes les tentatives échouent, le message est écrit dans la file d'attente des messages non livrés. Si aucune file d'attente de rebut n'est disponible, le canal s'arrête.

Si vous ne définissez pas d'exit de relance de message pour un canal et qu'un échec est susceptible d'être temporaire, par exemple MQRC_Q_FULL, l'agent MCA utilise le nombre de relances de message et les intervalles entre les relances de message définis lors de la définition du canal. Si l'échec est de nature plus permanente et que vous n'avez pas défini de programme d'exit pour le traiter, le message est écrit dans la file d'attente de rebut.

Programme d'exit de définition automatique de canal
L'exit de définition automatique de canal peut être utilisé lorsqu'une demande est reçue pour démarrer un canal récepteur ou de connexion serveur mais qu'aucune définition n'existe pour ce canal (pas pour IBM MQ for z/OS). Il peut également être appelé sur toutes les plateformes pour les canaux émetteur et récepteur de cluster afin d'autoriser la modification de définition pour une instance du canal.

L'exit de définition automatique de canal peut être appelé sur toutes les plateformes à l'exception de z/OS lorsqu'une demande est reçue pour démarrer un canal récepteur ou de connexion serveur, mais qu'il n'existe aucune définition de canal. Vous pouvez l'utiliser pour modifier la définition par défaut fournie pour un canal de réception ou de connexion serveur défini automatiquement, SYSTEM.AUTO.RECEIVERou SYSTEM.AUTO.SVRCON. Voir Préparation des canaux pour une description de la façon dont les définitions de canal peuvent être créées automatiquement.

L'exit de définition automatique de canal peut également être appelé lorsqu'une demande est reçue pour démarrer un canal émetteur de cluster. Il peut être appelé pour les canaux émetteur de cluster et récepteur de cluster afin de permettre la modification de définition pour cette instance du canal. Dans ce cas, l'exit s'applique également à IBM MQ for z/OS. Une utilisation courante de l'exit de définition automatique de canal consiste à modifier les noms des exits de message (MSGEXIT, RCVEXIT, SCYEXIT et SENDEXIT) car les noms d'exit ont des formats différents sur des plateformes différentes. Si aucun exit de définition automatique de canal n'est spécifié, le comportement par défaut sous z/OS consiste à examiner un nom d'exit distribué sous la forme *[path]/libraryname(function)* et à utiliser jusqu'à huit caractères de fonction, s'il existe, ou nom de bibliothèque. Sous z/OS, un programme d'exit de définition automatique de canal doit modifier les zones adressées par MsgExitPtr, MsgUserDataPtr, SendExitPtr, SendUserDataPtr, ReceiveExitPtr et ReceiveUserDataPtr, plutôt que MsgExit, MsgUserData, SendExit, SendUserData, ReceiveExit et les zones de données ReceiveUserelles-mêmes.

Pour plus d'informations, voir Utilisation de canaux définis automatiquement.

Comme pour les autres exits de canal, la liste des paramètres est la suivante:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

Les `ChannelExitParms` sont décrits dans MQCXP. `ChannelDefinition` est décrit dans MQCD.

MQCD contient les valeurs utilisées dans la définition de canal par défaut si elles ne sont pas modifiées par l'exit. L'exit ne peut modifier qu'un sous-ensemble des zones ; voir MQ_CHANNEL_AUTO_DEF_EXIT. Toutefois, la tentative de modification d'autres zones ne génère pas d'erreur.

L'exit de définition automatique de canal renvoie une réponse de MQXCC_OK ou de MQXCC_SUPPRESS_FUNCTION. Si aucune de ces réponses n'est renvoyée, l'agent MCA poursuit le traitement comme si la fonction MQXCC_SUPPRESS_FUNCTION était renvoyée. En d'autres termes, la définition automatique est abandonnée, aucune nouvelle définition de canal n'est créée et le canal ne peut pas démarrer.

ALW *Compilation des programmes d'exit de canal sur les systèmes AIX, Linux, and Windows*

Utilisez les exemples suivants pour vous aider à compiler des programmes d'exit de canal pour les systèmes AIX, Linux, and Windows .

Windows

Windows

La commande du compilateur et de l'éditeur de liens pour les programmes d'exit de canal sous Windows:

```
cl.exe /Ic:\mqm\tools\c\include /nologo /c myexit.c
link.exe /nologo /dll myexit.obj /def:myexit.def /out:myexit.dll
```

Systèmes AIX and Linux

Linux AIX

Dans ces exemples, `exit` est le nom de la bibliothèque et `ChannelExit` est le nom de la fonction. Sous AIX , le fichier d'exportation est appelé `exit.exp`. Ces noms sont utilisés par la définition de canal pour référencer le programme d'exit à l'aide du format décrit dans Définition de canal MQCD. Voir aussi le paramètre MSGEXIT de la commande DEFINE CHANNEL .

AIX Exemples de commandes de compilateur et d'éditeur de liens pour les exits de canal sous AIX:

```
$ xlc_r -q64 -e MQStart -bE:exit.exp -bM:SRE -o /var/mqm/exits64/exit
exit.c -I/usr/mqm/inc
```

Linux Exemples de commandes du compilateur et de l'éditeur de liens pour les exits de canal sous Linux où le gestionnaire de files d'attente est 32 bits:

```
$ gcc -shared -fPIC -o /var/mqm/exits/exit exit.c -I/opt/mqm/inc
```

Linux Exemples de commandes du compilateur et de l'éditeur de liens pour les exits de canal sur Linux où le gestionnaire de files d'attente est 64 bits:

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
```

Sur le client, un exit 32 bits ou 64 bits peut être utilisé. Cet exit doit être lié à mqic_r.

AIX Sous AIX, toutes les fonctions appelées par IBM MQ doivent être exportées. Exemple de fichier d'exportation pour ce fichier make:

```
#
!channelExit
MQStart
```

Configuration des exits de canal

Pour appeler l'exit de canal, vous devez le nommer dans la définition de canal.

Les exits de canal doivent être nommés dans la définition de canal. Vous pouvez utiliser cette appellation lorsque vous définissez les canaux pour la première fois, ou ajouter les informations ultérieurement à l'aide, par exemple, de la commande MQSC ALTER CHANNEL. Vous pouvez également attribuer des noms d'exit de canal dans la structure de données du canal MQCD. Le format du nom d'exit dépend de votre plateforme IBM MQ ; pour plus d'informations, voir MQCD ou commandes MQSC .

Si la définition de canal ne contient pas de nom de programme d'exit utilisateur, l'exit utilisateur n'est pas appelé.

L'exit de définition automatique de canal est la propriété du gestionnaire de files d'attente et non celle du canal individuel. Pour que cet exit puisse être appelé, il doit être nommé dans la définition du gestionnaire de files d'attente. Pour modifier une définition de gestionnaire de files d'attente, utilisez la commande MQSC ALTER QMGR.

Ecriture des exits de conversion de données

Cette collection de rubriques contient des informations sur l'écriture des exits de conversion de données.

Remarque : Non pris en charge dans MQSeries for VSE/ESA.

Lorsque vous effectuez une opération MQPUT, votre application crée le descripteur de message (MQMD) du message. Etant donné que IBM MQ doit pouvoir comprendre le contenu du MQMD, quelle que soit la plateforme sur laquelle il est créé, il est converti automatiquement par le système.

Toutefois, les données d'application ne sont pas converties automatiquement. Si des données de type caractères sont échangées entre des plateformes où les zones `CodedCharSetId` et `Encoding` diffèrent, par exemple entre ASCII et EBCDIC, l'application doit organiser la conversion du message. La conversion des données d'application peut être effectuée par le gestionnaire de files d'attente lui-même ou par un programme d'exit utilisateur, appelé *exit de conversion de données*. Le gestionnaire de files d'attente peut effectuer la conversion de données lui-même, à l'aide de l'une de ses routines de conversion intégrées, si les données d'application se trouvent dans l'un des formats intégrés (tels que MQFMT_STRING). Cette rubrique contient des informations sur la fonction d'exit de conversion de données fournie par IBM MQ lorsque les données d'application ne sont pas dans un format intégré.

Le contrôle peut être transmis à l'exit de conversion de données lors d'un appel MQGET. Cela évite la conversion sur différentes plateformes avant d'atteindre la destination finale. Toutefois, si la destination finale est une plateforme qui ne prend pas en charge la conversion de données sur MQGET, vous devez spécifier CONVERT (YES) sur le canal émetteur qui envoie les données à sa destination finale. Cela garantit que IBM MQ convertit les données lors de la transmission. Dans ce cas, votre exit de conversion de données doit résider sur le système sur lequel le canal émetteur est défini.

L'appel MQGET est émis directement par l'application. Définissez les zones `CodedCharSetId` et `Encoding` dans le MQMD sur le jeu de caractères et le codage requis. Si votre application utilise le même jeu de caractères et le même codage que le gestionnaire de files d'attente, définissez `CodedCharSetId` sur MQCCSI_Q_MGR et `Encoding` sur MQENC_NATIVE. Une fois l'appel MQGET terminé, ces zones ont les valeurs appropriées aux données de message renvoyées. Ces valeurs peuvent différer des valeurs requises si la conversion n'a pas abouti. Votre application doit réinitialiser ces zones avec les valeurs requises avant chaque appel MQGET.

Les conditions requises pour l'appel de l'exit de conversion de données sont définies pour l'appel MQGET dans MQGET.

Pour une description des paramètres transmis à l'exit de conversion de données et des remarques détaillées sur l'utilisation, voir Conversion de données pour l'appel MQ_DATA_CONV_EXIT et la structure MQDXP.

Les programmes qui convertissent les données d'application entre différents codages de machine et CCSID doivent être conformes à l'interface de conversion de données IBM MQ (DCI).

Pour les clients multidiffusion, les exits API et les exits de conversion de données doivent pouvoir s'exécuter côté client car certains messages risquent de ne pas passer par le gestionnaire de files d'attente. Les bibliothèques suivantes font partie des packages client ainsi que des packages serveur:

Tableau 143. Bibliothèques qui se trouvent dans les packages client et serveur

Système d'exploitation	**Bibliothèques**
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bits & 64 bits: libmqm.so & libmqm_r.so
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb

Appel de l'exit de conversion de données

Un exit de conversion de données est un exit écrit par l'utilisateur qui reçoit le contrôle lors du traitement d'un appel MQGET.

L'exit est appelé si les instructions suivantes sont vraies:

- L'option MQGMO_CONVERT est spécifiée dans l'appel MQGET.

- Certaines ou toutes les données de message ne sont pas dans le jeu de caractères ou le codage demandé.
- La zone *Format* de la structure MQMD associée au message n'est pas MQFMT_NONE.
- Le *BufferLength* spécifié dans l'appel MQGET est différent de zéro.
- La longueur des données de message n'est pas égale à zéro.
- Le message contient des données dont le format est défini par l'utilisateur. Le format défini par l'utilisateur peut occuper la totalité du message ou être précédé d'un ou de plusieurs formats intégrés. Par exemple, le format défini par l'utilisateur peut être précédé d'un format MQFMT_DEAD_LETTER_HEADER. L'exit est appelé pour convertir uniquement le format défini par l'utilisateur ; le gestionnaire de files d'attente convertit tous les formats intégrés qui précèdent le format défini par l'utilisateur.

  Un exit écrit par l'utilisateur peut également être appelé pour convertir un format intégré, mais cela se produit uniquement si les routines de conversion intégrées ne peuvent pas convertir le format intégré avec succès.

Il existe d'autres conditions, décrites en détail dans les remarques d'utilisation de l'appel MQ_DATA_CONV_EXIT dans MQ_DATA_CONV_EXIT.

Pour plus d'informations sur l'appel MQGET, voir MQGET . Les exits de conversion de données ne peuvent pas utiliser d'appels MQI autres que MQXCNVC.

Une nouvelle copie de l'exit est chargée lorsqu'une application tente d'extraire le premier message qui utilise cette *Format* depuis que l'application s'est connectée au gestionnaire de files d'attente. Une nouvelle copie peut également être chargée à d'autres moments si le gestionnaire de files d'attente a supprimé une copie précédemment chargée.

L'exit de conversion de données s'exécute dans un environnement tel que celui du programme qui a émis l'appel MQGET. Outre les applications utilisateur, le programme peut être un agent MCA (Message Channel Agent) qui envoie des messages à un gestionnaire de files d'attente de destination qui ne prend pas en charge la conversion de messages. L'environnement inclut l'espace adresse et le profil utilisateur, le cas échéant. L'exit ne peut pas compromettre l'intégrité du gestionnaire de files d'attente car il ne s'exécute pas dans l'environnement du gestionnaire de files d'attente.

Conversion de données sous z/OS

Sous z/OS, tenez compte des points suivants:

- Les programmes d'exit ne peuvent être écrits qu'en langage d'assemblage.
- Les programmes d'exit doivent être réentrants et capables de s'exécuter n'importe où dans l'espace de stockage.
- Les programmes d'exit doivent restaurer l'environnement lors de l'exit à l'entrée et doivent libérer la mémoire obtenue.
- Les programmes d'exit ne doivent pas être WAIT ou émettre des ESTAE ou des SPEI.
- Les programmes d'exit sont généralement appelés comme si z/OS LINK dans:
  - Etat du programme d'incident non autorisé
  - Mode de contrôle de l'espace adresse principal
  - Mode non transmémoire
  - Mode non accès-enregistrement
  - Mode d'adressage 31 bits
  - Mode TCB-PRB
- Lorsqu'il est utilisé par une application CICS , l'exit est appelé par EXEC CICS LINK et doit être conforme aux conventions de programmation CICS . Les paramètres sont transmis par des pointeurs (adresses) dans la zone de communication CICS (COMMAREA).

Bien que cela ne soit pas recommandé, les programmes d'exit utilisateur peuvent également utiliser des appels API CICS , avec les précautions suivantes:

- Ne pas émettre de points de synchronisation, car les résultats pourraient influencer les unités de travail déclarées par l'agent MCA.
- Ne mettez pas à jour les ressources contrôlées par un gestionnaire de ressources autre que IBM MQ for z/OS, y compris celles contrôlées par CICS Transaction Server.

Pour les canaux avec CONVERT = YES, l'exit est chargé à partir du fichier référencé par l'instruction de définition de données CSQXLIB. Les exits fournis par MQCSQCBDCI et CSQCBDCO pour IBM MQ CICS Bridge sont dans SCSQAUTH.

Ecriture d'un programme d'exit de conversion de données pour IBM i

Informations sur les étapes à prendre en compte lors de l'écriture de programmes d'exit de conversion de données MQ pour IBM i.

Procédez comme suit :

1. Nommez votre format de message. Le nom doit tenir dans la zone *Format* du MQMD. Le nom *Format* ne doit pas comporter de blancs imbriqués de début, et les blancs de fin sont ignorés. Le nom de l'objet ne doit pas comporter plus de huit caractères non blancs, car *Format* ne comporte que huit caractères. N'oubliez pas d'utiliser ce nom chaque fois que vous envoyez un message (notre exemple utilise le nom Format).
2. Créez une structure pour représenter votre message. Pour obtenir un exemple, voir Syntaxe valide .
3. Exécutez cette structure via la commande CVTMQMDTA pour créer un fragment de code pour votre exit de conversion de données.

   Les fonctions générées par la commande CVTMQMDTA utilisent des macros fournies dans le fichier QMQM/H (AMQSVMHA). Ces macros sont écrites en supposant que toutes les structures sont condensées ; elles sont modifiées si ce n'est pas le cas.
4. Effectuez une copie du fichier source squelette fourni, QMQMSAMP/QCSRC (AMQSVFC4), puis renommez-le. (Notre exemple utilise le nom EXIT_MOD.)
5. Recherchez les zones de commentaire suivantes dans le fichier source et insérez le code comme indiqué:

   a. Vers la fin du fichier source, une zone de commentaire commence par:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Ici, insérez le fragment de code généré à l'étape «3», à la page 1013.

   b. Vers le milieu du fichier source, une zone de commentaire commence par:

   ```
   /* Insert calls to the code fragments to convert the format's */
   ```

   Elle est suivie d'un appel mis en commentaire à la fonction `ConverttagSTRUCT`.

   Remplacez le nom de la fonction par le nom de la fonction que vous avez ajoutée à l'étape «5.a», à la page 1013. Supprimez les caractères de commentaire pour activer la fonction. S'il existe plusieurs fonctions, créez des appels pour chacune d'elles.

   c. Vers le début du fichier source, une zone de commentaire commence par:

   ```
   /* Insert the function prototypes for the functions produced by */
   ```

   Ici, insérez les instructions de prototype de fonction pour les fonctions ajoutées à l'étape «5.a», à la page 1013.

Si le message contient des données de type caractère, le code généré appelle MQXCNVC ; cette opération peut être résolue en liant le programme de service QMQM/LIBMQM.

6. Compilez le module source, EXIT_MOD, comme suit:

```
CRTCMOD MODULE(library/EXIT_MOD) +
SRCFILE(QCSRC) +
TERASPACE(*YES *TSIFC)
```

7. Créez / liez le programme.

   Pour les applications sans unités d'exécution, utilisez ce qui suit:

```
CRTPGM PGM(library/Format)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

   En plus de la création de l'exit de conversion de données pour l'environnement de base, un autre est requis dans l'environnement à unités d'exécution. Cet objet chargeable doit être suivi de _R. Utilisez la bibliothèque LIBMQM_R pour résoudre les appels à MQXCNVC. Les deux objets chargeables sont requis pour un environnement à unités d'exécution.

```
CRTPGM PGM(library/Format_R)  +
   MODULE(library/EXIT_MOD)  +
   BNDSRVPGM(QMQM/LIBMQM_R)  +
   ACTGRP(QMQM)  +
   USRPRF(*USER)
```

8. Placez la sortie dans la liste des bibliothèques pour le travail IBM MQ . Il est recommandé, pour la production, de stocker les programmes d'exit de conversion de données dans QSYS.

Remarque :

1. Si CVTMQMDTA utilise des structures condensées, toutes les applications IBM MQ doivent utiliser le qualificateur _Packed.
2. Les programmes d'exit de conversion de données doivent être réentrants.
3. MQXCNVC est le seul appel MQI qui peut être émis à partir d'un exit de conversion de données.
4. Compilez le programme d'exit avec l'option de compilation du profil utilisateur définie sur *USER, de sorte que l'exit s'exécute avec les droits de l'utilisateur.
5. L'activation de l'espace mémoire à téraoctets est requise pour tous les exits utilisateur avec IBM MQ for IBM i ; Indiquez TERASPACE (*YES *TSIFC) dans les commandes CRTCMOD et CRTBNDC.

Writing a data-conversion exit program for IBM MQ for z/OS

Information about steps to consider when writing data-conversion exit programs for IBM MQ for z/OS.

Follow these steps:

1. Take the supplied source skeleton CSQ4BAX9 (for non-CICS environments) or CSQ4CAX9 (for CICS) as your starting point.
2. Run the CSQUCVX utility.
3. Follow the instructions in the prolog of CSQ4BAX9 or CSQ4CAX9 to incorporate the routines generated by the CSQUCVX utility, in the order that the structures occur in the message that you want to convert.
4. The utility assumes that the data structures are not packed, that the implied alignment of the data is honored, and that the structures start on a fullword boundary, with bytes being skipped as required (as between ID and VERSION in the example in Valid syntax). If the structures are packed, omit the CMQXCALA macros that are generated. Therefore, consider declaring your structures in such a way that all fields are named and no bytes are skipped; in the example in Valid syntax, add a field "MQBYTE DUMMY;" between ID and VERSION.
5. The supplied exit returns an error if the input buffer is shorter than the message format to be converted. Although the exit converts as many complete fields as possible, the error causes an unconverted message to be returned to the application. If you want to allow short input buffers to

be converted as far as possible, including partial fields, change the TRUNC= value on the CSQXCDFA macro to YES: no error is returned, so the application receives a converted message. The application must handle the truncation.

6. Add any other special processing code that you need.
7. Rename the program to your data format name.
8. Compile and link-edit your program like a batch application program (unless it is for use with CICS applications). The macros in the code generated by the utility are in the library, **thlqual**.SCSQMACS.

   If the message contains character data, the generated code calls MQXCNVC. If your exit uses this call, link-edit it with the exit stub program CSQASTUB. The stub is language-independent and environment-independent. Alternatively, you can load the stub dynamically using the dynamic call name CSQXCNVC. See “Dynamically calling the IBM MQ stub” on page 1057 for more information.

   Place the link-edited module in your application load library, and in a data set that is referenced by the CSQXLIB DD statement of your task procedure started by your channel initiator.
9. If the exit is for use by CICS applications, compile and link-edit it like a CICS application program, including CSQASTUB if required. Place it in your CICS application program library. Define the program to CICS in the typical way, specifying EXECKEY(CICS) in the definition.

Note: Although the LE/370 runtime libraries are needed for running the CSQUCVX utility (see step “2” on page 1014), they are not needed for link-editing or running the data-conversion exit itself (see steps “8” on page 1015 and “9” on page 1015).

See “Writing IMS bridge applications” on page 77 for information about data conversion within the IBM MQ - IMS bridge.

Linux AIX *Ecriture d'un exit de conversion de données pour les systèmes IBM MQ for AIX or Linux*

Informations sur les étapes à prendre en compte lors de l'écriture de programmes d'exit de conversion de données pour les systèmes IBM MQ for AIX or Linux .

Procédez comme suit :

1. Nommez votre format de message. Le nom doit tenir dans la zone *Format* du MQMD et être en majuscules, par exemple MYFORMAT. Le nom *Format* ne doit pas commencer par des blancs. Les blancs de fin sont ignorés. Le nom de l'objet ne doit pas comporter plus de huit caractères non blancs, car *Format* ne comporte que huit caractères. N'oubliez pas d'utiliser ce nom chaque fois que vous envoyez un message.

   Si l'exit de conversion de données est utilisé dans un environnement à unités d'exécution, l'objet chargeable doit être suivi de _r pour indiquer qu'il s'agit d'une version à unités d'exécution.
2. Créez une structure pour représenter votre message. Pour obtenir un exemple, voir Syntaxe valide .
3. Exécutez cette structure via la commande `crtmqcvx` pour créer un fragment de code pour votre exit de conversion de données.

   Les fonctions générées par la commande `crtmqcvx` utilisent des macros qui supposent que toutes les structures sont condensées ; modifiez-les si ce n'est pas le cas.
4. Copiez le fichier source de squelette fourni, en le renommant avec le nom du format de message que vous avez défini à l'étape «1», à la page 1015. Le fichier source squelette et la copie sont en lecture seule.

   Le fichier source squelette est appelé amqsvfc0.c.
5. Sous IBM MQ for AIX, un fichier d'exportation de squelette appelé amqsvfc.exp est également fourni. Copiez ce fichier en le renommant dans MYFORMAT.EXP.
6. Le squelette inclut un exemple de fichier d'en-tête, amqsvmha.h, dans le répertoire *MQ_INSTALLATION_PATH*/inc, où *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé. Assurez-vous que votre chemin d'inclusion pointe vers ce répertoire pour récupérer ce fichier.

Le fichier amqsvmha.h contient des macros qui sont utilisées par le code généré par la commande `crtmqcvx` . Si la structure à convertir contient des données de type caractères, ces macros appellent MQXCNVC.

7. Recherchez les zones de commentaire suivantes dans le fichier source et insérez le code comme indiqué:
   a. Vers la fin du fichier source, une zone de commentaire commence par:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      Ici, insérez le fragment de code généré à l'étape «3», à la page 1015.
   b. Vers le milieu du fichier source, une zone de commentaire commence par:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Elle est suivie d'un appel mis en commentaire à la fonction `ConverttagSTRUCT`.

      Remplacez le nom de la fonction par le nom de la fonction que vous avez ajoutée à l'étape «7.a», à la page 1016. Supprimez les caractères de commentaire pour activer la fonction. S'il existe plusieurs fonctions, créez des appels pour chacune d'elles.
   c. Vers le début du fichier source, une zone de commentaire commence par:

      ```
      /* Insert the function prototypes for the functions produced by */
      ```

      Ici, insérez les instructions de prototype de fonction pour les fonctions ajoutées à l'étape «3», à la page 1015.
8. Compilez votre exit en tant que bibliothèque partagée, en utilisant MQStart comme point d'entrée. Pour ce faire, voir «Compilation des exits de conversion de données sur les systèmes AIX and Linux», à la page 1016.
9. Placez la sortie dans le répertoire d'exit. Le répertoire d'exit par défaut est `/var/mqm/exits` pour les systèmes 32 bits et `/var/mqm/exits64`pour les systèmes 64 bits. Vous pouvez modifier ces répertoires dans le fichier qm.ini ou mqclient.ini . Ce chemin peut être défini pour chaque gestionnaire de files d'attente et l'exit n'est recherché que dans ce ou ces chemins.

Remarque :

1. Si `crtmqcvx` utilise des structures condensées, toutes les applications IBM MQ doivent être compilées de cette manière.
2. Les programmes d'exit de conversion de données doivent être réentrants.
3. MQXCNVC est le seul appel MQI qui peut être émis à partir d'un exit de conversion de données.

Linux AIX *Compilation des exits de conversion de données sur les systèmes AIX and Linux*

Exemples de compilation d'un exit de conversion de données sur les systèmes AIX and Linux .

Sur toutes les plateformes, le point d'entrée du module est MQStart.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

AIX

AIX

Compilez le code source de l'exit à l'aide de l'une des commandes suivantes:

Applications 32 bits

Non unités d'exécution

```
cc -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT \
```

```
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
cc -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT \
     MYFORMAT.c -I MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -I  MQ_INSTALLATION_PATH/inc
```

Linux

Linux

Compilez le code source de l'exit à l'aide de l'une des commandes suivantes:

Applications 31 bits

Non unités d'exécution

```
 gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c \
      -I  MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Applications 32 bits

Non unités d'exécution

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -I  MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
     -I  MQ_INSTALLATION_PATH/inc
```

Windows *Ecriture d'un exit de conversion de données pour IBM MQ for Windows*

Informations sur les étapes à prendre en compte lors de l'écriture de programmes d'exit de conversion de données pour IBM MQ for Windows.

Procédez comme suit :

1. Nommez votre format de message. Le nom doit tenir dans la zone *Format* du MQMD. Le nom *Format* ne doit pas commencer par des blancs. Les blancs de fin sont ignorés. Le nom de l'objet ne doit pas comporter plus de huit caractères non blancs, car *Format* ne comporte que huit caractères.

   Un fichier .DEF appelé amqsvfcn.def est également fourni dans le répertoire des exemples, *MQ_INSTALLATION_PATH*\Tools\C\Samples. *MQ_INSTALLATION_PATH* est le répertoire dans lequel IBM MQ est installé. Effectuez une copie de ce fichier et renommez-le, par exemple, en MYFORMAT.DEF. Vérifiez que le nom de la DLL en cours de création et le nom spécifié dans MYFORMAT.DEF DEF sont les mêmes. Remplacez le nom FORMAT1 dans MYFORMAT.DEF avec le nouveau nom de format.

   N'oubliez pas d'utiliser ce nom chaque fois que vous envoyez un message.
2. Créez une structure pour représenter votre message. Pour obtenir un exemple, voir Syntaxe valide .
3. Exécutez cette structure via la commande `crtmqcvx` pour créer un fragment de code pour votre exit de conversion de données.

   Les fonctions générées par la commande CRTMQCVX utilisent des macros qui sont écrites en supposant que toutes les structures sont condensées ; modifiez-les si ce n'est pas le cas.
4. Copiez le fichier source de squelette fourni, amqsvfc0.c, en le renommant avec le nom du format de message que vous avez défini à l'étape «1», à la page 1018.

   amqsvfc0.c se trouve dans *MQ_INSTALLATION_PATH*\Tools\C\Samples , où *MQ_INSTALLATION_PATH* est le répertoire dans lequel IBM MQ est installé. (Le répertoire d'installation par défaut est `C:\Program Files\IBM\MQ`.)

   Le squelette inclut un exemple de fichier d'en-tête amqsvmha.h dans le répertoire *MQ_INSTALLATION_PATH*\Tools\C\include . Assurez-vous que votre chemin d'inclusion pointe vers ce répertoire pour récupérer ce fichier.

   Le fichier amqsvmha.h contient des macros qui sont utilisées par le code généré par la commande CRTMQCVX. Si la structure à convertir contient des données de type caractères, ces macros appellent MQXCNVC.
5. Recherchez les zones de commentaire suivantes dans le fichier source et insérez le code comme indiqué:

   a. Vers la fin du fichier source, une zone de commentaire commence par:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   Ici, insérez le fragment de code généré à l'étape «3», à la page 1018.

b. Vers le milieu du fichier source, une zone de commentaire commence par:

```
/* Insert calls to the code fragments to convert the format's */
```

Elle est suivie d'un appel mis en commentaire à la fonction ConverttagSTRUCT.

Remplacez le nom de la fonction par le nom de la fonction que vous avez ajoutée à l'étape «5.a», à la page 1018. Supprimez les caractères de commentaire pour activer la fonction. S'il existe plusieurs fonctions, créez des appels pour chacune d'elles.

c. Vers le début du fichier source, une zone de commentaire commence par:

```
/* Insert the function prototypes for the functions produced by */
```

Ici, insérez les instructions de prototype de fonction pour les fonctions ajoutées à l'étape «3», à la page 1018.

6. Créez le fichier de commandes suivant:

```
cl -I  MQ_INSTALLATION_PATH\Tools\C\Include -Tp   \
MYFORMAT.C



MYFORMAT.DEF
```

où *MQ_INSTALLATION_PATH* est le répertoire dans lequel IBM MQ est installé.

7. Exécutez le fichier de commandes pour compiler votre exit en tant que fichier DLL.
8. Placez la sortie dans le sous-répertoire exit sous le répertoire de données IBM MQ . Le répertoire par défaut pour l'installation de vos exits sur les systèmes 32 bits est *MQ_DATA_PATH*\Exits et pour les systèmes 64 bits, *MQ_DATA_PATH*\Exits64

Le chemin utilisé pour rechercher les exits de conversion de données est indiqué dans le registre. Le dossier du registre est:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

et la clé de registre est: ExitsDefaultPath. Ce chemin peut être défini pour chaque gestionnaire de files d'attente et l'exit n'est recherché que dans ce ou ces chemins.

Remarque :

1. Si CRTMQCVX utilise des structures condensées, toutes les applications IBM MQ doivent être compilées de cette manière.
2. Les programmes d'exit de conversion de données doivent être réentrants.
3. MQXCNVC est le seul appel MQI qui peut être émis à partir d'un exit de conversion de données.

Windows *Fichiers de sortie et de commutation de chargement sur les systèmes d'exploitation Windows*

Les processus du gestionnaire de files d'attente IBM WebSphere MQ for Windows 7.5 sont 32 bits. Par conséquent, lors de l'utilisation d'applications 64 bits, certains types de fichiers d'exit et de chargement de commutateur XA doivent également disposer d'une version 32 bits pouvant être utilisée par le gestionnaire de files d'attente. Si la version 32 bits de l'exit ou du fichier de commutation de chargement XA est requise et n'est pas disponible, l'appel d'API ou la commande appropriée échoue.

Deux attributs sont pris en charge dans qm.ini file for *ExitPath*. Il s'agit de ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits et ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé. L'utilisation de ces éléments permet de s'assurer que la bibliothèque appropriée est disponible. Si un

exit est utilisé dans un cluster IBM MQ , cela garantit également que la bibliothèque appropriée sur un système distant peut être trouvée.

Le tableau suivant répertorie les différents types de fichiers de chargement Exit et Switch et indique si des versions 32 bits ou 64 bits, ou les deux, sont requises, selon que des applications 32 bits ou 64 bits sont utilisées:

Types de règle	**Applications 32 bits**	**Applications 64 bits**
exit API	32 bits et 64 bits	64 bits
Exit de conversion de données	32 bits	64 bits
Exits de canal serveur (tous types)	64 bits	64 bits
Exits de canal client (tous types)	32 bits	64 bits
Exit de service installable	64 bits	64 bits
Exit WLM de cluster	64 bits	64 bits
Exit de routage de publication / d'abonnement	64 bits	64 bits
Fichiers de commutation de chargement de base de données	32 bits et 64 bits	64 bits
Bibliothèques AX du gestionnaire de transactions externe	32 bits	64 bits
Exit de préconnexion	32 bits	64 bits

Référencement des définitions de connexion à l'aide d'un exit de préconnexion à partir d'un référentiel

IBM MQ MQI clients peut être configuré pour rechercher un référentiel afin d'obtenir des définitions de connexion à l'aide d'une bibliothèque d'exit de préconnexion.

Introduction

Une application client peut se connecter à un gestionnaire de files d'attente à l'aide des tables de définition de canal du client (CCDT). En règle générale, le fichier CCDT se trouve sur un serveur de fichiers réseau central et les clients y font référence. Etant donné qu'il est difficile de gérer et d'administrer diverses applications client faisant référence au fichier CCDT, une approche flexible consiste à stocker les définitions de client dans un référentiel global tel qu'un annuaire LDAP, un registre WebSphere et un référentiel ou tout autre référentiel. Le stockage des définitions de connexion client dans un référentiel facilite la gestion des définitions de connexion client et les applications peuvent accéder aux définitions de connexion client correctes et les plus récentes.

Lors de l'exécution de l'appel MQCONN/X, IBM MQ MQI client charge une bibliothèque d'exit de préconnexion spécifiée par l'application et appelle une fonction d'exit pour extraire les définitions de connexion. Les définitions de connexion extraites sont ensuite utilisées pour établir une connexion à un gestionnaire de files d'attente. Les détails de la bibliothèque d'exit et de la fonction à appeler sont spécifiés dans le fichier de configuration mqclient.ini .

Syntaxe

`void MQ_PRECONNECT_EXIT` (*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*) ;

Paramètres

Paramètres pExit
Type: entrée / sortie PMQNXP

Structure du paramètre d'exit **PreConnection** .

La structure est allouée et gérée par l'appelant de l'exit.

pQMgrNom
Type: entrée / sortie PMQCHAR

Nom du gestionnaire de files d'attente.

En entrée, ce paramètre est la chaîne de filtrage fournie à l'appel de l'API MQCONN via le paramètre **QMgrName** . Cette zone peut être vide, explicite ou contenir certains caractères génériques. La zone est modifiée par l'exit. Le paramètre est NULL lorsque l'exit est appelé avec MQXR_TERM.

ppConnectOpts
Type: ppConnectOpts en entrée / sortie

Options qui contrôlent l'action de MQCONNX.

Il s'agit d'un pointeur vers une structure d'options de connexion MQCNO qui contrôle l'action de l'appel de l'API MQCONN. Le paramètre est NULL lorsque l'exit est appelé avec MQXR_TERM. Le client MQI fournit toujours une structure MQCNO à l'exit, même s'il n'a pas été fourni à l'origine par l'application. Si une application fournit une structure MQCNO, le client effectue un doublon pour la transmettre à l'exit où elle est modifiée. Le client conserve la propriété du MQCNO.

Un MQCD référencé via le MQCNO est prioritaire sur toute définition de connexion fournie via le tableau. Le client utilise la structure MQCNO pour se connecter au gestionnaire de files d'attente et les autres sont ignorés.

Code pComp
Type: entrée / sortie PMQLONG

Code achèvement.

Pointeur vers un MQLONG qui reçoit le code achèvement des exits. Il doit s'agir de l'une des valeurs suivantes:

- MQCC_OK -L'exécution a abouti
- MQCC_WARNING -Avertissement (achèvement partiel)
- MQCC_FAILED -Echec de l'appel

pReason
Type: entrée / sortie PMQLONG

Motif qualifiant le code pComp.

Pointeur vers un MQLONG qui reçoit le code anomalie de l'exit. Si le code achèvement est MQCC_OK, la seule valeur valide est:

- MQRC_NONE-(0, x'000') Aucun motif à signaler.

Si le code achèvement est MQCC_FAILED ou MQCC_WARNING, la fonction d'exit peut définir la zone de code anomalie sur n'importe quelle valeur MQRC_ * valide.

Appel 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*/
```

Ecriture et compilation des exits de publication

Vous pouvez configurer un exit de publication au niveau du gestionnaire de files d'attente pour modifier le contenu d'un message publié avant qu'il ne soit reçu par les abonnés. Vous pouvez également modifier l'en-tête du message ou ne pas le distribuer à un abonnement.

Remarque : Les exits de publication ne sont pas pris en charge sous z/OS.

Vous pouvez utiliser l'exit de publication pour inspecter et modifier les messages distribués aux abonnés:

- Examiner le contenu d'un message publié pour chaque abonné
- Modifier le contenu d'un message publié pour chaque abonné
- Modifier la file d'attente dans laquelle un message est inséré
- Arrêter la distribution d'un message à un abonné

Ecriture d'un exit de publication

Utilisez les étapes de la rubrique «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962pour vous aider à écrire et à compiler votre exit.

Le fournisseur de l'exit de publication définit les actions de l'exit. Toutefois, l'exit doit être conforme aux règles définies dans MQPSXP.

IBM MQ ne fournit pas d'implémentation du point d'entrée MQ_PUBLISH_EXIT. Il fournit une déclaration typedef de langage C. Utilisez typedef pour déclarer correctement les paramètres à un exit écrit par l'utilisateur. L'exemple suivant montre comment utiliser la déclaration 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 */
}
```

L'exit de publication s'exécute dans le processus du gestionnaire de files d'attente, suite aux opérations suivantes:

- Opération de publication dans laquelle un message est distribué à un ou plusieurs abonnés
- Une opération d'abonnement dans laquelle un ou plusieurs messages conservés sont distribués
- Une opération de demande d'abonnement dans laquelle un ou plusieurs messages conservés sont distribués

Si l'exit de publication est appelé pour une connexion, la première fois qu'il est appelé en tant que code *ExitReason* de MQXR_INIT est définie. Avant que la connexion ne se déconnecte après l'utilisation d'un exit de publication, l'exit est appelé avec le code *ExitReason* MQXR_TERM.

Si l'exit de publication est configuré, mais ne peut pas être chargé lorsque le gestionnaire de files d'attente est démarré, les opérations de message de publication / abonnement sont interdites pour le gestionnaire de files d'attente. Vous devez résoudre le problème ou redémarrer le gestionnaire de files d'attente avant de réactiver la messagerie de publication / abonnement.

Chaque connexion IBM MQ qui requiert l'exit de publication peut échouer à charger ou à initialiser l'exit. Si l'exit ne parvient pas à se charger ou à s'initialiser, les opérations de publication / abonnement qui requièrent l'exit de publication sont désactivées pour cette connexion. Les opérations échouent avec le code anomalie IBM MQ MQRC_PUBLISH_EXIT_ERROR.

Le contexte dans lequel l'exit de publication est appelé est la connexion par une application au gestionnaire de files d'attente. Une zone de données utilisateur est gérée par le gestionnaire de files d'attente pour chaque connexion qui effectue des opérations de publication. L'exit peut conserver des informations dans la zone de données utilisateur pour chaque connexion.

Un exit de publication peut utiliser des appels MQI. Il ne peut utiliser que les appels MQI qui manipulent les propriétés de message. Les appels sont les suivants:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Si l'exit de publication modifie le gestionnaire de files d'attente de destination ou le nom de la file d'attente, aucune nouvelle vérification des droits d'accès n'est effectuée.

Compilation d'un exit de publication

L'exit de publication est une bibliothèque chargée dynamiquement ; il peut être considéré comme un exit de canal. Pour plus d'informations sur la compilation des exits, voir «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962.

Exemple d'exit de publication

L'exemple de programme d'exit est appelé `amqspse0.c`. Il écrit un message différent dans un fichier journal selon que l'exit a été appelé pour des opérations d'initialisation, de publication ou d'arrêt. Il illustre également l'utilisation de la zone de zone utilisateur d'exit pour allouer et libérer de la mémoire de manière appropriée.

Configuration des exits de publication

Vous devez définir certains attributs pour configurer un exit de publication.

Sous Windows et Linux , vous pouvez utiliser l'explorateur IBM MQ pour définir les attributs. Les attributs sont définis dans la page des propriétés du gestionnaire de files d'attente, sous Publication / Abonnement.

Pour configurer l'exit de publication dans le fichier `qm.ini` sur les systèmes AIX and Linux , créez une section appelée `PublishSubscribe`. La section `PublishSubscribe` possède les attributs suivants:

PublishExitPath = [chemin] |*nom_module*
Nom et chemin du module contenant le code d'exit de publication. La longueur maximale de cette zone est `MQ_EXIT_NAME_LENGTH`. La valeur par défaut n'indique aucun exit de publication.

PublishExitFunction = *nom_fonction*
Nom du point d'entrée de fonction dans le module qui contient le code d'exit de publication. La longueur maximale de cette zone est `MQ_EXIT_NAME_LENGTH`.

IBM i Sous IBM i, si un programme est utilisé, omettez `PublishExitFunction`.

PublishExitData = *chaîne*
Si le gestionnaire de files d'attente appelle un exit de publication, il transmet une structure `MQPSXP` en entrée. Les données spécifiées à l'aide de l'attribut **PublishExitData** sont fournies dans la zone *`ExitData`* de la structure. La chaîne peut comporter jusqu'à `MQ_EXIT_DATA_LENGTH` caractères. La valeur par défaut est de 32 caractères blancs.

Ecriture et compilation des exits de charge de travail de cluster

Ecrivez un programme d'exit de charge de travail de cluster pour personnaliser la gestion de la charge de travail des clusters. Vous pouvez prendre en compte le coût d'utilisation d'un canal à différents moments de la journée, ou le contenu des messages, lors du routage des messages. Il s'agit de facteurs qui ne sont pas pris en compte par l'algorithme de gestion de charge de travail standard.

Dans la plupart des cas, l'algorithme de gestion de la charge de travail est suffisant pour vos besoins. Toutefois, pour que vous puissiez fournir votre propre programme d'exit utilisateur afin de personnaliser la gestion de la charge de travail, IBM MQ inclut un exit utilisateur, l'exit de charge de travail du cluster.

Vous pouvez disposer d'informations spécifiques sur votre réseau ou sur les messages que vous pouvez utiliser pour influencer l'équilibrage de la charge de travail. Vous pouvez savoir quels sont les canaux à haute capacité ou les routes de réseau bon marché, ou vous pouvez souhaiter acheminer les messages en fonction de leur contenu. Vous pouvez décider d'écrire un programme d'exit de charge de travail de cluster ou d'utiliser un programme fourni par un tiers.

L'exit de charge de travail de cluster est appelé lors de l'accès à une file d'attente de cluster. Il est appelé par MQOPEN, MQPUT1 et MQPUT.

Le gestionnaire de files d'attente cible sélectionné à l'heure MQOPEN est fixe si MQOO_BIND_ON_OPEN est spécifié. Dans ce cas, l'exit n'est exécuté qu'une seule fois.

Si le gestionnaire de files d'attente cible n'est pas fixe à l'heure MQOPEN , le gestionnaire de files d'attente cible est choisi au moment de l'appel MQPUT . Si le gestionnaire de files d'attente cible n'est pas disponible ou échoue alors que le message se trouve toujours dans la file d'attente de transmission, l'exit est de nouveau appelé. Un nouveau gestionnaire de files d'attente cible est sélectionné. Si le canal de transmission de messages échoue lors du transfert du message et que le message est annulé, un nouveau gestionnaire de files d'attente cible est sélectionné.

Multi Sous Multiplateformes, le gestionnaire de files d'attente charge le nouvel exit de charge de travail de cluster lors du prochain démarrage du gestionnaire de files d'attente.

Si la définition de gestionnaire de files d'attente ne contient pas de nom de programme d'exit de charge de travail de cluster, l'exit de charge de travail de cluster n'est pas appelé.

Diverses données sont transmises à un exit de charge de travail de cluster dans la structure de paramètres d'exit, MQWXP:

- La structure de définition de message, MQMD.
- Paramètre de longueur de message.
- Copie du message ou d'une partie du message.

Sur les plateformes nonz/OS , si vous utilisez CLWLMode=FAST, chaque processus de système d'exploitation charge sa propre copie de l'exit. Des connexions différentes au gestionnaire de files d'attente peuvent entraîner l'appel de différentes copies de l'exit. Si l'exit est exécuté dans le mode sans échec par défaut, CLWLMode=SAFE, une seule copie de l'exit s'exécute dans son propre processus distinct.

Écriture des exits de charge de travail de cluster

z/OS Pour plus d'informations sur l'écriture des exits de charge de travail de cluster pour z/OS, voir «Cluster workload exit programming for IBM MQ for z/OS», à la page 1026.

Depuis la IBM MQ 9.1.0, des exits de pondération de charge du cluster s'exécutent dans l'espace adresse d'initiateur de canal au lieu de l'espace adresse de gestionnaire de files d'attente. Si vous disposez d'un exit de pondération de charge du cluster, vous devez retirer l'instruction CSQXLIB DD de la procédure de tâche démarrée de votre gestionnaire de files d'attente et ajouter l'ensemble de données contenant l'exit de pondération de charge du cluster à la concaténation dans la procédure de tâche démarrée de votre initiateur de canal.

Multi Pour Multiplatforms, les exits de charge de travail de cluster ne doivent pas utiliser d'appels MQI. A d'autres égards, les règles d'écriture et de compilation des programmes d'exit de charge de travail de cluster sont similaires aux règles qui s'appliquent aux programmes d'exit de canal. Suivez les étapes de la rubrique «Ecriture d'exits et de services installables sous AIX, Linux, and Windows», à la page 962et utilisez l'exemple de programme «Exemple d'exit de charge de travail de cluster», à la page 1025 pour vous aider à écrire et à compiler votre exit.

Pour plus d'informations sur les exits de canal, voir «Ecriture de programmes d'exit de canal», à la page 991.

Configuration des exits de charge de travail de cluster

Vous pouvez nommer les exits de charge de travail de cluster dans la définition de gestionnaire de files d'attente en spécifiant l'attribut d'exit de charge de travail de cluster dans la commande `ALTER QMGR`. Exemple :

```
ALTER QMGR CLWLEXIT(myexit)
```

Référence associée

Appel d'exit de charge de travail de cluster et structures de données

Exemple d'exit de charge de travail de cluster

IBM MQ inclut un exemple de programme d'exit de charge de travail de cluster. Vous pouvez copier l'exemple et l'utiliser comme base pour vos propres programmes.

z/OS **IBM MQ for z/OS**

L'exemple de programme d'exit de charge de travail de cluster est fourni en assembleur et dans C. La version assembleur est appelée CSQ4BAF1 et se trouve dans la bibliothèque `thlqual.SCSQASMS`. La version C est appelée CSQ4BCF1 et se trouve dans la bibliothèque `thlqual.SCSQC37S`. `thlqual` est le qualificatif de haut niveau de la bibliothèque cible pour les fichiers IBM MQ de votre installation.

Multi **IBM MQ for Multiplatforms**

L'exemple de programme d'exit de charge de travail de cluster est fourni en C et est appelé `amqswlm0.c`. Il se trouve à l'emplacement suivant :

Tableau 144. Exemple d'emplacement de programme d'exit de charge de travail de cluster pour Multiplatforms

Plateforme	Chemin d'accès au fichier
AIX AIX	*MQ_INSTALLATION_PATH*`/samp`
Windows Windows	*MQ_INSTALLATION_PATH* `\Tools\c\Samples`
IBM i IBM i	La bibliothèque `qmqm`

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Cet exemple d'exit achemine tous les messages vers un gestionnaire de files d'attente particulier, sauf si ce dernier devient indisponible. Il réagit à l'échec du gestionnaire de files d'attente en acheminant les messages vers un autre gestionnaire de files d'attente.

Indiquez à quel gestionnaire de files d'attente vous souhaitez que les messages soient envoyés. Indiquez le nom du canal récepteur de cluster dans l'attribut `CLWLDATA` de la définition de gestionnaire de files d'attente. Exemple :

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Pour activer l'exit, indiquez son chemin d'accès complet et son nom dans l'attribut CLWLEXIT :

Linux AIX Sous AIX and Linux :

```
ALTER QMGR CLWLEXIT(' path /amqswlm(clwlFunction)')
```

Windows Sous Windows :

```
ALTER QMGR CLWLEXIT(' path \amqswlm(clwlFunction)')
```

z/OS Sous z/OS :

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

où x est 'A' ou 'C', selon le langage de programmation de la version que vous utilisez.

IBM i Sous IBM i, utilisez l'une des commandes suivantes:

- Utilisez la commande MQSC:

  ```
  ALTER QMGR CLWLEXIT('AMQSWLM       library       ')
  ```

  Le nom de programme et le nom de bibliothèque occupent tous deux 10 caractères et sont complétés par des blancs à droite si nécessaire.
- Utilisez la commande CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

A présent, au lieu d'utiliser l'algorithme de gestion de charge de travail fourni, IBM MQ appelle cet exit pour acheminer tous les messages vers le gestionnaire de files d'attente de votre choix.

z/OS *Cluster workload exit programming for IBM MQ for z/OS*

Cluster workload exits are invoked as if by a z/OS **LINK** command. Exits are subject to a number of stringent programming rules. Avoid using most SVC commands that involve waits, or using a STAE or ESTAE in a workload exit.

Cluster workload exits are invoked as if by a z/OS **LINK** in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31 bit addressing mode
- Storage key 8
- Program Key Mask 8
- TCB key 8

Put the link-edited modules in the data set specified by the CSQXLIB DD statement of the started task procedure of the channel initiator. The names of the load modules are specified as the workload exit names in the queue manager definition.

When writing workload exits for IBM MQ for z/OS, the following rules apply:

- You must write exits in assembler or C. If you use C, it must conform to the C systems programming environment for system exits, described in the *z/OS C/C++ Programming Guide*, SC09-4765.
- If using the MQXCLWLN call, link edit with CSQMFCLW, supplied in *thlqual*.SCSQLOAD.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the queue manager is running, with the new version used in the next MQCONN thread the queue manager starts.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment on return to that at entry.
- Exits must free any storage obtained, or ensure that storage is freed by a subsequent exit invocation.
- No MQI calls are allowed.
- Exits must not use any system services that could cause a wait, because a wait severely degrades the performance of the queue manager. In general, therefore, avoid an SVC, PC, or I/O.
- Exits must not issue an ESTAE or SPIE, apart from within any subtasks they attach.

Note: There are no absolute restrictions on what you can do in an exit. However, most SVCs involve waits, so avoid them, except for the following commands:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Do not use ESTAEs and ESPIEs because their error handling might interfere with the error handling performed by IBM MQ. IBM MQ might not be able to recover from an error, or your exit program might not receive all the error information.

The system parameter EXITLIM limits the amount of time an exit might run for. The default value for EXITLIM is 30 seconds. If you see the return code MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' your exit might be looping. If you think the exit needs more than 30 seconds to complete, increase the value of EXITLIM.

Création d'une application procédurale

Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

AIX Génération de votre application procédurale sur AIX

Les publications AIX expliquent comment générer des applications exécutables à partir des programmes que vous écrivez.

Cette rubrique décrit les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications IBM MQ for AIX à exécuter sous AIX. C, C + + et COBOL sont pris en charge. Pour plus d'informations sur la préparation de vos programmes C + +, voir Utilisation de C++.

Les tâches que vous devez effectuer pour créer une application exécutable à l'aide de IBM MQ for AIX varient en fonction du langage de programmation dans lequel votre code source est écrit. En plus de coder les appels MQI dans votre code source, vous devez ajouter les instructions de langage appropriées pour inclure les fichiers d'inclusion IBM MQ for AIX pour la langue que vous utilisez. Familiarisez-vous avec le contenu de ces fichiers. Pour une description complète, voir «Fichiers de définition de données IBM MQ», à la page 735 .

Lorsque vous exécutez des applications serveur à unités d'exécution ou client à unités d'exécution, définissez la variable d'environnement AIXTHREAD_SCOPE = S.

AIX *Préparation des programmes C dans AIX*

Cette rubrique contient des informations sur la liaison des bibliothèques nécessaires à la préparation des programmes C sous AIX.

Les programmes C précompilés sont fournis dans le répertoire *MQ_INSTALLATION_PATH*/samp/bin . Utilisez le compilateur ANSI et exécutez les commandes suivantes. Pour plus d'informations sur la programmation d'applications 64 bits, voir Codage des normes sur les plateformes 64 bits.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour les applications 32 bits:

```
$ xlc_r -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm
```

où amqsput0 est un exemple de programme.

Pour les applications 64 bits :

```
$ xlc_r -q64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm
```

où amqsput0 est un exemple de programme.

V 9.4.0 Pour les applications 32 bits utilisant le compilateur XLC 17:

```
$ ibm-clang -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-lmqm
```

où amqsput0 est un exemple de programme.

V 9.4.0 Pour les applications 64 bits utilisant le compilateur XLC 17:

```
$ ibm-clang -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/
lib64 -lmqm
```

où amqsput0 est un exemple de programme.

Si vous utilisez le compilateur VisualAge C/C++ pour les programmes C + +, vous devez inclure l'option -q namemangling=v5 pour que tous les symboles IBM MQ soient résolus lors de la liaison des bibliothèques.

Si vous souhaitez utiliser les programmes sur une machine sur laquelle seul IBM MQ MQI client for AIX est installé, recompilez les programmes pour les lier à la bibliothèque client (-lmqic) à la place.

Liaison de bibliothèques

Vous avez besoin des bibliothèques suivantes:

- Liez vos programmes à la bibliothèque appropriée fournie par IBM MQ.

  Dans un environnement sans unités d'exécution, établissez un lien vers l'une des bibliothèques suivantes:

Fichier de bibliothèque	Type de programme / d'exit
libmqm.a	Serveur pour C
libmqic.a & libmqm.a	Client pour C

  Dans un environnement à unités d'exécution, établissez un lien vers l'une des bibliothèques suivantes:

Fichier de bibliothèque	Type de programme / d'exit
libmqm_r.a	Serveur pour C
libmqic_r.a & libmqm_r.a	Client pour C

Par exemple, pour générer une application IBM MQ à unités d'exécution simples à partir d'une seule unité de compilation, exécutez les commandes suivantes.

Pour les applications 32 bits:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -lmqm_r
```

où `amqsput0` est un exemple de programme.

Pour les applications 64 bits :

```
$ xlc_r -q64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r
```

où `amqsput0` est un exemple de programme.

V 9.4.0 Pour les applications 32 bits utilisant le compilateur XLC 17:

```
$ ibm-clang_r -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib -lmqm_r
```

où `amqsput0` est un exemple de programme.

V 9.4.0 Pour les applications 64 bits utilisant le compilateur XLC 17:

```
$ ibm-clang_r -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib64 -lmqm_r
```

où `amqsput0` est un exemple de programme.

Si vous souhaitez utiliser les programmes sur une machine sur laquelle seul IBM MQ MQI client for AIX est installé, recompilez les programmes pour les lier à la bibliothèque client (`-lmqic`) à la place.

Remarque :

1. Vous ne pouvez pas créer de lien vers plusieurs bibliothèques. En d'autres termes, vous ne pouvez pas lier à la fois une bibliothèque à unités d'exécution et une bibliothèque non à unités d'exécution.
2. Si vous écrivez un service installable (voir Administration de IBM MQ pour plus d'informations), vous devez établir un lien vers la bibliothèque `libmqmzf.a` dans une application non à unités d'exécution et vers la bibliothèque `libmqmzf_r.a` dans une application à unités d'exécution.
3. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries, Encina ou BEA Tuxedo, vous devez établir une liaison avec `libmqmxa.a` (ou `libmqmxa64.a` si votre gestionnaire de transactions traite le type'long'comme 64 bits) et les bibliothèques `libmqz.a` dans une application non à unités d'exécution et avec `libmqmxa_r.a` (ou `libmqmxa64_r.a`) et des bibliothèques `libmqz_r.a` dans une application à unités d'exécution.
4. Vous devez lier des applications sécurisées aux bibliothèques IBM MQ à unités d'exécution. Toutefois, une seule unité d'exécution d'une application sécurisée sur les systèmes IBM MQ for AIX or Linux peut être connectée à la fois.
5. Vous devez lier les bibliothèques IBM MQ avant toute autre bibliothèque de produit.

AIX *Préparation de programmes COBOL dans AIX*

Utilisez ces informations lors de la préparation de programmes COBOL dans AIX à l'aide de IBM COBOL Set et Micro Focus COBOL.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

- Les fichiers de stockage COBOL 32 bits sont installés dans le répertoire suivant:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

et des liens symboliques sont créés dans:

```
MQ_INSTALLATION_PATH/inc
```

- Les fichiers de stockage COBOL 64 bits sont installés dans le répertoire suivant:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

Dans les exemples suivants, définissez la variable d'environnement **COBCPY** sur:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

pour les applications 32 bits, et:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

pour les applications 64 bits.

Vous devez lier votre programme à l'un des fichiers de bibliothèque suivants:

Fichier de bibliothèque	Type de programme / d'exit
libmqmcb.a	Serveur pour COBOL (application non à unités d'exécution)
libmqmcb_r.a	Server for COBOL (application à unités d'exécution)
libmqicb.a	Client for COBOL (application non à unités d'exécution
libmqicb_r.a	Client for COBOL (application à unités d'exécution)

Vous pouvez utiliser le compilateur IBM COBOL Set ou le compilateur Micro Focus COBOL en fonction du programme:

- Les programmes démarrant `amqm` sont adaptés au compilateur Micro Focus COBOL, et
- Les programmes démarrant `amq0` conviennent à l'un ou l'autre compilateur.

Préparation de programmes COBOL à l'aide de IBM COBOL Set for AIX

Des exemples de programmes COBOL sont fournis avec IBM MQ. Pour compiler un tel programme, entrez la commande appropriée dans la liste suivante:

Application de serveur non à unités d'exécution 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

Application client non à unités d'exécution 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

Application de serveur à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Application client à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Application serveur 64 bits sans unités d'exécution

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

Application client 64 bits sans unités d'exécution

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

Application serveur à unités d'exécution 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Application client à unités d'exécution 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

Préparation de programmes COBOL à l'aide de Micro Focus COBOL

Définissez les variables d'environnement avant de compiler votre programme comme suit:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Pour compiler un programme COBOL 32 bits à l'aide de Micro Focus COBOL, entrez:

- Serveur pour COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb
```

- Client pour COBOL

```
$ cob32 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb
```

- Serveur à unités d'exécution pour COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Client à unités d'exécution pour COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r
```

Pour compiler un programme COBOL 64 bits à l'aide de Micro Focus COBOL, entrez:

- Serveur pour COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb
```

- Client pour COBOL

```
$ cob64 -xvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb
```

- Serveur à unités d'exécution pour COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Client à unités d'exécution pour COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r
```

où `amqminqx` est un exemple de programme

Consultez la documentation Micro Focus COBOL pour obtenir une description des variables d'environnement que vous devez configurer.

AIX *Préparation des programmes d'application CICS dans AIX*

Utilisez ces informations lors de la préparation des programmes CICS dans AIX.

Utilisez les modules du *commutateur XA* pour lier CICS à IBM MQ. Pour plus d'informations sur la structure de commutateur XA, voir Structures de commutateur XA.

L'exemple de fichier de code source est fourni pour vous permettre de développer les commutateurs XA pour d'autres messages de transaction. Le nom du module de commutation de chargement fourni est indiqué dansTableau 145, à la page 1032.

Tableau 145. Code essentiel pour les programmes d'application CICS sous AIX: routine d'initialisation XA

Description	**C (source)**	**C (exec)-ajouter à votre XAD.Stanza**
Routine d'initialisation XA	amqzscix.c	*amqzsc* - CICS pour AIX

Utilisez la version prégénérée du fichier de commutation de chargement IBM MQ *amqzsc*, qui est fourni avec le produit.

Liez toujours vos transactions C à la bibliothèque IBM MQ autorisant les unités d'exécution multiples *libmqm_r.a.*, et vos transactions COBOL avec la bibliothèque COBOL *libmqmcb_r.a.*.

Vous trouverez plus d'informations sur la prise en charge des transactions CICS dans le guide d'administration du système Administration de IBM MQ IBM MQ .

AIX *prise en charge de TXSeriesCICS*

IBM MQ sur AIX prend en charge TXSeries CICS à l'aide de l'interface XA. Vérifiez que les applications CICS sont liées à la version à unités d'exécution des bibliothèques IBM MQ .

Vous pouvez exécuter des programmes CICS à l'aide de IBM COBOL Set for AIX ou Micro Focus COBOL. Les sections suivantes décrivent la différence entre l'exécution de programmes CICS sur IBM COBOL Set for AIX et Micro Focus COBOL.

Ecrivez les programmes IBM MQ qui sont chargés dans la même région CICS en C ou COBOL. Vous ne pouvez pas combiner des appels MQI C et COBOL dans la même région CICS . La plupart des appels MQI dans la deuxième langue utilisée échouent avec le code anomalie `MQRC_HOBJ_ERROR`.

Préparation de programmes COBOL CICS à l'aide de IBM COBOL Set for AIX

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour utiliser IBM COBOL, procédez comme suit:

1. Exportez la variable d'environnement suivante :

```
export LDFLAGS="-qLIB -bI:/usr/lpp/cics/lib/cicsprIBMCOB.exp \
                -I MQ_INSTALLATION_PATH/inc -I/usr/lpp/cics/include \
                -e _iwz_cobol_main  \
```

où LIB est une directive de compilation.

2. Traduisez, compilez et liez le programme en tapant:

```
cicstcl -l IBMCOB yourprog.ccp
```

Préparation de programmes COBOL CICS à l'aide de Micro Focus COBOL

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour utiliser Micro Focus COBOL, procédez comme suit:

1. Ajoutez le module de bibliothèque d'exécution COBOL IBM MQ à la bibliothèque d'exécution à l'aide de la commande suivante:

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

Remarque : Avec `cicsmkcobol`, IBM MQ ne vous permet pas d'effectuer des appels MQI dans le langage de programmation C à partir de votre application COBOL.

Si vos applications existantes ont de tels appels, il est recommandé de déplacer ces fonctions des applications COBOL vers votre propre bibliothèque, par exemple, `myMQ.so`. Après avoir déplacé les fonctions, n'incluez pas la IBM MQ bibliothèque `libmqmcbrt.o` lors de la génération de l'application COBOL pour CICS.

De plus, si votre application COBOL n'effectue aucun appel COBOL MQI, ne liez pas `libmqmz_r` à `cicsmkcobol`.

Cela crée le fichier de méthode de langage Micro Focus COBOL et permet à la bibliothèque COBOL d'exécution CICS d'appeler des systèmes IBM MQ for AIX or Linux .

Remarque : Exécutez `cicsmkcobol` uniquement lorsque vous installez l'un des produits suivants:

- Nouvelle version ou édition de Micro Focus COBOL
- Nouvelle version ou édition d' CICS for AIX
- Nouvelle version ou édition de tout produit de base de données pris en charge (pour les transactions COBOL uniquement)
- Nouvelle version ou édition de IBM MQ

2. Exportez la variable d'environnement suivante :

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. Traduisez, compilez et liez le programme en tapant:

```
cicstcl -l COBOL -e yourprog.ccp
```

Préparation des programmes CICS C

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Générez des programmes CICS C à l'aide des fonctions CICS standard:

1. Exportez ***un*** des variables d'environnement suivantes:
   - LDFLAGS = "-L/ *MQ_INSTALLATION_PATH* lib -lmqm_r" export LDFLAGS

- USERLIB = "-L *MQ_INSTALLATION_PATH* lib -lmqm_r" export USERLIB

2. Traduisez, compilez et liez le programme en tapant:

```
cicstcl -l C amqscic0.ccs
```

Exemple de transaction CICS C

L'exemple de source C pour une transaction AIX IBM MQ est fourni par AMQSCIC0.CCS. La transaction lit les messages de la file d'attente de transmission SYSTEM.SAMPLE.CICS.WORKQUEUE sur le gestionnaire de files d'attente par défaut et les place dans la file d'attente locale avec un nom de file d'attente contenu dans l'en-tête de transmission du message. Tous les échecs sont envoyés à la file d'attente SYSTEM.SAMPLE.CICS.File d'attente des messages non livrés Utilisez l'exemple de script MQSC AMQSCIC0.TST pour créer ces files d'attente et des exemples de files d'attente d'entrée.

IBM i Génération de votre application procédurale sur IBM i

Les publications IBM i expliquent comment générer des applications exécutables à partir des programmes que vous écrivez pour les exécuter avec IBM i sur les systèmes iSeries ou System i .

Cette rubrique décrit les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications de procédure IBM MQ for IBM i à exécuter sur les systèmes IBM i . Les langages de programmation COBOL, C, C + +, Java et RPG sont pris en charge. Pour plus d'informations sur la préparation de vos programmes C + +, voir Utilisation de C++. Pour plus d'informations sur la préparation de vos programmes Java , voir Utilisation de IBM MQ classes for Java.

Les tâches que vous devez effectuer pour créer une application IBM MQ for IBM i exécutable dépendent du langage de programmation dans lequel le code source est écrit. En plus de coder les appels MQI dans votre code source, vous devez ajouter les instructions de langage appropriées pour inclure les fichiers de définition de données IBM MQ for IBM i pour le langage que vous utilisez. Familiarisez-vous avec le contenu de ces fichiers. Pour une description complète, voir «Fichiers de définition de données IBM MQ», à la page 735 .

IBM i *Préparation des programmes C dans IBM i*

IBM MQ for IBM i prend en charge les messages d'une taille maximale de 100 Mo. Les programmes d'application écrits en ILE C, prenant en charge les messages IBM MQ de plus de 16 Mo, doivent utiliser l'option de compilation `Teraspace` pour allouer suffisamment de mémoire à ces messages.

Pour plus d'informations sur les options de compilation C, voir le manuel *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Pour compiler un module C, vous pouvez utiliser la commande IBM i **CRTCMOD**. Assurez-vous que la bibliothèque contenant les fichiers d'inclusion (QMQM) figure dans la liste des bibliothèques lors de la compilation.

Vous devez ensuite lier la sortie du compilateur au programme de service à l'aide de la commande **CRTPGM** .

Tableau 146. Exemple de CRTPGM dans des environnements sans unités d'exécution et avec unités d'exécution

Type d'environnement	Commande	Type de programme / d'exit
Environnement sans unités d'exécution	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM)`	Serveur ou client pour C
Environnement à unités d'exécution	`CRTPGM PGM( pgmname ) MODULE( pgmname ) BNDSRVPGM(QMQM/LIBMQM_R)`	Serveur ou client pour C

où *pgmname* est le nom de votre programme.

Le Tableau 147, à la page 1035 répertorie les bibliothèques requises lors de la préparation de programmes C sous IBM i dans un environnement sans unités d'exécution et un environnement à unités d'exécution.

Tableau 147. Bibliothèques requises pour les environnements non à unités d'exécution et à unités d'exécution

Type d'environnement	Fichier de bibliothèque	Type de programme / d'exit
Environnement sans unités d'exécution	libmqm	Serveur pour C
	LIBMQIC & LIBMQM	Client pour C
Environnement à unités d'exécution	libmqm_r	Serveur pour C
	LIBMQIC_R & LIBMQM_R	Client pour C

IBM i *Préparation de programmes COBOL dans IBM i*

Découvrez la préparation de programmes COBOL dans IBM i et la méthode d'accès à l'interface MQI à partir du programme COBOL.

Pourquoi et quand exécuter cette tâche

Pour accéder à l'interface MQI à partir de programmes COBOL, IBM MQ for IBM i fournit une interface d'appel de procédure liée fournie par des programmes de service. Cela permet d'accéder à toutes les fonctions MQI dans IBM MQ for IBM i et de prendre en charge les applications à unités d'exécution. Cette interface ne peut être utilisée qu'avec le compilateur ILE COBOL.

La syntaxe COBOL CALL standard est utilisée pour accéder aux fonctions MQI.

Les fichiers de copie COBOL contenant les constantes nommées et les définitions de structure à utiliser avec l'interface MQI sont contenus dans le fichier physique source QMQM/QCBLLESRC.

Les fichiers de copie COBOL utilisent le guillemet simple (') comme délimiteur de chaîne. Les compilateurs COBOL IBM i supposent que le délimiteur est le guillemet ("). Pour empêcher les compilateurs de générer des messages d'avertissement, spécifiez `OPTION (*APOST)` dans les commandes **CRTCBLPGM**, **CRTBNDCBL**ou **CRTCBLMOD**.

Pour que le compilateur accepte le guillemet simple (') comme délimiteur de chaîne dans les fichiers de copie COBOL, utilisez l'option de compilation \APOST.

Remarque : L'interface d'appel dynamique n'est pas fournie dans IBM MQ 9.0 ou version ultérieure.

Pour utiliser l'interface d'appel de procédure liée, procédez comme suit.

Procédure

1. Créez un module à l'aide du compilateur **CRTCBLMOD** en spécifiant le paramètre:

   ```
   LINKLIT(*PRC)
   ```

2. Utilisez la commande **CRTPGM** pour créer l'objet programme en spécifiant le paramètre approprié:

   Pour les applications sans unités d'exécution:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   Pour les applications à unités d'exécution:

```
BNDSRVPGM(QMQM/AMQ0STUB_R)        Server for COBOL for threaded applications
BNDSRVPGM(QMQM/AMQCSTUB_R)        Client for COBOL for threaded applications
```

Remarque : A l'exception des programmes créés à l'aide du compilateur V4R4 ILE COBOL et contenant l'option THREAD (SERIALIZE) dans l'instruction PROCESS, les programmes COBOL ne doivent pas utiliser les bibliothèques IBM MQ à unités d'exécution. Même si un programme COBOL a été rendu compatible avec les unités d'exécution de cette manière, soyez prudent lorsque vous concevez l'application, car THREAD (SERIALIZE) force la sérialisation des procédures COBOL au niveau du module et peut affecter les performances globales.

Pour plus d'informations, voir *WebSphere Development Studio: ILE COBOL Programmer's Guide* et *WebSphere Development Studio: ILE COBOL Reference* .

Pour plus d'informations sur la compilation d'une application CICS , voir le manuel *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *Préparation des programmes CICS dans IBM i*

Découvrez les étapes requises lors de la préparation des programmes CICS dans IBM i.

Pour créer un programme qui inclut des instructions EXEC CICS et des appels MQI, procédez comme suit:

1. Si nécessaire, préparez les mappes à l'aide de la commande CRTCICSMAP.
2. Convertissez les commandes EXEC CICS en instructions en langage natif. Utilisez la commande CRTCICSC pour un programme C. Utilisez la commande CRTCICSCBL pour un programme COBOL.

   Incluez `CICSOPT(*NOGEN)` dans la commande CRTCICSC ou CRTCICSCBL. Le traitement s'arrête pour vous permettre d'inclure les programmes de service CICS et IBM MQ appropriés. Cette commande place le code, par défaut, dans QTEMP/QACYCICS.
3. Compilez le code source à l'aide de la commande CRTCMOD (pour un programme C) ou de la commande CRTCBLMOD (pour un programme COBOL).
4. Utilisez CRTPGM pour lier le code compilé aux programmes de service CICS et IBM MQ appropriés. Cette opération crée le programme exécutable.

Voici un exemple de ce code (il compile l'exemple de programme CICS fourni):

```
CRTCICSC  OBJ(QTEMP/AMQSCIC0) SRCFILE(/MQSAMP/QCSRC) +
     SRCMBR(AMQSCIC0) OUTPUT(*PRINT) +
     CICSOPT(*SOURCE *NOGEN)
CRTCMOD   MODULE(MQTEST/AMQSCIC0) +
     SRCFILE(QTEMP/QACYCICS) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/AMQSCIC0) MODULE(MQTEST/AMQSCIC0) +
     BNDSRVPGM(QMQM/LIBMQIC QCICS/AEGEIPGM)
```

IBM i *Préparation de programmes RPG dans IBM i*

Si vous utilisez IBM MQ for IBM i, vous pouvez écrire vos applications dans RPG.

Pour plus d'informations, voir «Codage des programmes IBM MQ en RPG (IBM i uniquement)», à la page 1085et consultez le manuel IBM i Application Programming Reference (ILE/RPG).

IBM i *Remarques sur la programmation SQL pour IBM i*

Découvrez les étapes requises lors de la génération d'une application sur IBM i à l'aide de SQL.

Si votre programme contient des instructions EXEC SQL et des appels MQI, procédez comme suit:

1. Convertissez les commandes EXEC SQL en instructions en langage natif. Utilisez la commande CRTSQLCI pour un programme C. Utilisez la commande CRTSQLCBLI pour un programme COBOL.

Incluez `OPTION(*NOGEN)` dans la commande CRTSQLCI ou CRTSQLCBLI. Le traitement s'arrête pour vous permettre d'inclure les programmes de service IBM MQ appropriés. Cette commande place le code, par défaut, dans QTEMP/QSQLTEMP.

2. Compilez le code source à l'aide de la commande CRTCMOD (pour un programme C) ou de la commande CRTCBLMOD (pour un programme COBOL).
3. Utilisez CRTPGM pour lier le code compilé aux programmes de service IBM MQ appropriés. Cette opération crée le programme exécutable.

Voici un exemple de ce code (il compile un programme, SQLTEST, dans la bibliothèque SQLUSER):

```
CRTSQLCI  OBJ(MQTEST/SQLTEST) SRCFILE(SQLUSER/QCSRC) +
     SRCMBR(SQLTEST) OUTPUT(*PRINT) OPTION(*NOGEN)
CRTCMOD   MODULE(MQTEST/SQLTEST) +
     SRCFILE(QTEMP/QSQLTEMP) OUTPUT(*PRINT)
CRTPGM   PGM(MQTEST/SQLTEST) +
     BNDSRVPGM(QMQM/LIBMQIC)
```

Linux Génération de votre application procédurale sur Linux

Ces informations décrivent les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications IBM MQ for Linux à exécuter.

C et C++ sont pris en charge. Pour plus d'informations sur la préparation de vos programmes C + +, voir Utilisation de C++.

Linux *Préparation des programmes C dans Linux*

Les programmes C précompilés sont fournis dans le répertoire *MQ_INSTALLATION_PATH*`/samp/bin` . Pour générer un exemple à partir du code source, utilisez le compilateur `gcc` .

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Travaillez dans votre environnement normal. Pour plus d'informations sur la programmation d'applications 64 bits, voir Normes de codage sur les plateformes 64 bits.

Liaison de bibliothèques

Les tableaux suivants répertorient les bibliothèques requises lors de la préparation des programmes C sous Linux.

- Vous devez lier vos programmes à la bibliothèque appropriée fournie par IBM MQ.

  Dans un environnement sans unités d'exécution, créez un lien vers une seule des bibliothèques suivantes:

Fichier de bibliothèque	Type de programme / d'exit
libmqm.so	Serveur pour C
libmqic.so & libmqm.so	Client pour C

  Dans un environnement à unités d'exécution, créez un lien vers une seule des bibliothèques suivantes:

Fichier de bibliothèque	Type de programme / d'exit
libmqm_r.so	Serveur pour C
libmqic_r.so & libmqm_r.so	Client pour C

  Remarque :

  1. Vous ne pouvez pas créer de lien vers plusieurs bibliothèques. En d'autres termes, vous ne pouvez pas lier à la fois une bibliothèque à unités d'exécution et une bibliothèque non à unités d'exécution.

2. Si vous écrivez un service installable (voir Administration de IBM MQ pour plus d'informations), vous devez établir un lien vers la bibliothèque `libmqmzf.so` .
3. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encina ou BEA Tuxedo, vous devez établir un lien vers `libmqmxa.so` (ou `libmqmxa64.so` si votre gestionnaire de transactions traite le type'long'comme 64 bits) et les bibliothèques `libmqz.so` dans une application sans unités d'exécution et vers `libmqmxa_r.so` (ou `libmqmxa64_r.so`) et des bibliothèques `libmqz_r.so` dans une application à unités d'exécution.
4. Vous devez lier les bibliothèques IBM MQ avant toute autre bibliothèque de produit.

Linux *Génération d'applications 31 bits*

Cette rubrique contient des exemples de commandes utilisées pour générer des programmes 31 bits dans divers environnements.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Application client C, 31 bits, sans unité d'exécution

```
gcc -m31 -o famqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Application client C, 31 bits, avec unités d'exécution

```
gcc -m31 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Application serveur C, 31 bits, sans unités d'exécution

```
gcc -m31 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Application serveur C, 31 bits, avec unités d'exécution

```
gcc -m31 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Application client C++, 31 bits, sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Application client C + +, 31 bits, avec unités d'exécution

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Application serveur C + +, 31 bits, sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Application serveur C + +, 31 bits, à unités d'exécution

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
```

```
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 31 bits, sans unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Exit client C, 31 bits, avec unités d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Exit de serveur C, 31 bits, sans unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Exit de serveur C, 31 bits, unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Génération d'applications 32 bits*

Cette rubrique contient des exemples de commandes utilisées pour générer des programmes 32 bits dans divers environnements.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Application client C, 32 bits, sans unités d'exécution

```
gcc -m32 -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Application client C, 32 bits, avec unités d'exécution

```
gcc -m32 -o amqsputc_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Application serveur C, 32 bits, sans unités d'exécution

```
gcc -m32 -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Application serveur C, 32 bits, à unités d'exécution

```
gcc -m32 -o amqsput_32_r amqsput0.c -I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Application client C++ 32 bits, sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Application client C++ 32 bits, avec unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Application serveur C + +, 32 bits, sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Application serveur C + +, 32 bits, avec unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 32 bits, sans unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic
```

Exit client C, 32 bits, avec unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Exit de serveur C, 32 bits, sans unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm
```

Exit de serveur C, 32 bits, à unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -Wl,-rpath= MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Linux *Génération d'applications 64 bits*

Cette rubrique contient des exemples de commandes utilisées pour générer des programmes 64 bits dans divers environnements.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Application client C, 64 bits, sans unités d'exécution

```
gcc -m64 -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Application client C, 64 bits, avec unités d'exécution

```
gcc -m64 -o amqsputc_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Application serveur C, 64 bits, sans unités d'exécution

```
gcc -m64 -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Application serveur C, 64 bits, à unités d'exécution

```
gcc -m64 -o amqsput_64_r amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Application client C++ 64 bits, sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application client C++ 64 bits, avec unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Application serveur C++ 64 bits, sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Application serveur C++ 64 bits, avec unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 64 bits, sans unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Exit client C, 64 bits, avec unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/cliexit_64_r cliexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Exit de serveur C, 64 bits, sans unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
```

```
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Exit de serveur C, 64 bits, avec unités d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/srvexit_64_r srvexit.c
-I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Linux *Préparation de programmes COBOL dans Linux*

Découvrez comment préparer des programmes COBOL dans Linux et des programmes COBOL à l'aide d' IBM COBOL for Linux sur x86 et Micro Focus COBOL.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

1. Les fichiers de stockage COBOL 32 bits sont installés dans le répertoire suivant:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   et des liens symboliques sont créés dans:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. Sur les plateformes 64 bits, les fichiers de stockage COBOL 64 bits sont installés dans le répertoire suivant:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. Dans les exemples suivants, définissez COBCPY sur:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   pour les applications 32 bits, et:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   pour les applications 64 bits.

Vous devez lier votre programme à l'un des éléments suivants:

Fichier de bibliothèque	Type de programme / d'exit
libmqmcb.so	Serveur pour COBOL
libmqicb.so	Client pour COBOL
libmqmcb_r.so	Server for COBOL (application à unités d'exécution)
libmqicb_r.so	Client for COBOL (application à unités d'exécution)

Préparation de programmes COBOL à l'aide d' IBM COBOL for Linux sur x86

Des exemples de programmes COBOL sont fournis avec IBM MQ. Pour compiler un tel programme, entrez la commande appropriée dans la liste suivante:

Application de serveur non à unités d'exécution 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

Application client non à unités d'exécution 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqicb -ICOBCPY_VALUE
```

Application de serveur à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqmcb_r -ICOBCPY_VALUE
```

Application client à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-qTHREAD -L MQ_INSTALLATION_PATH/lib -lmqicb_r -ICOBCPY_VALUE
```

Préparation de programmes COBOL à l'aide de Micro Focus COBOL

Définissez les variables d'environnement avant de compiler votre programme comme suit:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Pour compiler un programme COBOL 32 bits, s'il est pris en charge, à l'aide de Micro Focus COBOL, entrez:

```
$ 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
```

Pour compiler un programme COBOL 64 bits à l'aide de Micro Focus COBOL, entrez:

```
$ 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
```

où `amqsput` est un exemple de programme

Consultez la documentation Micro Focus COBOL pour obtenir une description des variables d'environnement dont vous avez besoin.

Windows Génération de votre application procédurale sur Windows

Les publications du système Windows expliquent comment générer des applications exécutables à partir des programmes que vous écrivez.

Cette rubrique décrit les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications IBM MQ for Windows à exécuter sous les systèmes Windows . Les langages de programmation C, C + +, COBOL et Visual Basic sont pris en charge. Pour plus d'informations sur la préparation de vos programmes C + +, voir Utilisation de C++.

Les tâches que vous devez effectuer pour créer une application exécutable à l'aide de IBM MQ for Windows varient en fonction du langage de programmation dans lequel votre code source est écrit. En plus de coder les appels MQI dans votre code source, vous devez ajouter les instructions de langage appropriées pour inclure les fichiers d'inclusion IBM MQ for Windows pour la langue que vous utilisez. Familiarisez-vous avec le contenu de ces fichiers. Pour une description complète, voir «Fichiers de définition de données IBM MQ», à la page 735 .

Windows *Génération d'applications 64 bits sous Windows*

Les applications 32 bits et 64 bits sont prises en charge sur IBM MQ for Windows. Les fichiers exécutables et les fichiers de bibliothèque IBM MQ sont fournis dans des formats 32 bits et 64 bits. Utilisez la version appropriée en fonction de l'application que vous utilisez.

Fichiers exécutables et bibliothèques

Les versions 32 bits et 64 bits des bibliothèques IBM MQ sont fournies dans les emplacements suivants:

Tableau 148. Emplacement des bibliothèques IBM MQ	
Version de bibliothèque	**Répertoire contenant les fichiers de bibliothèque**
32 bits	*MQ_INSTALLATION_PATH* \Tools\Lib
64 bits	*MQ_INSTALLATION_PATH* \Tools\Lib64

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Les applications 32 bits continuent de fonctionner normalement après la migration. Les fichiers 32 bits se trouvent dans le même répertoire que dans les versions précédentes du produit.

Si vous souhaitez créer une version 64 bits, vous devez vous assurer que votre environnement est configuré pour utiliser les fichiers de bibliothèque dans *MQ_INSTALLATION_PATH* \Tools\Lib64. Vérifiez que la variable d'environnement LIB n'est pas définie pour rechercher dans le dossier contenant les bibliothèques 32 bits.

Windows *Préparation des programmes C dans Windows*

Travaillez dans votre environnement Windows standard ; IBM MQ for Windows ne nécessite rien de spécial.

Pour plus d'informations sur la programmation des applications 64 bits, voir Codage des normes sur les plateformes 64 bits.

- Liez vos programmes aux bibliothèques appropriées fournies par IBM MQ:

Fichier de bibliothèque	Type de programme / d'exit
MQ_INSTALLATION_PATH \Tools\Lib\mqm.lib	serveur pour C 32 bits
MQ_INSTALLATION_PATH \Tools\Lib\mqic.lib	client pour C 32 bits
MQ_INSTALLATION_PATH \Tools\Lib\mqicxa.lib	client pour C 32 bits avec coordination de transaction
MQ_INSTALLATION_PATH \Tools\Lib64\mqm.lib	serveur pour C 64 bits
MQ_INSTALLATION_PATH \Tools\Lib64\mqic.lib	client pour C 64 bits
MQ_INSTALLATION_PATH \Tools\Lib64\mqicxa.lib	client pour C 64 bits avec coordination de transaction

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

La commande suivante fournit un exemple de compilation de l'exemple de programme amqsget0 (à l'aide du compilateur Microsoft Visual C++).

Pour les applications 32 bits:

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib
```

Pour les applications 64 bits :

```
cl -MD amqsget0.c -Feamqsget.exe MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib
```

Remarque :

- Si vous écrivez un service installable (voir Administration de IBM MQ pour plus d'informations), vous devez vous connecter à la bibliothèque mqmzf.lib .
- Si vous créez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encina ou BEA Tuxedo, vous devez établir une liaison à la bibliothèque mqmxa.lib ou mqmxa.lib .
- Si vous écrivez un exit CICS , accédez à la bibliothèque mqmcics4.lib .
- Vous devez lier les bibliothèques IBM MQ avant toute autre bibliothèque de produit.

- Les DLL doivent se trouver dans le chemin (PATH) que vous avez indiqué.
- Si vous utilisez des caractères minuscules dans la mesure du possible, vous pouvez passer de IBM MQ for Windows à IBM MQ for AIX or Linux , où l'utilisation de minuscules est nécessaire.

Préparation des programmes CICS et Transaction Server

L'exemple de source C pour une transaction CICS IBM MQ est fourni par AMQSCIC0.CCS. Vous le générez à l'aide des fonctions CICS standard. Par exemple, pour TXSeries for Windows 2000:

1. Définissez la variable d'environnement (entrez le code suivant sur une seule ligne):

   ```
   set CICS_IBMC_FLAGS=-I MQ_INSTALLATION_PATH\Tools\C\Include;
   %CICS_IBMC_FLAGS%
   ```

2. Définissez la variable d'environnement USERLIB:

   ```
   set USERLIB=MQM.LIB;%USERLIB%
   ```

3. Traduisez, compilez et liez l'exemple de programme:

   ```
   cicstcl -l IBMC amqscic0.ccs
   ```

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Ceci est décrit dans le document *Transaction Server for Windows NT Application Programming Guide (CICS) V4*.

Pour plus d'informations sur la prise en charge des transactions CICS , voir Administration de IBM MQ.

Windows *Préparation de programmes COBOL dans Windows*

Utilisez ces informations pour apprendre à préparer des programmes COBOL dans Windowset à préparer des programmes CICS et Transaction Server.

1. Les fichiers de stockage COBOL 32 bits sont installés dans le répertoire suivant: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. Les fichiers de stockage COBOL 64 bits sont installés dans le répertoire suivant: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64

3. Dans les exemples suivants, définissez CopyBook sur:

```
CopyBook
```

pour les applications 32 bits, et:

```
CopyBook64
```

pour les applications 64 bits.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour préparer des programmes COBOL sur des systèmes Windows , liez votre programme à l'une des bibliothèques suivantes fournies par IBM MQ:

Fichier de bibliothèque	Type de programme ou d'exit
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	Serveur 32 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib\mqiccb	Client 32 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib64\mqmcb	Serveur 64 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH \Tools\Lib64\mqiccb	Client 64 bits pour Micro Focus COBOL

Lorsque vous exécutez un programme dans l'environnement client MQI, vérifiez que la bibliothèque DOSCALLS apparaît avant toute bibliothèque COBOL ou IBM MQ .

Préparation de programmes COBOL à l'aide de Micro Focus COBOL

Reliez tous les programmes IBM MQ Micro Focus COBOL 32 bits existants à l'aide de `mqmcb.lib` ou `mqiccb.lib`, plutôt que les bibliothèques `mqmcbb` et `mqiccbb` .

Pour compiler, par exemple, l'exemple de programme `amq0put0`à l'aide de Micro Focus COBOL:

1. Définissez la variable d'environnement COBCPY pour qu'elle pointe vers les fichiers de stockage COBOL IBM MQ (entrez le code suivant sur une ligne):

```
set COBCPY= MQ_INSTALLATION_PATH\
Tools\Cobol\Copybook
```

2. Compilez le programme pour vous fournir un fichier objet:

```
cobol amq0put0 LITLINK
```

3. Liez le fichier objet au système d'exécution.
   - Définissez la variable d'environnement LIB pour qu'elle pointe vers les bibliothèques COBOL du compilateur.
   - Liez le fichier objet à utiliser sur le serveur IBM MQ :

```
cbllink amq0put0.obj mqmcb.lib
```

   - Ou liez le fichier objet à utiliser sur le client IBM MQ :

```
cbllink amq0put0.obj mqiccb.lib
```

Préparation des programmes CICS et Transaction Server

Pour compiler et lier un programme TXSeries for Windows NT, V5.1 à l'aide de IBM VisualAge COBOL:

1. Définissez la variable d'environnement (entrez le code suivant sur une seule ligne):

```
set CICS_IBMCOB_FLAGS= MQ_INSTALLATION_PATH\
Cobol\Copybook\VAcobol;%CICS_IBMCOB_FLAGS%
```

2. Définissez la variable d'environnement USERLIB:

```
  set USERLIB=MQMCBB.LIB
```

3. Traduisez, compilez et liez votre programme:

```
cicstcl -l IBMCOB myprog.ccp
```

Ceci est décrit dans le manuel *Transaction Server for Windows NT, V4 Application Programming* .

Pour compiler et lier un programme CICS for Windows V5 à l'aide de Micro Focus COBOL:

- Définissez la variable INCLUDE:

```
set
INCLUDE=drive:\programname\ibm\websphere\tools\c\include;
        drive:\opt\cics\include;%INCLUDE%
```

- Définissez la variable d'environnement COBCPY:

```
setCOBCPY=drive:\programname\ibm\websphere\tools\cobol\copybook;
          drive:\opt\cics\include
```

- Définissez les options COBOL:
  - `set`
  - `COBOPTS=/LITLINK /NOTRUNC`

et exécutez le code suivant:

```
cicstran cicsmq00.ccp
cobol cicsmq00.cbl /LITLINK /NOTRUNC
cbllink -D -Mcicsmq00 -Ocicsmq00.cbmfnt cicsmq00.obj
%CICSLIB%\cicsprCBMFNT.lib user32.lib msvcrt.lib kernel32.lib mqmcb.lib
```

Windows *Préparation des programmes Visual Basic dans Windows*

Informations à prendre en compte lors de l'utilisation des programmes Microsoft Visual Basic sous Windows.

Deprecated Depuis la IBM MQ 9.0, la prise en charge de Microsoft Visual Basic 6.0 est obsolète. IBM MQ classes for .NET est la technologie de remplacement recommandée. Pour plus d'informations, voir Développement d'applications .NET.

Remarque : Les versions 64 bits des fichiers de module Visual Basic ne sont pas fournies.

Pour préparer des programmes Visual Basic sous Windows:

1. Créer un nouveau projet.
2. Ajoutez le fichier de module fourni, CMQB.BAS, au projet.
3. Ajoutez d'autres fichiers de module fournis si vous en avez besoin:
   - CMQBB.BAS: MQAI
   - CMQCFB.BAS:
   - CMQXB.BAS:
   - CMQPSB.BAS: Publication / abonnement

Pour plus d'informations sur l'utilisation de l'appel MQCONNXAny depuis Visual Basic, voir «Codage dans Visual Basic», à la page 1080 .

Appelez la procédure MQ_SETDEFAULTS avant d'effectuer des appels MQI dans le code de projet. Cette procédure configure les structures par défaut requises par les appels MQI.

Indiquez si vous créez un serveur ou un client IBM MQ , avant de compiler ou d'exécuter le projet, en définissant la variable de compilation conditionnelle *MqType*. Définissez *MqType* dans un projet Visual Basic sur 1 pour un serveur ou 2 pour un client comme suit:

1. Sélectionnez le menu Projet.
2. Sélectionnez *Name* Propriétés (où *Name* est le nom du projet en cours).
3. Sélectionnez l'onglet Make dans la boîte de dialogue.
4. Dans la zone Arguments de compilation conditionnels, entrez ce qui suit pour un serveur:

   ```
   MqType=1
   ```

   ou ceci pour un client:

   ```
   MqType=2
   ```

Concepts associés

«Codage dans Visual Basic», à la page 1080
Informations à prendre en compte lors du codage des programmes IBM MQ dans Microsoft Visual Basic. Visual Basic est pris en charge uniquement sous Windows.

Référence associée

«Liaison d'applications Visual Basic avec le code IBM MQ MQI client», à la page 945
Vous pouvez lier des applications Microsoft Visual Basic avec le code IBM MQ MQI client sous Windows.

Windows *Exit de sécurité SSPI*

IBM MQ for Windows fournit un exit de sécurité pour le IBM MQ MQI client et le serveur IBM MQ .
Il s'agit d'un programme d'exit de canal qui fournit l'authentification pour les canaux IBM MQ à l'aide de l'interface SSPI (Security Services Programming Interface). SSPI fournit les fonctions de sécurité intégrées des systèmes Windows .

Les modules de sécurité sont chargés à partir de security.dll ou de secur32.dll. Ces DLL sont fournies avec votre système d'exploitation.

L'authentification unidirectionnelle est fournie à l'aide des services d'authentification NTLM.
L'authentification bidirectionnelle est fournie à l'aide des services d'authentification Kerberos .

Le programme d'exit de sécurité est fourni au format source et objet. Vous pouvez utiliser le code objet tel qu'il est ou utiliser le code source comme point de départ pour créer vos propres programmes d'exit utilisateur.

Voir aussi «Utilisation de l'exit de sécurité SSPI sous Windows», à la page 1168.

Introduction aux exits de sécurité

Un exit de sécurité établit une connexion sécurisée entre deux programmes d'exit de sécurité, l'un de ces programmes étant destiné à l'agent MCA émetteur et l'autre, à l'agent MCA destinataire.

Le programme qui lance la connexion sécurisée, c'est-à-dire le premier programme à prendre le contrôle après l'établissement de la session MCA, est appelé *initiateur de contexte*. Le programme partenaire est appelé *accepteur de contexte*.

Le tableau suivant présente certains des types de canal qui sont des initiateurs de contexte et leurs accepteurs de contexte associés.

Tableau 149. Initiateurs de contexte et accepteurs de contexte associés	
Initiateur de contexte	**Accepteur de contexte**
MQCHT_CLNTCONN	MQCHT_SVRCONN
MQCHT_RECEIVER	EXPÉDITEUR_MQCH
MQCHT_CLUSRCVR	MQCHT_CLUSSDR

Le programme d'exit de sécurité comporte deux points d'entrée:

- **SCY_NTLM**

  Cela utilise les services d'authentification NTLM, qui fournissent une authentification unidirectionnelle. NTLM permet aux serveurs de vérifier l'identité de leurs clients. Il ne permet pas aux clients de vérifier l'identité d'un serveur ou à un serveur de vérifier l'identité d'un autre serveur. L'authentification NTLM a été conçue pour un environnement réseau dans lequel les serveurs sont supposés être authentiques.

- **SCY_KERBEROS**

  Cela utilise les services d'authentification mutuelle Kerberos . Le protocole Kerberos ne suppose pas que les serveurs d'un environnement réseau sont authentiques. Les parties aux deux extrémités d'une connexion réseau peuvent vérifier l'identité de l'autre partie. C'est-à-dire que les serveurs peuvent vérifier l'identité des clients et d'autres serveurs, et que les clients peuvent vérifier l'identité d'un serveur.

Ce que fait l'exit de sécurité

Cette rubrique décrit les actions des programmes d'exit de canal SSPI.

Les programmes d'exit de canal fournis fournissent une authentification unidirectionnelle ou bidirectionnelle (mutuelle) d'un système partenaire lorsqu'une session est en cours d'établissement. Pour un canal particulier, chaque programme d'exit est associé à un *principal* (similaire à un ID utilisateur, voir «Contrôle d'accès IBM MQ et principaux Windows», à la page 1050). Une connexion entre deux programmes d'exit est une association entre les deux principaux.

Une fois la session sous-jacente établie, une connexion sécurisée est établie entre deux programmes d'exit de sécurité (un pour l'agent MCA émetteur et un pour l'agent MCA récepteur). La séquence des opérations est la suivante:

1. Chaque programme est associé à un principal particulier, par exemple à la suite d'une opération de connexion explicite.
2. L'initiateur de contexte demande une connexion sécurisée avec le partenaire à partir du package de sécurité (pour Kerberos, le partenaire nommé) et reçoit un jeton (appelé token1). Le jeton est envoyé, à l'aide de la session sous-jacente déjà établie, au programme partenaire.
3. Le programme partenaire (l'accepteur de contexte) transmet token1 au package de sécurité, qui vérifie que l'initiateur de contexte est authentique. Pour NTLM, la connexion est maintenant établie.
4. Pour l'exit de sécurité fourni par Kerberos(c'est-à-dire pour l'authentification mutuelle), le package de sécurité génère également un deuxième jeton (appelé token2), que l'accepteur de contexte renvoie à l'initiateur de contexte à l'aide de la session sous-jacente.
5. L'initiateur de contexte utilise token2 pour vérifier que l'accepteur de contexte est authentique.
6. A ce stade, si les deux applications sont satisfaites de l'authenticité du jeton du partenaire, la connexion sécurisée (authentifiée) est établie.

Contrôle d'accès IBM MQ et principaux Windows

Le contrôle d'accès fourni par IBM MQ est basé sur l'utilisateur et le groupe. L'authentification fournie par Windows est basée sur les principaux, tels que l'utilisateur et le nom servicePrincipal(SPN). Dans le cas du nom servicePrincipal, un grand nombre d'entre eux peuvent être associés à un seul utilisateur.

L'exit de sécurité SSPI utilise les principaux Windows appropriés pour l'authentification. Si l'authentification Windows aboutit, l'exit transmet l'ID utilisateur associé au principal Windows à IBM MQ pour le contrôle d'accès.

Les principaux Windows qui sont pertinents pour l'authentification varient en fonction du type d'authentification utilisé.

- Pour l'authentification NTLM, le principal Windows de l'initiateur de contexte est l'ID utilisateur associé au processus en cours d'exécution. Etant donné que cette authentification est unidirectionnelle, le principal associé à l'accepteur de contexte n'est pas pertinent.
- Pour l'authentification Kerberos , sur les canaux CLNTCONN, le principal Windows est l'ID utilisateur associé au processus en cours d'exécution. Sinon, le principal Windows est le nom servicePrincipalqui est formé en ajoutant le préfixe suivant au nom QueueManager.

```
ibmMQSeries/
```

z/OS Building your procedural application on z/OS

The CICS, IMS, and z/OS publications describe how to build applications that run in these environments.

This collection of topics describes the additional tasks, and the changes to the standard tasks, that you must perform when building IBM MQ for z/OS applications for these environments. COBOL, C, C++, Assembler, and PL/I programming languages are supported. (For information about building C++ applications see Utilisation de C++.)

The tasks that you must perform to create an executable IBM MQ for z/OS application depend on both the programming language that the program is written in, and the environment in which the application will run.

In addition to coding the MQI calls in your program, add the appropriate language statements to include the IBM MQ for z/OS data definition file for the language that you are using. Make yourself familiar with the contents of these files. See “Fichiers de définition de données IBM MQ” on page 735 for a full description.

Note

The name **thlqual** is the high-level qualifier of the installation library on z/OS.

z/OS *Preparing your program to run*

After you have written the program for your IBM MQ application to create an executable application, you have to compile or assemble it, then link-edit the resulting object code with the stub program that IBM MQ for z/OS supplies for each environment that it supports.

How you prepare your program depends on both the environment (batch, CICS, IMS(BMP or MPP), Linux or z/OS UNIX System Services) in which the application runs, and the structure of the data sets on your z/OS installation.

“Dynamically calling the IBM MQ stub” on page 1057 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on MQSeries for OS/390®, V5.2 must not be link-edited with a stub program supplied with IBM MQ for z/OS V7.

z/OS *Building 64 bit C applications*

In z/OS, 64 bit C applications are built using the LP64 compiler and binder options. The IBM MQ for z/OS *cmqc.h* header file recognizes when this option is provided to the compiler, and generates IBM MQ data types and structures appropriate for 64 bit operation.

C code built with this option must be built to use dynamic-link libraries (DLLs) appropriate for the coordination semantic required. To achieve this, you bind the compiled code with the appropriate side-deck defined in the following table:

Table 150. Side-deck name required for each coordination semantic

Coordination	**Side-deck name**
Single phase commit MQI	CSQBMQ2X
Two phase commit with RRS coordination, using RRS verbs	CSQBRR2X
Two phase commit with RRS coordination, using MQI verbs	CSQBRI2X

Note: For 31-bit C applications you also set compiler options for the calling interface (either Language Environment or XPLINK), as described in "Building z/OS batch applications using 31-bit Language Environment or XPLINK" on page 1053. For 64-bit C applications you do not specify the calling interface, because the only supported linkage is XPLINK.

Use the EDCQCB JCL procedure, supplied with *z/OS XL C/C++*, to build a single phase commit IBM MQ program as a batch job, as follows:

```
//PROCS  JCLLIB ORDER=CBC.SCCNPRC
//CLG EXEC EDCQCB,
// INFILE='thlqual.SCSQC37S(CSQ4BCG1)',   < MQ SAMPLES
// CPARM='RENT,SSCOM,DLL,LP64,LIST,NOMAR,NOSEQ',  < COMPILER OPTIONS
// LIBPRFX='CEE',            < PREFIX FOR LIBRARY DSN
// LNGPRFX='CBC',            < PREFIX FOR LANGUAGE DSN
// BPARM='MAP,XREF,RENT,DYNAM=DLL', < LINK EDIT OPTIONS
// OUTFILE='userid.LOAD(CSQ4BCG1),DISP=SHR'
//COMPILE.SYSLIB DD
//        DD
//        DD DISP=SHR,DSN=thlqual.SCSQC370
//BIND.SCSQDEFS DD DISP=SHR,DSN=thlqual.SCSQDEFS
//BIND.SYSIN DD *
INCLUDE SCSQDEFS(CSQBMQ2X)
NAME CSQ4BCG1
```

To build an RRS coordinated program in z/OS UNIX System Services, compile and link as follows:

```
cc -o mqsamp -W c,LP64,DLL -W l,DYNAM=DLL,LP64 -I"//'thlqual.SCSQC370'" "//'thlqual.SCSQDEFS(CSQBRR2X)'" mqsamp.c
```

z/OS *Building z/OS batch applications*

Learn how to build z/OS batch applications and the steps to consider when doing so.

To build an application for IBM MQ for z/OS that runs under z/OS batch, create job control language (JCL) that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object code created in step "1" on page 1051.
3. For PL/I applications, use the compiler option EXTRN(SHORT).

4. Link-edit the object code created in step “1” on page 1051 (or step “2” on page 1051 for a C application) to produce a load module. When you link-edit the code, you must include one of the IBM MQ for z/OS batch stub programs (CSQBSTUB or one of the RRS stub programs: CSQBRRSI or CSQBRSTB).

   CSQBSTUB
   single-phase commit provided by IBM MQ for z/OS

   CSQBRRSI
   two-phase commit provided by RRS using the MQI

   CSQBRSTB
   two-phase commit provided by RRS directly

   Notes:

   a. If you use CSQBRSTB, you must also link-edit your application with ATRSCSS from SYS1.CSSLIB. Figure 113 on page 1052 and Figure 114 on page 1052 show fragments of JCL to do this. The stubs are language-independent and are supplied in library **thlqual**.SCSQLOAD.

   b. If your application runs under Language Environment, you should ensure you link-edit with the Language Environment DLL instead as described in “Building z/OS batch applications using 31-bit Language Environment or XPLINK” on page 1053.

5. Store the load module in an application load library.

```
.
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
.
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
.
/*
```

Figure 113. Fragments of JCL to link-edit the object module in the batch environment, using single-phase commit

```
.
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//CSSLIB    DD  DSN=SYS1.CSSLIB,DISP=SHR
//*
.
//SYSIN     DD *
INCLUDE CSQSTUB(CSQBRSTB)
INCLUDE CSSLIB(ATRSCSS)
.
/*
```

Figure 114. Fragments of JCL to link-edit the object module in the batch environment, using two-phase commit

To run a batch or RRS program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB or JOBLIB data set concatenation.

To run a TSO program, you must include the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD in the STEPLIB used by the TSO session.

To run a batch program from the z/OS UNIX System Services shell, add the libraries **thlqual**.SCSQAUTH and **thlqual**.SCSQLOAD to the STEPLIB specification in your $HOME?.profile like this:

```
STEPLIB= thlqual.SCSQAUTH: thlqual.SCSQLOAD
export STEPLIB
```

z/OS *Building z/OS batch applications using 31-bit Language Environment or XPLINK*

IBM MQ for z/OS provides a set of dynamic link libraries (DLLs) that must be used when you link-edit your applications.

There are two variants of the libraries that allow the application to use one of the following calling interfaces:

- The 31-bit Language Environment calling interface.
- The 31-bit XPLINK calling interface. z/OS XPLINK is a high performance calling convention available for C applications. See XPLINK | NOXPLINK in the z/OS 2.2 documentation.

To use the DLLs, the application is bound or linked against so called *sidedecks*, instead of the stubs provided with earlier versions. The sidedecks are found in the SCSQDEFS library (instead of the SCSQLOAD library).

Table 151. Variants of dynamic link libraries

Commit	31-bit Language Environment DLL	31-bit XPLINK DLL	Equivalent stub name
1 phase commit MQI libraries	CSQBMQ1	CSQBMQ1X	CSQBSTUB
2 phase commit with RRS co-ordination using RRS transaction-control verbs	CSQBRR1	CSQBRR1X	CSQBRSTB
2 phase commit with RRS co-ordination using MQI transaction-control verbs	CSQBRI1	CSQBRI1X	CSQBRRSI

Note: All sidedecks contain a definition of the data conversion entry point, MQXCNVC, previously resolved by including CSQASTUB.

Common issues:

- The following message appears on the job log if your application uses asynchronous message consume (MQCB, MQCTL or MQSUB calls) and the previous DLL interface is not used:

  ```
  CSQB001E Language environment programs running in z/OS batch or z/OS UNIX System Services must
  use the DLL interface to IBM MQ
  ```

  Solution: Rebuild your application using sidedecks instead of stubs as detailed previously.

- At program build time, the following message appears

  ```
  IEW2469E The Attributes of a reference to MQAPI-NAME from section your-code do not match the
  attributes of
  the target symbol
  ```

  Reason: This means that you have compiled your XPLINK program with V701 (or later) version of cmqc.h, but are not binding with sidedecks.

  Solution: Change your program's build file to bind against the appropriate sidedeck from SCSQDEFS instead of a stub from SCSQLOAD

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit Language Environment DLL calling interface:

```
//CLG EXEC EDCCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1)
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **DLL** option. The link-edit uses **DYNAM=DLL** option and the references the **CSQBMQ1** library.

The following sample JCL demonstrates how you can compile and link-edit a C program to use the 31 bit XPLINK DLL calling interface:

```
//CLG  EXEC EDCXCB,
//  INFILE=MYPROGS.CPROGS(MYPROGRAM),
//  CPARM='OPTF(DD:OPTF)',
//  BPARM='XREF,MAP,DYNAM=DLL'           < LINKEDIT OPTIONS
//COMPILE.OPTF DD *
RENT,CHECKOUT(ALL),SSCOM,DEFINE(MVS),NOMARGINS,NOSEQ,XPLINK,DLL
SE(DD:SYSLIBV)
//COMPILE.SYSLIB DD
//               DD
//               DD DISP=SHR,DSN=hlq.SCSQC370
//COMPILE.SYSLIBV DD DISP=SHR,DSN=hlq.BASE.H
/*
//BIND.SYSOBJ DD DISP=SHR,DSN=CEE.SCEEOBJ
//            DD DISP=SHR,DSN=hlq.SCSQDEFS
//BIND.SYSLMOD DD DISP=SHR,DSN=hlq.LOAD(MYPROGAM)
//BIND.SYSIN  DD *
 ENTRY CEESTART
 INCLUDE SYSOBJ(CSQBMQ1X)
 NAME MYPROGAM(R)
//
```

Note: The compilation uses the **XPLINK** and **DLL** options. The link-edit uses **DYNAM=DLL** option and references the **CSQBMQ1X** library.

Ensure that you add the compilation option DLL to each program in the module. Messages such as IEW2456E 9207 SYMBOL CSQ1BAK UNRESOLVED are an indication that you need to check that all of the programs have been compiled with the DLL option.

z/OS *Building CICS applications in z/OS*

Use this information when building CICS applications in z/OS.

To build an application for IBM MQ for z/OS that runs under CICS, you must:

- Translate the CICS commands in your program into the language in which the rest of your program is written.
- Compile or assemble the output from the translator to produce object code.
  - For PL/I programs, use the compiler option EXTRN(SHORT).
  - For C applications, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
- Link-edit the object code to create a load module.

CICS provides a procedure to execute these steps in sequence for each of the programming languages it supports.

- For CICS Transaction Server for z/OS, the *CICS Transaction Server for z/OS System Definition Guide* describes how to use these procedures and the *CICS/ESA Application Programming Guide* gives more information on the translation process.

You must include:

- In the SYSLIB statement of the compilation (or assembly) stage, statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
  - For COBOL, **thlqual**.SCSQCOBC
  - For assembler language, **thlqual**.SCSQMACS
  - For C, **thlqual**.SCSQC370
  - For PL/I, **thlqual**.SCSQPLIC
- In your link-edit JCL, the IBM MQ for z/OS CICS stub program (CSQCSTUB). Figure 115 on page 1055 shows fragments of JCL code to do this. The stub is language-independent and is supplied in library **thlqual**.SCSQLOAD.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING CICS STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQCSTUB)
  ⋮
/*
```

Figure 115. Fragments of JCL to link-edit the object module in the CICS environment

- For CICS versions later than CICS TS 3.2, or, if you want to use IBM MQ message property APIs, or IBM MQ APIs MQCB, MQCTL, MQSTAT, MQSUB or MQSUBR, you must linkedit your object code with the CICS supplied stub, DFHMQSTB and not the IBM MQ supplied CSQCSTUB. For more information about building IBM MQ programs for CICS, see API stub program to access IBM MQ MQI calls in the CICS product documentation.

When you have completed these steps, store the load module in an application load library and define the program to CICS in the usual way.

Before you run a CICS program, your system administrator must define it to CICS as an IBM MQ program and transaction, You can then run it in the typical way.

z/OS *Building IMS (BMP or MPP) applications*

Use this information when building IMS (BMP or MPP) applications.

If you are building batch DL/I programs, see "Building z/OS batch applications" on page 1051. To build other applications that run under IMS (either as a BMP or an MPP), create JCL that performs these tasks:

1. Compile (or assemble) the program to produce object code. The JCL for your compilation must include SYSLIB statements that make the product data definition files available to the compiler. The data definitions are supplied in the following IBM MQ for z/OS libraries:
   - For COBOL, **thlqual**.SCSQCOBC
   - For assembler language, **thlqual**.SCSQMACS
   - For C, **thlqual**.SCSQC370
   - For PL/I, **thlqual**.SCSQPLIC
2. For a C application, prelink the object module created in step "1" on page 1055.

3. For PL/I programs, use the compiler option EXTRN(SHORT).
4. For a C application, if the application is not using XPLINK, use the compiler option DEFINE(MQ_OS_LINKAGE=1).
5. Link-edit the object code created in step “1” on page 1055 (or step “2” on page 1055 for a C/370 application) to produce a load module:
   a. Include the IMS language interface module (DFSLI000).
   b. Include the IBM MQ for z/OS IMS stub program (CSQQSTUB). Figure 116 on page 1056 shows fragments of JCL to do this. The stub is language independent and is supplied in library **thlqual**.SCSQLOAD.

      Note: If you are using COBOL, select the NODYNAM compiler option to enable the linkage editor to resolve references to CSQQSTUB unless you intend to use dynamic linking as described in “Dynamically calling the IBM MQ stub” on page 1057.
6. Store the load module in an application load library.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Figure 116. Fragments of JCL to link-edit the object module in the IMS environment

Before you run an IMS program, your system administrator must define it to IMS as an IBM MQ program and transaction: you can then run it in the typical way.

z/OS *Building z/OS UNIX System Services applications*

Use this information when building z/OS UNIX System Services applications.

To build a C application for IBM MQ for z/OS that runs under z/OS UNIX System Services, compile and link your application as follows:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

where **thlqual** is the high-level qualifier used by your installation.

To run the C program, you need to add the following to your `.profile` file; this should be in your root directory:

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

Note that you need to exit from z/OS UNIX System Services, and enter z/OS UNIX System Services again, for the change to be recognized.

If you want to run multiple shells, add the word export at the beginning of the line, that is:

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

Once this completes successfully you can link the CSQBSTUB and issue IBM MQ calls.

“Dynamically calling the IBM MQ stub” on page 1057 describes an alternative method of making MQI calls in your programs so that you do not need to link-edit an IBM MQ stub. This method is not available for all languages and environments.

Do not link-edit a higher level of stub program than that of the version of IBM MQ for z/OS on which your program is running. For example, a program running on IBM WebSphere MQ for z/OS 7.1 must not be link-edited with a stub program supplied with IBM MQ for z/OS 8.0.

z/OS *Dynamically calling the IBM MQ stub*

Instead of link-editing the IBM MQ stub program with your object code, you can dynamically call the stub from within your program.

You can do this in the batch, IMS, and CICS environments. This facility is not supported in the RRS environment. If your application program uses RRS to coordinate updates, see “RRS Considerations” on page 1061.

However, this method:

- Increases the complexity of your programs
- Increases the storage required by your programs at execution time
- Reduces the performance of your programs
- Means that you cannot use the same programs in other environments

If you call the stub dynamically, the appropriate stub program and its aliases must be available at execution time. To ensure this, include the IBM MQ for z/OS data set SCSQLOAD:

- For batch and IMS, in the STEPLIB concatenation of the JCL.
- For CICS, in the CICS DFHRPL concatenation.

For IMS, ensure that the library containing the dynamic stub (built as described in the information about installing the IMS adapter in Setting up the IMS adapter) is ahead of the data set SCSQLOAD in the STEPLIB concatenation of the region JCL.

Use the names shown in Table 152 on page 1057 when you call the stub dynamically. In PL/I, only declare the call names used in your program.

Table 152. Call names for dynamic linking

MQI call	Batch (non-RRS) dynamic call names	CICS dynamic call names	IMS dynamic call names
MQBACK	CSQBBACK	not supported	Not supported
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Not supported
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	not supported	Not supported
MQCONN	CSQBCONN	CSQCCONN	MQCONN
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Not supported
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGET	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP

Table 152. Call names for dynamic linking (continued)			
MQI call	**Batch (non-RRS) dynamic call names**	**CICS dynamic call names**	**IMS dynamic call names**
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Note: 1. These API calls are available only when using CICS TS 3.2 or later and the CSQCSTUB shipped with CICS must be used. For CICS TS 3.2, APAR PK66866 must be applied. For CICS TS 4.1, APAR PK89844 must be applied.

For examples of how to use this technique, see the following figures:

- Batch and COBOL: see Figure 117 on page 1058
- CICS and COBOL: see Figure 118 on page 1059
- IMS and COBOL: see Figure 119 on page 1059
- Batch and assembler: see Figure 120 on page 1059
- CICS and assembler: see Figure 121 on page 1059
- IMS and assembler: see Figure 122 on page 1060
- Batch and C: Figure 123 on page 1060
- CICS and C: see Figure 124 on page 1060
- IMS and C: see Figure 125 on page 1060
- Batch and PL/I: see Figure 126 on page 1060
- IMS and PL/I: see Figure 127 on page 1061

```
...
      WORKING-STORAGE SECTION.
...
         05 WS-MQOPEN                    PIC X(8) VALUE 'CSQBOPEN'.
...
      PROCEDURE DIVISION.
...
          CALL WS-MQOPEN WS-HCONN
                         MQOD
                         WS-OPTIONS
                         WS-HOBJ
                         WS-COMPCODE
                         WS-REASON.
...
```

Figure 117. Dynamic linking using COBOL in the batch environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'CSQCOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 118. Dynamic linking using COBOL in the CICS environment

```
...
       WORKING-STORAGE SECTION.
...
          05 WS-MQOPEN                     PIC X(8) VALUE 'MQOPEN'.
...
       PROCEDURE DIVISION.
...
           CALL WS-MQOPEN WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
      * ------------------------------------------------------------ *
      *
      *    If the compilation option 'DYNAM' is specified
      *    then you may code the MQ calls as follows
      *
      * ------------------------------------------------------------ *
...
           CALL 'MQOPEN'  WS-HCONN
                          MQOD
                          WS-OPTIONS
                          WS-HOBJ
                          WS-COMPCODE
                          WS-REASON.
...
```

Figure 119. Dynamic linking using COBOL in the IMS environment

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

Figure 120. Dynamic linking using assembly language in the batch environment

```
...
         EXEC CICS LOAD PROGRAM('CSQCOPEN') ENTRY(R15)
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         EXEC CICS RELEASE PROGRAM('CSQCOPEN')
...
```

Figure 121. Dynamic linking using assembly language in the CICS environment

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

Figure 122. Dynamic linking using assembly language in the IMS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqbopen;
...
csqbopen = (CALL_ME *) fetch("CSQBOPEN");
(*csqbopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 123. Dynamic linking using C language in the batch environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * csqcopen;
...
  EXEC CICS LOAD PROGRAM("CSQCOPEN") ENTRY(csqcopen);
(*csqcopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 124. Dynamic linking using C language in the CICS environment

```
...
typedef void CALL_ME();
#pragma linkage(CALL_ME, OS)
...
main()
{
CALL_ME * mqopen;
...
mqopen = (CALL_ME *) fetch("MQOPEN");
(*mqopen)(Hconn,&ObjDesc,Options,&Hobj,&CompCode,&Reason);
...
```

Figure 125. Dynamic linking using C language in the IMS environment

```
...
      DCL CSQBOPEN ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH CSQBOPEN;

      CALL CSQBOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE CSQBOPEN;
```

Figure 126. Dynamic linking using PL/I in the batch environment

```
...
      DCL MQOPEN   ENTRY EXT OPTIONS(ASSEMBLER INTER);
...
      FETCH MQOPEN;

      CALL   MQOPEN(HQM,
                    MQOD,
                    OPTIONS,
                    HOBJ,
                    COMPCODE,
                    REASON);

      RELEASE   MQOPEN;
```

Figure 127. Dynamic linking using PL/I in the IMS environment

z/OS *RRS Considerations*

Consider using this information if your application program uses RRS to coordinate updates.

IBM MQ provides two different stubs for batch programs which need RRS coordination - see "The RRS batch adapter" on page 917. The difference in behavior of later API calls is determined at MQCONN time by the batch adapter from information passed by the stub routine on the MQCONN or MQCONNX API. This means that dynamic API calls are available for batch programs which need RRS coordination, provided that the initial connection to IBM MQ was done by using the appropriate stub. The following example illustrates this:

```
       WORKING-STORAGE SECTION.
           05 WS-MQOPEN       PIC X(8) VALUE 'MQOPEN' .
.
.
.
       PROCEDURE DIVISION.
.
.
.
       *
       * Static call to MQCONN must be resolved by linkage edit to
       * CSQBRSTB or CSQBRRSI for RRS coordination
       *
           CALL 'MQCONN' USING W00-QMGR
                               W03-HCONN
                               W03-COMPCODE
                               W03-REASON.
.
.
.
       *
           CALL WS-MQOPEN  WS-HCONN
                           MQOD
                           WS-OPTIONS
                           WS-HOBJ
                           WS-COMPCODE
                           WS-REASON.
```

z/OS *Debugging your programs*

Use this information to learn about debugging TSO and CICS programs, and an insight into CICS trace.

The main aids to debugging IBM MQ for z/OS application programs are the reason codes returned by each API call. For a list of these, including ideas for corrective action, see:

- Messages, codes achèvement et codes anomalie IBM MQ for z/OS for IBM MQ for z/OS
- Messages et codes anomalie for all other IBM MQ platforms

This topic also suggests other debugging tools to use in particular environments.

Debugging TSO programs

The following interactive debugging tools are available for TSO programs:

- TEST tool
- VS COBOL II interactive debugging tool
- INSPECT interactive debugging tool for C and PL/I programs

Debugging CICS programs

You can use the CICS Execution Diagnostic Facility (CEDF) to test your CICS programs interactively without having to modify the program or program-preparation procedure.

For more information about EDF, see the *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS trace

You will probably also find it helpful to use the CICS Trace Control transaction (CETR) to control CICS trace activity.

For more information about CETR, see *CICS Transaction Server for z/OS CICS-Supplied Transactions* manual.

To determine whether CICS trace is active, display connection status using the CKQC panel. This panel also shows the trace number.

To interpret CICS trace entries, see Table 153 on page 1062.

The CICS trace entry for these values is AP0 *xxx* (where *xxx* is the trace number specified when the CICS adapter was enabled). All trace entries except CSQCTEST are issued by CSQCTRUE. CSQCTEST is issued by CSQCRST and CSQCDSP.

Table 153. CICS adapter trace entries

Name	Description	Trace sequence	Trace data
CSQCABNT	Abnormal termination	Before issuing END_THREAD ABNORMAL to IBM MQ. This is because of the end of the task and an implicit backout could be performed by the application. A ROLLBACK request is included in the END_THREAD call in this case.	Unit of work information. You can use this information when finding out about the status of work. (For example, it can be verified against the output produced by the DISPLAY THREAD command, or the IBM MQ for z/OS log print utility.)
CSQCBACK	Syncpoint backout	Before issuing BACKOUT to IBM MQ for z/OS. This is due to an explicit backout request from the application.	Unit of work information.
CSQCCCRC	Completion code and reason code	After unsuccessful return from API call.	Completion code and reason code.
CSQCCOMM	Syncpoint commit	Before issuing COMMIT to IBM MQ for z/OS. This can be due to a single-phase commit request or the second phase of a two-phase commit request. The request is due to an explicit syncpoint request from the application.	Unit of work information.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCEXER	Execute resolve	Before issuing EXECUTE_RESOLVE to IBM MQ for z/OS.	The unit of work information of the unit of work issuing the EXECUTE_RESOLVE. This is the last indoubt unit of work in the resynchronization process.
CSQCGETW	GET wait	Before issuing CICS wait.	Address of the ECB to be waited on.
CSQCGMGD	GET message data	After successful return from MQGET.	Up to 40 bytes of the message data.
CSQCGMGH	GET message handle	Before issuing MQGET to IBM MQ for z/OS.	Object handle.
CSQCGMGI	Get message ID	After successful return from MQGET.	Message ID and correlation ID of the message.
CSQCINDL	Indoubt list	After successful return from the second INQUIRE_INDOUBT.	The indoubt units of work list.
CSQCINDO	IBM use only		
CSQCINDS	Indoubt list size	After successful return from the first INQUIRE_INDOUBT and the indoubt list is not empty.	Length of the list. Divided by 64 gives the number of indoubt units of work.
CSQCINQH	INQ handle	Before issuing MQINQ to IBM MQ for z/OS.	Object handle.
CSQCLOSH	CLOSE handle	Before issuing MQCLOSE to IBM MQ for z/OS.	Object handle.
CSQCLOST	Disposition lost	During the resynchronization process, CICS informs the adapter that it has been restarted so no disposition information regarding the unit of work being resynchronized is available.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNIND	Disposition not indoubt	During the resynchronization process, CICS informs the adapter that the unit of work being resynchronized should not have been indoubt (that is, perhaps it is still running).	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCNORT	Normal termination	Before issuing END_THREAD NORMAL to IBM MQ for z/OS. This is due to the end of the task and therefore the application might perform an implicit syncpoint commit. A COMMIT request is included in the END_THREAD call in this case.	Unit of work information.

Table 153. CICS adapter trace entries (continued)

Name	Description	Trace sequence	Trace data
CSQCOPNH	OPEN handle	After successful return from MQOPEN.	Object handle.
CSQCOPNO	OPEN object	Before issuing MQOPEN to IBM MQ for z/OS.	Object name.
CSQCPMGD	PUT message data	Before issuing MQPUT to IBM MQ for z/OS.	Up to 40 bytes of the message data.
CSQCPMGH	PUT message handle	Before issuing MQPUT to IBM MQ for z/OS.	Object handle.
CSQCPMGI	PUT message ID	After successful MQPUT from IBM MQ for z/OS.	Message ID and correlation ID of the message.
CSQCPREP	Syncpoint prepare	Before issuing PREPARE to IBM MQ for z/OS in the first phase of two-phase commit processing. This call can also be issued from the distributed queuing component as an API call.	Unit of work information.
CSQCP1MD	PUTONE message data	Before issuing MQPUT1 to IBM MQ for z/OS.	Up to 40 bytes of data of the message.
CSQCP1MI	PUTONE message ID	After successful return from MQPUT1.	Message ID and correlation ID of the message.
CSQCP1ON	PUTONE object name	Before issuing MQPUT1 to IBM MQ for z/OS.	Object name.
CSQCRBAK	Resolved backout	Before issuing RESOLVE_ROLLBACK to IBM MQ for z/OS.	Unit of work information.
CSQCRCMT	Resolved commit	Before issuing RESOLVE_COMMIT to IBM MQ for z/OS.	Unit of work information.
CSQCRMIR	RMI response	Before returning to the CICS RMI (resource manager interface) from a specific invocation.	Architected RMI response value. Its meaning depends of the type of the invocation. These values are documented in the *CICS Transaction Server for z/OS Customization Guide*. To determine the type of invocation, look at previous trace entries produced by the CICS RMI component.
CSQCRSYN	Resynchronization	Before the resynchronization process starts for the task.	Unit of work ID known to CICS for the unit of work being resynchronized.
CSQCSETH	SET handle	Before issuing MQSET to IBM MQ for z/OS.	Object handle.
CSQCTASE	IBM use only		

Table 153. CICS adapter trace entries (continued)			
Name	**Description**	**Trace sequence**	**Trace data**
CSQCTEST	Trace test	Used in EXEC CICS ENTER TRACE call to verify the trace number supplied by the user or the trace status of the connection.	No data.
CSQCDCFF	IBM use only		

Traitement des erreurs de programme de procédure

Ces informations expliquent les erreurs associées aux appels MQI de vos applications lors de l'exécution d'un appel ou de la distribution de son message à sa destination finale.

Dans la mesure du possible, le gestionnaire de files d'attente renvoie des erreurs dès qu'un appel MQI est effectué. Il s'agit d' *erreurs déterminées localement*.

Lors de l'envoi de messages à une file d'attente éloignée, des erreurs peuvent ne pas apparaître lorsque l'appel MQI est effectué. Dans ce cas, le gestionnaire de files d'attente qui identifie les erreurs les signale en envoyant un autre message au programme d'origine. Il s'agit d' *erreurs déterminées à distance*.

Erreurs déterminées localement

Informations sur les erreurs déterminées en local qui incluent: l'échec d'un appel MQI, les interruptions du système et les messages contenant des données incorrectes.

Les trois causes les plus courantes d'erreurs que le gestionnaire de files d'attente peut signaler immédiatement sont les suivantes:

- Echec d'un appel MQI ; par exemple, parce qu'une file d'attente est saturée
- Interruption de l'exécution d'une partie du système dont dépend votre application ; par exemple, le gestionnaire de files d'attente
- Messages contenant des données qui ne peuvent pas être traitées avec succès

Si vous utilisez la fonction d'insertion asynchrone, les erreurs ne sont pas signalées immédiatement. Utilisez l'appel MQSTAT pour extraire des informations de statut sur les opérations d'insertion asynchrone précédentes.

Echec d'un appel MQI

Le gestionnaire de files d'attente peut signaler immédiatement toute erreur dans le codage d'un appel MQI. Cette opération est effectuée à l'aide d'un ensemble de codes retour prédéfinis. Ces codes sont divisés en codes achèvement et codes raison.

Pour indiquer si un appel aboutit, le gestionnaire de files d'attente renvoie un *code achèvement* à la fin de l'appel. Il existe trois codes achèvement indiquant la réussite, l'achèvement partiel et l'échec de l'appel. Le gestionnaire de files d'attente renvoie également un *code anomalie* qui indique la raison de l'achèvement partiel ou de l'échec de l'appel.

Les codes achèvement et raison de chaque appel sont répertoriés avec la description de cet appel dans Codes retour. Pour des informations plus détaillées, y compris des idées de mesures correctives, voir:

- z/OS Messages, codes achèvement et codes anomalie IBM MQ for z/OS pour IBM MQ for z/OS
- Messages et codes anomalie pour toutes les autres plateformes IBM MQ

Concevez vos programmes pour gérer tous les codes retour qui peuvent survenir à chaque appel.

Nterruptions System i

Il se peut que votre application ne détecte aucune interruption si le gestionnaire de files d'attente auquel elle est connectée doit effectuer une reprise après une défaillance du système. Toutefois, vous devez concevoir votre application pour vous assurer que vos données ne sont pas perdues si une telle interruption se produit.

Les méthodes que vous pouvez utiliser pour vous assurer que vos données restent cohérentes dépendent de la plateforme sur laquelle votre gestionnaire de files d'attente s'exécute:

z/OS

Dans les environnements CICS et IMS , vous pouvez effectuer des appels MQPUT et MQGET dans des unités de travail gérées par CICS ou IMS. Dans l'environnement de traitement par lots, vous pouvez effectuer des appels MQPUT et MQGET de la même manière, mais vous devez déclarer des points de synchronisation à l'aide de la commande suivante:

- Les appels IBM MQ for z/OS MQCMIT et MQBACK (voir «Validation et annulation d'unités de travail», à la page 876), ou
- Les services RRS (z/OS Transaction Management and Recoverable Resource Manager Services) fournissent une prise en charge des points de synchronisation en deux phases. RRS vous permet de mettre à jour IBM MQ et d'autres ressources de produit compatibles RRS, telles que les ressources de procédure mémorisée Db2 , au sein d'une seule unité d'oeuvre logique. Pour plus d'informations sur la prise en charge des points de synchronisation RRS, voir «Transaction management and recoverable resource manager services», à la page 881.

IBM i

Vous pouvez effectuer vos appels MQPUT et MQGET dans des unités d'oeuvre globales gérées par le contrôle de validation IBM i . Vous pouvez déclarer des points de synchronisation à l'aide des commandes IBM i COMMIT et ROLLBACK natives ou des commandes spécifiques à la langue. Les unités d'oeuvre locales sont gérées par IBM MQ à l'aide des appels MQCMIT et MQBACK.

AIX, Linux, and Windows systèmes

Dans ces environnements, vous pouvez effectuer vos appels MQPUT et MQGET de la manière habituelle, mais vous devez déclarer des points de synchronisation à l'aide des appels MQCMIT et MQBACK (voir «Validation et annulation d'unités de travail», à la page 876). Dans l'environnement CICS , les commandes MQCMIT et MQBACK sont désactivées, car vous pouvez effectuer vos appels MQPUT et MQGET dans des unités de travail gérées par CICS.

Utilisez des messages persistants pour transporter toutes les données que vous ne pouvez pas vous permettre de perdre. Les messages persistants sont réintégrés dans les files d'attente si le gestionnaire de files d'attente doit effectuer une reprise après incident. ALW Avec IBM MQ on AIX, Linux, and Windows, un appel MQGET ou MQPUT dans votre application échouera au point de remplir tous les fichiers journaux, avec le message MQRC_RESOURCE_PROBLEM. Pour plus d'informations sur les fichiers journaux sous AIX, Linux, and Windows, voir Administration de IBM MQ. z/OS Pour z/OS , voir Planification sur z/OS.

Si le gestionnaire de files d'attente est arrêté par un opérateur alors qu'une application est en cours d'exécution, l'option de mise au repos est généralement utilisée. Le gestionnaire de files d'attente entre dans un état de mise au repos dans lequel les applications peuvent continuer à travailler, mais elles doivent s'arrêter dès que cela leur convient. Les petites applications rapides peuvent probablement ignorer l'état de mise au repos et continuer jusqu'à ce qu'elles s'arrêtent normalement. Les applications plus longues, ou celles qui attendent l'arrivée de messages, doivent utiliser l'option *fail if quiescing* lorsqu'elles utilisent les appels MQOPEN, MQPUT, MQPUT1et MQGET. Ces options signifient que les appels échouent lorsque le gestionnaire de files d'attente est mis au repos, mais l'application peut encore avoir le temps de s'arrêter proprement en émettant des appels qui ignorent l'état de mise au repos. Ces applications peuvent également valider ou annuler les modifications qu'elles ont apportées, puis s'arrêter.

Si l'arrêt du gestionnaire de files d'attente est forcé (arrêt sans mise au repos), les applications reçoivent le code anomalie MQRC_CONNECTION_BROKEN lorsqu'elles effectuent des appels MQI.

Quittez l'application ou, sur les systèmes IBM i IBM MQ for IBM i, AIX, Linux, and Windows , émettez un appel MQDISC.

Messages contenant des données incorrectes

Lorsque vous utilisez des unités de travail dans votre application, si un programme ne parvient pas à traiter un message qu'il extrait d'une file d'attente, l'appel MQGET est annulé.

Le gestionnaire de files d'attente gère un comptage (dans la zone *BackoutCount* du descripteur de message) du nombre de fois qu'il se produit. Il gère ce nombre dans le descripteur de chaque message affecté. Ce nombre peut fournir des informations précieuses sur l'efficacité d'une application. Les messages dont le nombre d'annulations augmente avec le temps sont rejetés à plusieurs reprises ; concevez votre application de sorte qu'elle en analyse les raisons et traite ces messages en conséquence.

z/OS Sous IBM MQ for z/OS, pour que le nombre d'annulations survit aux redémarrages du gestionnaire de files d'attente, définissez l'attribut **HardenGetBackout** sur MQQA_BACKOUT_HARDENED; sinon, si le gestionnaire de files d'attente doit redémarrer, il ne conserve pas un nombre d'annulations précis pour chaque message. La définition de l'attribut de cette manière ajoute la pénalité de traitement supplémentaire.

Sur les systèmes IBM MQ for IBM i IBM i, AIX, Linux, and Windows , le nombre d'annulations survit toujours aux redémarrages du gestionnaire de files d'attente.

z/OS En outre, sous IBM MQ for z/OS, lorsque vous supprimez des messages d'une file d'attente au sein d'une unité d'oeuvre, vous pouvez marquer un message de sorte qu'il ne soit plus disponible si l'unité d'oeuvre est annulée par l'application. Le message marqué est traité comme s'il avait été extrait sous une nouvelle unité de travail. Vous marquez le message à ignorer à l'aide de l'option MQGMO_MARK_SKIP_BACKOUT(dans la structure MQGMO) lorsque vous utilisez l'appel MQGET. Pour plus d'informations sur cette technique, voir «Annulation ignorée», à la page 820 .

Utilisation des messages de rapport pour l'identification des incidents

Le gestionnaire de files d'attente éloignées ne peut pas signaler des erreurs telles que l'échec de l'insertion d'un message dans une file d'attente lorsque vous effectuez votre appel MQI, mais il peut vous envoyer un message de rapport indiquant comment il a traité votre message.

Dans votre application, vous pouvez créer des messages de rapport (MQPUT) et sélectionner l'option permettant de les recevoir (auquel cas ils sont envoyés par une autre application ou par un gestionnaire de files d'attente).

Création de messages de rapport

Les messages de rapport permettent à une application d'indiquer à une autre application qu'elle ne peut pas traiter le message qui a été envoyé.

Toutefois, la zone *Report* doit d'abord être analysée pour déterminer si l'application qui a envoyé le message souhaite être informée de tout problème. Après avoir déterminé qu'un message de rapport est requis, vous devez décider:

- Que vous souhaitiez inclure l'intégralité du message d'origine, uniquement les 100 premiers octets de données ou aucun des messages d'origine.
- Que faire avec le message d'origine. Vous pouvez le supprimer ou le laisser dans la file d'attente des messages non livrés.
- Indique si le contenu des zones *MsgId* et *CorrelId* est également nécessaire.

Utilisez la zone *Feedback* pour indiquer la raison pour laquelle le message de rapport est généré. Placez vos messages de rapport dans la file d'attente de réponse d'une application. Pour plus d'informations, voir Commentaires en retour .

Demande et réception de messages de rapport (MQGET)

Lorsque vous envoyez un message à une autre application, vous n'êtes informé d'aucun problème sauf si vous renseignez la zone *Report* pour indiquer les commentaires en retour dont vous avez besoin. Pour connaître les options disponibles, voir Structure de la zone de rapport .

Les gestionnaires de files d'attente placent toujours des messages de rapport dans la file d'attente de réponses d'une application et il est recommandé que vos propres applications fassent de même. Lorsque vous utilisez la fonction de génération de messages de rapport, indiquez le nom de votre file d'attente de réponses dans le descripteur de message de votre message ; sinon, l'appel MQPUT échoue.

Votre application doit contenir des procédures qui surveillent votre file d'attente de réponse et traitent les messages qui lui parviennent. N'oubliez pas qu'un message de rapport peut contenir tous les messages d'origine, les 100 premiers octets du message d'origine ou aucun des messages d'origine.

Le gestionnaire de files d'attente définit la zone *Feedback* du message de rapport pour indiquer la raison de l'erreur ; par exemple, la file d'attente cible n'existe pas. Vos programmes devraient faire de même.

Pour plus d'informations sur les messages de rapport, voir «Messages de rapport», à la page 21.

Erreurs déterminées à distance

Lorsque vous envoyez des messages à une file d'attente éloignée, même lorsque le gestionnaire de files d'attente local a traité votre appel MQI sans trouver d'erreur, d'autres facteurs peuvent influencer la façon dont votre message est géré par un gestionnaire de files d'attente éloignées.

Par exemple, la file d'attente que vous ciblez peut être saturée ou même inexistante. Si votre message doit être traité par d'autres gestionnaires de files d'attente intermédiaires sur la route vers la file d'attente cible, une erreur peut être détectée.

Problèmes lors de la distribution d'un message

Lorsqu'un appel MQPUT échoue, vous pouvez essayer de placer à nouveau le message dans la file d'attente, de le renvoyer à l'expéditeur ou de le placer dans la file d'attente de rebut.

Chaque option a ses avantages, mais vous ne souhaiterez peut-être pas réessayer d'insérer un message si l'échec de MQPUT est dû au fait que la file d'attente de destination est saturée. Dans ce cas, le fait de le placer dans la file d'attente des messages non livrés vous permet de le distribuer ultérieurement à la file d'attente de destination appropriée.

Relancer la distribution du message

Avant que le message ne soit inséré dans une file d'attente de rebut, un gestionnaire de files d'attente éloignées tente de le placer à nouveau dans la file d'attente si les attributs *MsgRetryCount* et *MsgRetryInterval* ont été définis pour le canal ou s'il existe un programme d'exit de relance à utiliser (dont le nom est conservé dans la zone *MsgRetryExitId* de l'attribut de canal).

Si la zone *MsgRetryExitId* est vide, les valeurs des attributs *MsgRetryCount* et *MsgRetryInterval* sont utilisées.

Si la zone *MsgRetryExitId* n'est pas vide, le programme d'exit de ce nom s'exécute. Pour plus d'informations sur l'utilisation de vos propres programmes d'exit, voir «Programmes d'exit de canal pour les canaux de messagerie», à la page 988.

Renvoyer le message à l'expéditeur

Vous renvoyez un message à l'expéditeur en demandant qu'un message de rapport soit généré pour inclure tous les messages d'origine.

Pour plus de détails sur les options de message de rapport, voir «Messages de rapport», à la page 21 .

Utilisation de la file d'attente de rebut (messages non livrés)

Lorsqu'un gestionnaire de files d'attente ne parvient pas à distribuer un message, il tente de le placer dans sa file d'attente de rebut. Cette file d'attente doit être définie lorsque le gestionnaire de files d'attente est installé.

Vos programmes peuvent utiliser la file d'attente de rebut de la même manière que le gestionnaire de files d'attente l'utilise. Vous pouvez trouver le nom de la file d'attente de rebut en ouvrant l'objet gestionnaire de files d'attente (à l'aide de l'appel MQOPEN) et en demandant l'attribut **DeadLetterQName** (à l'aide de l'appel MQINQ).

Lorsque le gestionnaire de files d'attente insère un message dans cette file d'attente, il ajoute un en-tête au message, dont le format est décrit par la structure d'en-tête de rebut (MQDLH) ; voir MQDLH-en-tête de rebut. Cet en-tête inclut le nom de la file d'attente cible et la raison pour laquelle le message a été inséré dans la file d'attente de rebut. Il doit être supprimé et le problème doit être résolu avant que le message ne soit inséré dans la file d'attente prévue. De plus, le gestionnaire de files d'attente modifie la zone *Format* du descripteur de message (MQMD) pour indiquer que le message contient une structure MQDLH.

Structure MQDLH

Il est recommandé d'ajouter une structure MQDLH à tous les messages que vous placez dans la file d'attente de rebut. Toutefois, si vous prévoyez d'utiliser le gestionnaire de rebut fourni par certains produits IBM MQ , vous devez ajouter une structure MQDLH à vos messages.

L'ajout de l'en-tête à un message peut rendre le message trop long pour la file d'attente des messages non livrés. Par conséquent, assurez-vous toujours que vos messages sont plus courts que la taille maximale autorisée pour la file d'attente des messages non livrés, d'au moins la valeur de la constante MQ_MSG_HEADER_LENGTH. La taille maximale des messages autorisés dans une file d'attente est déterminée par la valeur de l'attribut **MaxMsgLength** de la file d'attente. Pour la file d'attente de rebut, assurez-vous que cet attribut est défini sur la valeur maximale autorisée par le gestionnaire de files d'attente. Si votre application ne peut pas distribuer de message et que le message est trop long pour être inséré dans la file d'attente de rebut, suivez les conseils donnés dans la description de la structure MQDLH.

Vérifiez que la file d'attente des messages non livrés est surveillée et que tous les messages qui y arrivent sont traités. Le gestionnaire de files d'attente de rebut s'exécute en tant qu'utilitaire de traitement par lots et peut être utilisé pour effectuer diverses actions sur les messages sélectionnés de la file d'attente de rebut. Pour plus de détails, voir «Traitement de la file d'attente de rebut», à la page 1069.

Si une conversion de données est nécessaire, le gestionnaire de files d'attente convertit les informations d'en-tête lorsque vous utilisez l'option MQGMO_CONVERT sur l'appel MQGET. Si le processus d'insertion du message est un agent MCA, l'en-tête est suivi de tout le texte du message d'origine.

Les messages placés dans la file d'attente de rebut peuvent être tronqués s'ils sont trop longs pour cette file d'attente. Une indication possible de cette situation est que les messages de la file d'attente de rebut ont la même longueur que la valeur de l'attribut **MaxMsgLength** de la file d'attente.

Traitement de la file d'attente de rebut
Ces informations contiennent des informations générales sur l'interface de programmation lors de l'utilisation du traitement de la file d'attente de rebut.

Le traitement de la file d'attente de rebut dépend de la configuration système locale requise, mais tenez compte des points suivants lorsque vous rédigez la spécification:

- Le message peut être identifié comme ayant un en-tête de file d'attente de rebut car la valeur de la zone de format dans le MQMD est MQFMT_DEAD_LETTER_HEADER.
- Sous IBM MQ for z/OS avec CICS, si un agent MCA insère ce message dans la file d'attente de rebut, la zone *PutApplType* est MQAT_CICS et la zone *PutApplName* est la *ApplId* du système CICS , suivie du nom de transaction de l'agent MCA.
- La raison pour laquelle le message doit être acheminé vers la file d'attente de rebut est contenue dans la zone *Reason* de l'en-tête de la file d'attente de rebut.
- L'en-tête de la file d'attente de rebut contient des détails sur le nom de la file d'attente de destination et le nom du gestionnaire de files d'attente.

- L'en-tête de file d'attente de rebut contient des zones qui doivent être réintégrées dans le descripteur de message avant que le message ne soit inséré dans la file d'attente de destination. Il s'agit des fonctions suivantes :
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- Le descripteur de message est identique à PUT par l'application d'origine, à l'exception des trois zones affichées (Encoding, CodedCharSetIdet Format).

Votre application de file d'attente de rebut doit effectuer une ou plusieurs des opérations suivantes:

- Examinez la zone *Reason* . Un message peut avoir été inséré par un agent MCA pour les raisons suivantes:
  - La taille du message était supérieure à la taille maximale du message pour le canal

    La raison est MQRC_MSG_TOO_BIG_FOR_CHANNEL
  - Le message n'a pas pu être inséré dans sa file d'attente de destination

    Il s'agit d'un code anomalie MQRC_ * qui peut être renvoyé par une opération MQPUT .
  - Un exit utilisateur a demandé cette action

    Le code anomalie est celui fourni par l'exit utilisateur ou le MQRC_SUPPRESSED_BY_EXIT par défaut
- Essayez de transmettre le message à sa destination prévue, là où cela est possible.
- Conservez le message pendant un certain temps avant de le supprimer lorsque le motif de la déviation est déterminé, mais qu'il n'est pas immédiatement corrigeable.
- Donnez des instructions aux administrateurs pour qu'ils corrigent les problèmes lorsqu'ils ont été déterminés.
- Supprimez les messages endommagés ou qui ne sont pas processibles.

Il existe deux façons de traiter les messages que vous avez récupérés de la file d'attente des messages non livrés:

1. Si le message concerne une file d'attente locale:
   - Effectuer les traductions de code requises pour extraire les données d'application
   - Effectuer des conversions de code sur ces données s'il s'agit d'une fonction locale
   - Insertion du message résultant dans la file d'attente locale avec tous les détails du descripteur de message restaurés
2. Si le message concerne une file d'attente éloignée, placez le message dans la file d'attente.

Pour plus d'informations sur la façon dont les messages non distribués sont gérés dans un environnement de mise en file d'attente répartie, voir Que se passe-t-il lorsqu'un message ne peut pas être distribué?.

Programmation multidiffusion

Utilisez ces informations pour en savoir plus sur les tâches de programmation de multidiffusion IBM MQ telles que la connexion à un gestionnaire de files d'attente et la génération de rapports d'exceptions.

IBM MQ Multicast a été conçu pour être aussi transparent que possible pour l'utilisateur tout en restant compatible avec les applications existantes. La définition d'un objet COMMINFO et des paramètres **MCAST** et **COMMINFO** de l'objet TOPIC signifie que les applications IBM MQ existantes ne nécessitent pas de réécriture substantielle pour utiliser la multidiffusion. Toutefois, il peut y avoir des limitations (voir «Multidiffusion et MQI», à la page 1071 pour plus d'informations) et des problèmes de sécurité à prendre en compte (voir Sécurité multidiffusion pour plus d'informations).

Multidiffusion et MQI

Utilisez ces informations pour comprendre les principaux concepts de l'interface MQI (Message Queue Interface) et leur relation avec IBM MQ Multicast.

Les abonnements multidiffusion sont non durables ; comme aucune file d'attente physique n'est impliquée, il n'est pas possible de stocker les messages hors ligne créés par les abonnements durables.

Une fois qu'une application s'est abonnée à une rubrique de multidiffusion, elle reçoit un descripteur d'objet qu'elle peut utiliser ou à partir duquel elle peut utiliser MQGET, comme s'il s'agissait d'un descripteur d'une file d'attente. Cela signifie que seuls les abonnements multidiffusion gérés (abonnements créés avec MQSO_MANAGED) sont pris en charge, c'est-à-dire qu'il n'est pas possible de créer un abonnement et de'pointer'les messages sur une file d'attente. Cela signifie que les messages doivent être consommés à partir du descripteur d'objet renvoyé lors de l'appel d'abonnement. Sur le client, les messages sont stockés dans une mémoire tampon de messages jusqu'à ce qu'ils soient consommés par le client ; voir la section MessageBuffer du fichier de configuration du client pour plus d'informations. Si le client ne parvient pas à suivre le débit de publication, les messages sont supprimés selon les besoins, les messages les plus anciens étant supprimés en premier.

Il s'agit normalement d'une décision d'administration si une application utilise la multidiffusion ou non, spécifiée en définissant l'attribut MCAST d'un objet TOPIC. Si une application de publication doit s'assurer que la multidiffusion n'est pas utilisée, elle peut utiliser l'option `MQOO_NO_MULTICAST` . De même, une application d'abonnement peut s'assurer que la multidiffusion n'est pas utilisée en s'abonnant avec l'option `MQSO_NO_MULTICAST` .

IBM MQ Multicast prend en charge l'utilisation des sélecteurs de message. Un sélecteur est utilisé par une application pour enregistrer son intérêt uniquement dans les messages dont les propriétés satisfont la requête SQL92 que la chaîne de sélection représente. Pour plus d'informations sur les sélecteurs de message, voir «Sélecteurs», à la page 32.

Le tableau suivant répertorie tous les principaux concepts MQI et la manière dont ils sont liés à la multidiffusion:

Tableau 154. Concepts MQI et leurs relations avec la multidiffusion

Concept MQI	**Action lors d'une tentative d'utilisation de la multidiffusion**	**Code raison**
Insertion d'un message de longueur nulle	Refusé	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Regroupement	Refusé	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	Refusé	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Listes de diffusion	Refusé	2154 (086A) (RC2154): MQRC_RECS_Présentat_ERROR
MQINQ	Rejeté pour les descripteurs de rubriques: MQINQ et MQSET des rubriques n'est pas pris en charge.	2038 (07F6) (RC2038): MQRC_NOT_OPEN_FOR_INQUIRE

Tableau 154. Concepts MQI et leurs relations avec la multidiffusion (suite)

Concept MQI	**Action lors d'une tentative d'utilisation de la multidiffusion**	**Code raison**
MQINQ	Accepté pour le descripteur géré. Seule l'option `Profondeur en cours` peut être interrogée.	• Si la valeur est Profondeur en cours, il n'y a pas de code raison applicable. • Si la valeur est autre que la profondeur en cours, le code anomalie est 2067 (0813) (RC2067): MQRC_SELECTOR_ERROR.
MQSET	Rejeté pour tous les descripteurs.	2040 (07F8) (RC2040): MQRC_NOT_OPEN_FOR_SET
Transactions (XA ou non)	Refusé	2072 (0818) (RC2072) : MQRC_SYNCPOINT_NOT_AVAILABLE
Navigation dans les messages	Refusé	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Messages de verrouillage	Refusé	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Exploration avec marque	Refusé	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Transmettre un contexte	Refusé	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Rejeté. Il n'est pas valide d'essayer MQPUT1 dans une rubrique de multidiffusion uniquement.	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
Abonnement durable	Rejeté si la rubrique est marquée comme "Multidiffusion uniquement", sinon un abonnement non multidiffusion est effectué.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	Rejeté. Si la chaîne de rubrique comporte plus de 255 caractères, elle est rejetée dans le client.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR

Tableau 154. Concepts MQI et leurs relations avec la multidiffusion (suite)		
Concept MQI	**Action lors d'une tentative d'utilisation de la multidiffusion**	**Code raison**
Abonnement non géré effectué	Rejeté si la rubrique est marquée comme "Multidiffusion uniquement", sinon un abonnement non multidiffusion est effectué.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Refusé	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Les éléments suivants développent certains des concepts MQI du tableau précédent et fournissent des informations sur certains des concepts MQI qui ne figurent pas dans le tableau:

Persistance des messages
Pour les abonnés multidiffusion non durables, les messages persistants du diffuseur sont distribués de manière irrémédiable.

Troncature de message
La troncature de message est prise en charge, ce qui signifie qu'une application peut:

1. Emettez une requête MQGET.
2. Echec de l'obtention de MQRC_TRUNCATED_MSG_FAILED.
3. Allouez une mémoire tampon plus importante.
4. Emettez à nouveau la commande MQGET pour extraire le message.

Expiration de l'abonnement
L'expiration de l'abonnement n'est pas prise en charge. Toute tentative de définition d'une expiration est ignorée.

Haute disponibilité pour la multidiffusion

Utilisez ces informations pour comprendre l'opération IBM MQ Multicast d'égal à égal continu ; bien que IBM MQ se connecte à un gestionnaire de files d'attente IBM MQ , les messages ne transitent pas par ce gestionnaire de files d'attente.

Bien qu'une connexion à un gestionnaire de files d'attente doit être établie pour que MQOPEN ou MQSUB l'objet de rubrique de multidiffusion, les messages eux-mêmes ne transitent pas par le gestionnaire de files d'attente. Par conséquent, une fois que MQOPEN ou MQSUB est terminé sur l'objet de rubrique de multidiffusion, il est possible de continuer à transmettre des messages de multidiffusion même si la connexion au gestionnaire de files d'attente a été perdue. Il existe deux modes de fonctionnement:

Une connexion normale est établie avec le gestionnaire de files d'attente
La communication multidiffusion est possible lorsque la connexion au gestionnaire de files d'attente existe. Si la connexion échoue, les règles MQI normales sont appliquées, par exemple ; une instruction MQPUT sur le descripteur d'objet multidiffusion renvoie 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Une connexion client de reconnexion est établie avec le gestionnaire de files d'attente
La communication multidiffusion est possible même pendant le cycle de reconnexion. Cela signifie que même lorsque la connexion au gestionnaire de files d'attente a été interrompue, l'insertion et la consommation de messages de multidiffusion ne sont pas affectées. Le client tente de se reconnecter

à un gestionnaire de files d'attente et si cette reconnexion échoue, le descripteur de connexion est interrompu et tous les appels MQI, y compris les appels multidiffusion, échouent. Pour plus d'informations, voir: Reconnexion automatique du client

Si une application émet explicitement un MQDISC, tous les abonnements de multidiffusion et les descripteurs d'objet sont fermés.

Opération d'égal à égal continue de multidiffusion

L'un des avantages de la communication d'égal à égal entre les clients est que les messages n'ont pas besoin d'être transmis via le gestionnaire de files d'attente ; par conséquent, si la connexion au gestionnaire de files d'attente est interrompue, le transfert de messages se poursuit. Les restrictions suivantes s'appliquent aux exigences de message continu de ce mode:

- La connexion doit être établie à l'aide de l'une des options MQCNO_RECONNECT_ * pour une opération continue. Ce processus signifie que même si la session de communication peut être interrompue, le descripteur de connexion réel n'est pas interrompu et est à la place à l'état de reconnexion. Si la reconnexion échoue, le descripteur de connexion est désormais rompu, ce qui empêche tous les autres appels MQI.
- Seuls MQPUT, MQGET, MQINQ et Async Consume sont pris en charge dans ce mode. Toutes les instructions MQOPEN, MQCLOSE ou MQDISC nécessitent une reconnexion au gestionnaire de files d'attente pour se terminer.
- Le statut passe à l'arrêt du gestionnaire de files d'attente ; tout état dans le gestionnaire de files d'attente peut donc être périmé ou manquant. Cela signifie que les clients peuvent envoyer et recevoir des messages et qu'aucun statut n'est connu sur le gestionnaire de files d'attente. Pour plus d'informations, voir: Surveillance des applications de multidiffusion

Conversion de données dans l'interface MQI pour la messagerie multidiffusion

Utilisez ces informations pour comprendre le fonctionnement de la conversion de données pour la messagerie IBM MQ Multicast.

IBM MQ Multicast est un protocole partagé et sans connexion. Par conséquent, il n'est pas possible pour chaque client d'effectuer des demandes spécifiques de conversion de données. Chaque client abonné au même flux de multidiffusion reçoit les mêmes données binaires ; par conséquent, si une conversion de données IBM MQ est requise, la conversion est effectuée localement sur chaque client.

Les données sont converties sur le client pour le trafic de multidiffusion IBM MQ . Si l'option **MQGMO_CONVERT** est spécifiée, la conversion des données est effectuée comme demandé. Les formats définis par l'utilisateur nécessitent que l'exit de conversion de données soit installé sur le client ; voir «Ecriture des exits de conversion de données», à la page 1010 pour plus d'informations sur les bibliothèques qui se trouvent désormais dans les packages client et serveur.

Pour plus d'informations sur l'administration de la conversion de données, voir Activation de la conversion de données pour la messagerie multidiffusion.

Pour plus d'informations sur la conversion de données, voir Conversion de données.

Pour plus d'informations sur les exits de conversion de données et `ClientExitPath`, voir la section ClientExitPath du fichier de configuration client.

Génération de rapports sur les exceptions de multidiffusion

Utilisez ces informations pour en savoir plus sur les gestionnaires d'événements de multidiffusion IBM MQ et la génération de rapports sur les exceptions de multidiffusion IBM MQ .

IBM MQ Multicast aide à l'identification des problèmes en appelant le gestionnaire d'événements pour signaler les événements de multidiffusion qui sont signalés à l'aide du mécanisme de gestionnaire d'événements IBM MQ standard.

Un événement de multidiffusion individuel peut entraîner l'appel de plusieurs événements IBM MQ car il peut exister plusieurs descripteurs de connexion MQHCONN utilisant le même émetteur ou récepteur de multidiffusion. Toutefois, chaque exception de multidiffusion entraîne l'appel d'un seul gestionnaire d'événements par connexion IBM MQ .

La constante IBM MQ MQCBDO_EVENT_CALL permet aux applications d'enregistrer un rappel pour recevoir uniquement des événements IBM MQ et aux applications MQCBDO_MC_EVENT_CALL d'enregistrer un rappel pour recevoir uniquement des événements de multidiffusion. Si les deux constantes sont utilisées, les deux types d'événement sont reçus.

Demande d'événements de multidiffusion

IBM MQ Les événements de multidiffusion utilisent la constante MQCBDO_MC_EVENT_CALL dans la zone cbd.Options . L'exemple suivant montre comment demander des événements de multidiffusion:

```
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);
```

Lorsque l'option MQCBDO_MC_EVENT_CALL est spécifiée pour la zone cbd.Options , le gestionnaire d'événements est envoyé uniquement aux événements de multidiffusion IBM MQ au lieu des événements de niveau connexion. Pour demander que les deux types d'événement soient envoyés au gestionnaire d'événements, l'application doit spécifier la constante MQCBDO_EVENT_CALL dans la zone cbd.Options ainsi que la constante MQCBDO_MC_EVENT_CALL , comme illustré dans l'exemple suivant:

```
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 aucune de ces constantes n'est utilisée, seuls les événements de niveau connexion sont envoyés au gestionnaire d'événements.

Pour plus d'informations sur les valeurs de la zone Options , voir Options (MQLONG).

Format d'événement de multidiffusion

Les exceptions de multidiffusion IBM MQ incluent des informations de support qui sont renvoyées dans le paramètre **Buffer** de la fonction de rappel. Le pointeur **Buffer** pointe vers un tableau de pointeurs et la zone MQCBC.DataLength indique la taille, en octets, du tableau. Le premier élément du tableau pointe toujours vers une brève description textuelle de l'événement. D'autres paramètres peuvent être fournis en fonction du type d'événement. Le tableau suivant répertorie les exceptions:

Tableau 155. Descriptions des codes d'événement de multidiffusion

Code d'événement	Description	Données supplémentaires
MQMCEV_PACKET_LOSS	Perte de paquet irrécupérable	Nombre de paquets perdus
MQMCEV_HEARTBEAT_TIMEOUT	Absence prolongée de paquet de contrôle du signal de présence	Non disponible
MQMCEV_VERSION_CONFLIT	Réception de paquets de version de protocole plus récents	Non disponible
FIABILITÉ_MQMCEV_	Différents modes de fiabilité de l'émetteur et du récepteur	Non disponible

Tableau 155. Descriptions des codes d'événement de multidiffusion (suite)

Code d'événement	Description	Données supplémentaires
MQMCEV_FERME_TRANS	La transmission de rubrique est fermée par 1 source	Non disponible
Erreur MQMCEV_STREAM_ERROR	Erreur détectée sur le flux	Non disponible
MQMCEV_NOUVEAU_SOURCE	Une nouvelle source commence à transmettre sur le sujet	Structure source
MQMCEV_RECEIVE_QUEUE_TRIMMED	Paquets supprimés de PacketQ en raison de l'expiration du temps ou de l'espace	Nombre de paquets tronqués
MQMCEV_PACKET_LOSS_NACK_EXPIRE	Perte de paquet irrécupérable due à l'expiration de NACK	Nombre de paquets perdus
MQMCEV_ACK_RETRIES_EXCEEDED	Paquets supprimés de l'historique après le dépassement de la valeur **max_ack_retries**	Nombre de paquets supprimés
MQMCEV_STREAM_SUSPEND_NACK	Les accusés de réception négatifs ont été suspendus sur un flux accepté par cette rubrique	ID de flux de suspension Temps en millisecondes pendant lequel le flux est suspendu
MQMCEV_STREAM_RESUME_NACK	Les accusés de réception négatifs ont été repris après avoir été suspendus dans un flux	ID de flux
MQMCEV_STREAM_EXPULSÉ	Un flux accepté par cette rubrique a été rejeté en raison d'une demande d'expulsion	ID de flux
MQMCEV_PREMIER_MESSAGE	Premier message d'une source	Numéro de message
MQMCEV_LATE_JOIN_FAILURE	Echec du démarrage de la session de jointure tardive	Non disponible
MQMCEV_MESSAGE_PERTE	Perte de message irrémédiable	Nombre de messages perdus
MQMCEV_SEND_PACKET_FAILURE	L'émetteur de multidiffusion n'est pas parvenu à envoyer un paquet de multidiffusion	Non disponible
MQMCEV_REPAIR_DELAY	Le récepteur de multidiffusion n'a pas reçu de paquet de réparation pour un NAK en attente	Non disponible
MQMCEV_MEMORY_ALERT_ON	Les tampons de réception du récepteur sont en train de se remplir	Pourcentage d'utilisation du pool de mémoire tampon
MQMCEV_MEMORY_ALERT_OFF	Les tampons de réception du récepteur sont arrêtés à la normale	Pourcentage d'utilisation du pool de mémoire tampon

Tableau 155. Descriptions des codes d'événement de multidiffusion (suite)

Code d'événement	Description	Données supplémentaires
MQMCEV_NACK_ALERT_ON	Le taux de demandes de paquets de réparation du récepteur a atteint la cote d'alerte haute	Taux actuel de demandes de réparation en paquets par seconde
MQMCEV_NACK_ALERT_OFF	Le taux de demandes de paquets de réparation du récepteur est inférieur à la normale	Taux actuel de demandes de réparation en paquets par seconde
MQMCEV_REPAIR_ALERT_ON	Le débit d'envoi des paquets de réparation de l'émetteur a atteint la cote d'alerte haute	Non disponible
MQMCEV_REPAIR_ALERT_OFF	Le débit d'envoi des paquets de réparation de l'émetteur est inférieur à la normale	Non disponible
MQMCEV_SHM_DEST_INUTILISABLE	La région de mémoire partagée utilisée par une destination de rubrique de l'émetteur a été détectée comme étant inutilisable	Non disponible
MQMCEV_PORT_SHM_INUTILISABLE	Le port de mémoire partagée utilisé par une instance de récepteur a été détecté comme étant inutilisable	Non disponible
MQMCEV_CCT_GETTIME_FAILED	Echec de l'obtention de l'heure à partir du temps de cluster coordonné	Non disponible
MQMCEV_DEST_INTERFACE_FAILURE	L'interface réseau utilisée par une destination de sujet de l'émetteur a échoué et une interface réseau de secours n'est pas disponible	
MQMCEV_DEST_INTERFACE_FAILOVER	L'interface réseau utilisée par une destination de rubrique d'émetteur a échoué et une reprise en ligne réussie vers une autre interface a été effectuée	
MQMCEV_PORT_INTERFACE-ECHEC	L'interface réseau utilisée par un récepteur rmmPort a échoué et une interface réseau de secours n'est pas disponible (ou a également échoué)	Configuration deRMM
MQMCEV_PORT_INTERFACE_FAILOVER	L'interface réseau utilisée par un récepteur rmmPort a échoué et une reprise en ligne réussie vers une autre interface a été effectuée	Configuration deRMM

Codage en C

Notez les informations des sections suivantes lors du codage des programmes IBM MQ dans C.

Paramètres des appels MQI

Les paramètres *d'entrée uniquement* et de type MQHCONN, MQHOBJ, MQHMSG ou MQLONG sont transmis par valeur ; pour tous les autres paramètres, l' *adresse* du paramètre est transmise par valeur.

Tous les paramètres transmis par adresse n'ont pas besoin d'être spécifiés chaque fois qu'une fonction est appelée. Lorsqu'un paramètre particulier n'est pas requis, un pointeur null peut être spécifié en tant que paramètre sur l'appel de fonction, à la place de l'adresse des données de paramètre. Les paramètres pour lesquels cela est possible sont identifiés dans les descriptions d'appel.

Aucun paramètre n'est renvoyé comme valeur de la fonction ; dans la terminologie C, cela signifie que toutes les fonctions sont vides.

Les attributs de la fonction sont définis par la variable de macro MQENTRY ; la valeur de cette variable de macro dépend de l'environnement.

Paramètres avec type de données non défini

Les fonctions MQGET, MQPUT et MQPUT1 comportent chacune un paramètre **Buffer** dont le type de données n'est pas défini. Ce paramètre est utilisé pour envoyer et recevoir les données de message de l'application.

Les paramètres de ce type sont présentés dans les exemples C sous la forme de tableaux de MQBYTE. Vous pouvez déclarer les paramètres de cette manière, mais il est généralement plus pratique de les déclarer comme la structure qui décrit la présentation des données dans le message. Le paramètre de fonction est déclaré comme un pointeur vers vide, de sorte que l'adresse de toute donnée peut être spécifiée comme paramètre sur l'appel de fonction.

types de données

Tous les types de données sont définis avec l'instruction `typedef` .

Pour chaque type de données, le type de données de pointeur correspondant est également défini. Le nom du type de données de pointeur est le nom du type de données élémentaire ou de structure précédé de la lettre P pour désigner un pointeur. Les attributs du pointeur sont définis par la variable de macro MQPOINTER ; la valeur de cette variable de macro dépend de l'environnement. Le code suivant montre comment déclarer des types de données de pointeur:

```
#define MQPOINTER                  /* depends on environment */
...
typedef MQLONG  MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;     /* pointer to MQMD */
```

Manipulation des chaînes binaires

Les chaînes de données binaires sont déclarées comme l'un des types de données MQBYTEn.

Chaque fois que vous copiez, comparez ou définissez des zones de ce type, utilisez les fonctions C `memcpy`, `memcmp`ou `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));
```

N'utilisez pas les fonctions de chaîne strcpy, strcmp, strncpy ou strncmp car elles ne fonctionnent pas correctement avec les données déclarées en tant que MQBYTE24.

Manipulation des chaînes de caractères

Lorsque le gestionnaire de files d'attente renvoie des données de type caractère à l'application, il remplit toujours les données de type caractère avec des blancs sur la longueur définie de la zone. Le gestionnaire de files d'attente ne renvoie pas de chaînes à terminaison nulle, mais vous pouvez les utiliser dans votre entrée. Par conséquent, lors de la copie, de la comparaison ou de la concaténation de ces chaînes, utilisez les fonctions de chaîne strncpy, strncmp ou strncat.

N'utilisez pas les fonctions de chaîne qui nécessitent que la chaîne se termine par une valeur null (strcpy, strcmp et strcat). En outre, n'utilisez pas la fonction strlen pour déterminer la longueur de la chaîne ; utilisez plutôt la fonction sizeof pour déterminer la longueur de la zone.

Valeurs initiales pour les structures

Le fichier d'inclusion <cmqc.h> définit diverses variables de macro que vous pouvez utiliser pour fournir des valeurs initiales pour les structures lors de la déclaration d'instances de ces structures. Ces variables macro ont des noms de la forme MQxxx_DEFAULT, où MQxxx représente le nom de la structure. Utilisez-les comme suit:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

Pour certaines zones alphanumériques, l'interface MQI définit des valeurs particulières qui sont valides (par exemple, pour les zones *StrucId* ou pour la zone *Format* dans MQMD). Pour chacune des valeurs valides, deux variables de macro sont fournies:

- Une variable de macro définit la valeur sous la forme d'une chaîne de longueur, à l'exclusion de la valeur null implicite, qui correspond exactement à la longueur définie de la zone. Dans les exemples suivants, le symbole ¬ représente un caractère blanc unique:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Utilisez ce formulaire avec les fonctions memcpy et memcmp.
- L'autre variable de macro définit la valeur comme un tableau de caractères ; le nom de cette variable de macro est le nom de la forme de chaîne suffixé avec _ARRAY. Exemple :

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Utilisez ce formulaire pour initialiser la zone lorsqu'une instance de la structure est déclarée avec des valeurs différentes de celles fournies par la variable macro MQMD_DEFAULT.

Valeurs initiales pour les structures dynamiques

Lorsqu'un nombre variable d'instances d'une structure est requis, les instances sont généralement créées dans la mémoire principale obtenue dynamiquement à l'aide des fonctions calloc ou malloc.

Pour initialiser les champs dans de telles structures, la technique suivante est recommandée:

1. Déclarez une instance de la structure à l'aide de la variable de macro MQxxx_DEFAULT appropriée pour initialiser la structure. Cette instance devient le *modèle* pour les autres instances:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                     /* declare model instance */
   ```

   Codez les mots clés statiques ou automatiques dans la déclaration pour donner à l'instance de modèle une durée de vie statique ou dynamique, selon les besoins.
2. Utilisez les fonctions calloc ou malloc pour obtenir du stockage pour une instance dynamique de la structure:

   ```
   PMQMD InstancePtr;
   InstancePtr = malloc(sizeof(MQMD));
                                     /* get storage for dynamic instance */
   ```
3. Utilisez la fonction memcpy pour copier l'instance de modèle dans l'instance dynamique:

   ```
   memcpy(InstancePtr,&ModelMsgDesc,sizeof(MQMD));
                                     /* initialize dynamic instance */
   ```

Utiliser à partir de C++

Pour le langage de programmation C + +, les fichiers d'en-tête contiennent les instructions supplémentaires suivantes qui sont incluses uniquement lorsqu'un compilateur C++ est utilisé:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Codage dans Visual Basic

Informations à prendre en compte lors du codage des programmes IBM MQ dans Microsoft Visual Basic. Visual Basic est pris en charge uniquement sous Windows.

Remarque :

Stabilized Depuis IBM WebSphere MQ 7.0, en dehors de l'environnement .NET , la prise en charge de Visual Basic (VB) a été stabilisée au niveau IBM WebSphere MQ 6.0 . La plupart des nouvelles fonctions ajoutées à IBM WebSphere MQ 7.0 ou à une version ultérieure ne sont pas disponibles pour les applications VB. Si vous programmez dans VB.NET, utilisez les classes IBM MQ pour .NET. Pour plus d'informations, voir Développement d'applications .NET.

Deprecated Depuis la IBM MQ 9.0, la prise en charge de Microsoft Visual Basic 6.0 est obsolète. IBM MQ classes for .NET est la technologie de remplacement recommandée.

Pour éviter une traduction non intentionnelle des données binaires transmises entre Visual Basic et IBM MQ, utilisez une définition MQBYTE à la place de MQSTRING. CMQB.BAS définit plusieurs nouveaux types MQBYTE qui sont équivalents à une définition d'octet C et les utilise dans les structures IBM MQ . Par

exemple, pour la structure MQMD (descripteur de message), MsgId (identificateur de message) est défini comme MQBYTE24.

Visual Basic n'ayant pas de type de données de pointeur, les références à d'autres structures de données IBM MQ sont effectuées par décalage et non par pointeur. Déclarez une structure composée composée des deux structures de composant et spécifiez la structure composée sur l'appel. La prise en charge de IBM MQ pour Visual Basic fournit un appel MQCONNXAny pour rendre cela possible et permettre aux applications client de spécifier les propriétés de canal sur une connexion client. Il accepte une structure sans type (MQCNOCD) à la place de la structure MQCNO standard.

La structure MQCNOCD est une structure composée d'un MQCNO suivi d'un MQCD. Cette structure est déclarée dans le fichier d'en-tête des exits CMQXB. Utilisez la routine MQCNOCD_DEFAULTS pour initialiser une structure MQCNOCD. Un exemple d'appel MQCONNX est fourni (amqscnxb.vbp).

MQCONNXAny possède les mêmes paramètres que MQCONNX, sauf que le paramètre **ConnectOpts** est déclaré comme étant de type de données Any plutôt que de type de données MQCNO. Cela permet à la fonction d'accepter la structure MQCNO ou MQCNOCD. Cette fonction est déclarée dans le fichier d'en-tête principal CMQB.

Concepts associés

«Préparation des programmes Visual Basic dans Windows», à la page 1047
Informations à prendre en compte lors de l'utilisation des programmes Microsoft Visual Basic sous Windows.

Référence associée

«Liaison d'applications Visual Basic avec le code IBM MQ MQI client», à la page 945
Vous pouvez lier des applications Microsoft Visual Basic avec le code IBM MQ MQI client sous Windows.

Codage en COBOL

Notez les informations de la section suivante lors du codage des programmes IBM MQ en COBOL.

Constante nommée

Les noms des constantes sont affichés avec le caractère de soulignement (_) dans le nom. En COBOL, vous devez utiliser le trait d'union (-) à la place du trait de soulignement. Les constantes qui ont des valeurs de chaîne de caractères utilisent le caractère de guillemet simple (') comme délimiteur de chaîne. Pour que le compilateur accepte ce caractère, utilisez l'option de compilation APOST.

Le fichier de copie CMQV contient des déclarations des constantes nommées en tant qu'éléments level-10 . Pour utiliser les constantes, déclarez l'élément level-01 explicitement, puis utilisez l'instruction COPY pour copier les déclarations des constantes:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Cependant, cette méthode fait que les constantes occupent de la mémoire dans le programme même si elles ne sont pas référencées. Si les constantes sont incluses dans de nombreux programmes distincts au sein d'une même unité d'exécution, plusieurs copies de ces constantes existeront, ce qui peut entraîner l'utilisation d'une quantité importante de mémoire principale. Vous pouvez éviter cela en ajoutant la clause GLOBAL à la déclaration level-01 :

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

Cela n'alloue de la mémoire qu'à *un* ensemble de constantes au sein de l'unité d'exécution. Toutefois, les constantes peuvent être référencées par *n'importe quel* programme au sein de l'unité d'exécution, et pas seulement par le programme qui contient la déclaration level-01 .

Assurer l'alignement de la structure

Veillez à ce que les structures IBM MQ qui sont transmises pour démarrer sur les appels MQ soient alignées sur les limites de mot. Une limite de mot est de 4 octets pour les processus 32 bits, de 8 octets pour les processus 64 bits et de 16 octets pour les processus 128 bits (IBM i).

Dans la mesure du possible, placez toutes les structures IBM MQ ensemble de sorte qu'elles soient toutes alignées sur les limites.

Coding in System/390 assembler language (Message queue interface)

Note the information in the following sections when coding IBM MQ for z/OS programs in assembler language.

Names

The names of parameters in the descriptions of calls, and the names of fields in the descriptions of structures are shown in mixed case. In the assembler-language macros supplied with IBM MQ, all names are in uppercase.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention.

In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

Declaring constants

Most constants are declared as equates in macro CMQA.

However, the following constants cannot be defined as equates, and these are not included when you call the macro using default options:

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- MQFMT_ADMIN
- MQFMT_COMMAND_1

- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

To include them, add the keyword EQUONLY=NO when you call the macro.

CMQA is protected against multiple declaration, so you can include it many times. However, the keyword EQUONLY takes effect only the first time that the macro is included.

Specifying the name of a structure

To allow more than one instance of a structure to be declared, the macro that generates the structure prefixes the name of each field with a user-specifiable string and an underscore character (_).

Specify the string when you invoke the macro. If you do not specify a string, the macro uses the name of the structure to construct the prefix:

```
* Declare two object descriptors
CMQODA         Prefix used="MQOD_" (the default)
MY_MQOD CMQODA         Prefix used="MY_MQOD_"
```

The structure declarations in Call descriptions show the default prefix.

Specifying the form of a structure

The macros can generate structure declarations in one of two forms, controlled by the DSECT parameter:

DSECT=YES
An assembler-language DSECT instruction is used to start a new data section; the structure definition immediately follows the DSECT statement. No storage is allocated, so no initialization is possible. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO
Assembler-language DC instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which you can specify by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

DSECT=NO is assumed if the DSECT parameter is not specified.

Controlling the listing

You can control the appearance of the structure declaration in the assembler-language listing with the LIST parameter:

LIST=YES
The structure declaration appears in the assembler-language listing.

LIST=NO
The structure declaration does not appear in the assembler-language listing. This is assumed if the LIST parameter is not specified.

Specifying initial values for fields

You can specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message descriptor structure with the *MsgType* field initialized with MQMT_REQUEST, and the *ReplyToQ* field initialized with the string MY_REPLY_TO_QUEUE, use the following code:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

If you specify a named constant (or equate) as a value on the macro invocation, use the CMQA macro to define the named constant. You must not enclose in single quotation marks (' ') values that are character strings.

Writing reenterable programs

IBM MQ uses its structures for both input and output. If you want your program to remain reenterable:

1. Define working storage versions of the structures as DSECTs, or define the structures inline within an already-defined DSECT. Then copy the DSECT to storage that is obtained using:
   - For batch and TSO programs, the STORAGE or GETMAIN z/OS assembler macros
   - For CICS, the working storage DSECT (DFHEISTG) or the EXEC CICS GETMAIN command

   To correctly initialize these working storage structures, copy a constant version of the corresponding structure to the working storage version.

   Note: The MQMD and MQXQH structures are each more than 256 bytes long. To copy these structures to storage, use the MVCL assembler instruction.
2. Reserve space in storage by using the LIST form (MF=L) of the CALL macro. When you use the CALL macro to make an MQI call, use the EXECUTE form (MF=E) of the macro, using the storage reserved earlier, as shown in the example under “Using CEDF” on page 1084. For more examples of how to do this, see the assembler language sample programs as shipped with IBM MQ.

Use the assembler language RENT option to help you to determine if your program is reenterable.

For information on writing reenterable programs, see *z/OS MVS Application Development Guide: Assembler Language Programs*.

Using CEDF

If you want to use the CICS-supplied transaction, CEDF (CICS Execution Diagnostic Facility) to help you to debug your program, add the ,VL keyword to each CALL statement, for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

The previous example is reenterable assembler-language code where PARMAREA is an area in the working storage that you specified.

Using the MQI calls

The MQI is a call interface, so assembler-language programs must observe the OS linkage convention. In particular, before they issue an MQI call, assembler-language programs must point register R13 at a save area of at least 18 full words. This save area provides storage for the called program. It stores the registers of the caller before their contents are destroyed, and restores the contents of the caller's registers on return.

Note: This is important for CICS assembler-language programs that use the DFHEIENT macro to set up their dynamic storage, but that choose to override the default DATAREG from R13 to other registers. When the CICS Resource Manager Interface receives control from the stub, it saves the current contents of the registers at the address to which R13 is pointing. Failing to reserve a proper save area for this purpose gives unpredictable results, and will probably cause an abend in CICS.

IBM i Codage des programmes IBM MQ en RPG (IBM i uniquement)

Dans la documentation IBM MQ , les paramètres des appels, les noms des types de données, les zones des structures et les noms des constantes sont tous décrits à l'aide de leurs noms longs. Dans RPG, ces noms sont abrégés en six caractères majuscules ou moins.

Par exemple, la zone *MsgType* devient *MDMT* dans RPG. Pour plus d'informations, voir IBM i Application Programming Reference (ILE/RPG).

Coding in PL/I (z/OS only)

Useful information when coding for IBM MQ in PL/I.

Structures

Structures are declared with the BASED attribute, and so do not occupy any storage unless the program declares one or more instances of a structure.

An instance of a structure can be declared using the `like` attribute, for example:

```
dcl my_mqmd    like MQMD; /* one instance */
dcl my_other_mqmd like MQMD; /* another one */
```

The structure fields are declared with the INITIAL attribute; when the `like` attribute is used to declare an instance of a structure, that instance inherits the initial values defined for that structure. You need to set only those fields where the value required is different from the initial value.

PL/I is not sensitive to case, and so the names of calls, structure fields, and constants can be coded in lowercase, uppercase, or mixed case.

Named constants

The named constants are declared as macro variables; as a result, named constants that are not referred to by the program do not occupy any storage in the compiled procedure.

However, the compiler option that causes the source to be processed by the macro preprocessor must be specified when the program is compiled.

All the macro variables are character variables, even the ones that represent numeric values. Although this might seem counter intuitive, it does not result in any data-type conflict after the macro variables have been substituted by the macro processor, for example:

```
%dcl MQMD_STRUC_ID char;
%MQMD_STRUC_ID = '''MD ''';

%dcl MQMD_VERSION_1 char;
%MQMD_VERSION_1 = '1';
```

Utilisation des exemples de programmes procéduraux IBM MQ

Ces exemples de programmes sont écrits dans des langages procéduraux et présentent des utilisations typiques de l'interface MQI (Message Queue Interface). Programmes IBM MQ sur différentes plateformes.

Pourquoi et quand exécuter cette tâche

Il existe deux ensembles d'échantillons:

- Multi Exemples de programmes pour Multiplatforms.
- z/OS Exemples de programmes pour z/OS.

Procédure

- Utilisez les liens suivants pour en savoir plus sur les exemples de programmes:
  - Multi «Utilisation des exemples de programme sur Multiplatforms», à la page 1086
  - z/OS «Using the sample programs for z/OS», à la page 1195

Concepts associés

«Concepts de développement d'applications», à la page 7
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM MQ . Avant de commencer à concevoir et à écrire vos applications IBM MQ , familiarisez-vous avec les concepts de base de IBM MQ .

«Développement d'applications pour IBM MQ», à la page 5
Vous pouvez développer des applications pour envoyer et recevoir des messages et pour gérer vos gestionnaires de files d'attente et les ressources associées. IBM MQ prend en charge les applications écrites dans de nombreux langages et infrastructures différents.

«Remarques sur la conception des applications IBM MQ», à la page 52
Une fois que vous avez décidé comment vos applications peuvent tirer parti des plateformes et des environnements dont vous disposez, vous devez décider comment utiliser les fonctions offertes par IBM MQ.

«Ecriture d'une application de procédure pour la mise en file d'attente», à la page 738
La présente rubrique explique comment écrire des applications de mise en file d'attente et décrit la connexion à un gestionnaire de files d'attente et la déconnexion de celui-ci, la publication/l'abonnement, ainsi que l'ouverture et la fermeture des objets.

«Ecriture d'applications de procédure client», à la page 937
Ce que vous devez savoir pour écrire des applications client sur IBM MQ à l'aide d'un langage procédural.

«Ecriture d'applications de publication / abonnement», à la page 829
Commencez à écrire des applications IBM MQ de publication / abonnement.

«Création d'une application procédurale», à la page 1027
Vous pouvez écrire une application IBM MQ dans l'un des langages procéduraux et exécuter l'application sur plusieurs plateformes différentes.

«Traitement des erreurs de programme de procédure», à la page 1065
Ces informations expliquent les erreurs associées aux appels MQI de vos applications lors de l'exécution d'un appel ou de la distribution de son message à sa destination finale.

Multi Utilisation des exemples de programme sur Multiplatforms

Ces exemples de programmes procéduraux sont fournis avec le produit. Les exemples sont écrits en langage C et COBOL et illustrent les utilisations typiques de l'interface MQI (Message Queue Interface).

Pourquoi et quand exécuter cette tâche

Les exemples ne sont pas destinés à illustrer des techniques de programmation générales. Par conséquent, certaines vérifications d'erreur que vous pouvez souhaiter inclure dans un programme de production sont omises.

Le code source de tous les exemples est fourni avec le produit ; cette source inclut des commentaires qui expliquent les techniques de mise en file d'attente des messages présentées dans les programmes.

IBM i Pour la programmation RPG, voir IBM i Application Programming Reference (ILE/RPG).

Les noms des exemples commencent par le préfixe `amq`. Le quatrième caractère indique le langage de programmation et le compilateur si nécessaire:

- s: langage C
- 0: langage COBOL sur les compilateurs IBM et Micro Focus
- i: langage COBOL sur les compilateurs IBM uniquement
- m: langage COBOL sur les compilateurs Micro Focus uniquement

Le huitième caractère de l'exécutable indique si l'exemple s'exécute en mode de liaison locale ou en mode client. S'il n'y a pas de huitième caractère, l'exemple s'exécute en mode de liaisons locales. Si le huitième caractère est'c', l'exemple s'exécute en mode client.

Avant de pouvoir exécuter les exemples d'application, vous devez d'abord créer et configurer un gestionnaire de files d'attente. Pour configurer le gestionnaire de files d'attente afin qu'il accepte les connexions client, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.

Procédure

- Utilisez les liens suivants pour en savoir plus sur les exemples de programmes:
  - «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088
  - «Préparation et exécution des exemples de programmes», à la page 1098
  - «Exemple de programme d'exit API», à la page 1106
  - «Exemple de programme de consommation asynchrone», à la page 1106
  - «Exemple de programme d'insertion asynchrone», à la page 1108
  - «Exemples de programmes de navigation», à la page 1108
  - «Exemple de programme de navigateur», à la page 1110
  - «Exemple de transaction CICS», à la page 1111
  - «Exemple de programme Connect», à la page 1111
  - «Exemple de programme de conversion de données», à la page 1113
  - «Exemples de coordination de base de données», à la page 1113
  - «Exemple de gestionnaire de files d'attente de rebut», à la page 1120
  - «Exemple de programme Liste de distribution», à la page 1120
  - «Exemples de programmes Echo», à la page 1121
  - «Exemples de programmes Get», à la page 1122
  - «Exemples de programmes à haute disponibilité», à la page 1124
  - «Exemples de programmes Inquire», à la page 1128
  - «Exemple de programme d'interrogation des propriétés d'un descripteur de message», à la page 1129
  - «Exemples de programmes de publication / abonnement», à la page 1129
  - «Exemple de programme d'exit de publication», à la page 1134
  - «Exemples de programmes Put», à la page 1135
  - «Exemples de programmes de message de référence», à la page 1137
  - «Exemples de programmes de demande», à la page 1145
  - «Exemples de programme Set», à la page 1151

- «Exemple de programme TLS», à la page 1152
- «Exemples de programmes de déclenchement», à la page 1155
- «Utilisation des exemples TUXEDO sur AIX, Linux, and Windows», à la page 1157
- «Utilisation de l'exit de sécurité SSPI sous Windows», à la page 1168
- «Exécution des exemples à l'aide de files d'attente distantes», à la page 1169
- «Exemple de programme de surveillance de file d'attente de cluster (AMQSCLM)», à la page 1169
- «Exemple de programme de recherche de noeud final de connexion (CEPL)», à la page 1179

Concepts associés

«Exemples de programmes C++», à la page 542
Quatre exemples de programmes sont fournis pour illustrer l'obtention et l'insertion de messages.

Tâches associées

«Using the sample programs for z/OS», à la page 1195
The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

Multi *Fonctions démontrées dans les exemples de programmes sur Multiplatforms*

Collection de tableaux présentant les techniques mises en évidence par les exemples de programme IBM MQ .

Tous les exemples ouvrent et ferment des files d'attente à l'aide des appels MQOPEN et MQCLOSE, de sorte que ces techniques ne sont pas répertoriées séparément dans les tableaux. Voir l'en-tête qui inclut la plateforme qui vous intéresse.

z/OS Pour la plateforme z/OS , voir «Using the sample programs for z/OS», à la page 1195.

Linux AIX *Exemples pour les systèmes AIX and Linux*

Les techniques mises en évidence par les exemples de programme pour IBM MQ for AIX or Linux.

Voir «Préparation et exécution d'exemples de programmes sous AIX and Linux», à la page 1102 pour savoir où sont stockés les exemples de programme pour IBM MQ for AIX or Linux .

Tableau 156, à la page 1088 Le tableau répertorie les fichiers source C et COBOL qui sont fournis et indique si un exécutable serveur ou client est inclus.

Tableau 156. Exemples de programmes illustrant l'utilisation de l'interface MQI (C et COBOL) sous AIX and Linux.

Tableau à quatre colonnes. Les premières colonnes répertorient les techniques mises en évidence par les échantillons. La deuxième colonne répertorie les exemples C et la troisième colonne répertorie les exemples COBOL qui illustrent chacune des techniques répertoriées dans la première colonne. La quatrième colonne indique si un exécutable C serveur est inclus ou non et la cinquième colonne indique si un exécutable C client est inclus ou non.

Technique	**C (source) («1», à la page 1091)**	**COBOL (source) («2», à la page 1091)**	**Serveur (exécutable C)**	**Client (exécutable C)**
Utilisation de l'interface de publication / abonnement	amqspuba amqssuba amqssbxa	aucun échantillon	amqspub amqssub amqssbx	aucun échantillon
Insertion de messages à l'aide de l'appel MQPUT	amqsput0	amq0put0	amqsput	amqsputc

Tableau 156. Exemples de programmes illustrant l'utilisation de l'interface MQI (C et COBOL) sous AIX and Linux.

Tableau à quatre colonnes. Les premières colonnes répertorient les techniques mises en évidence par les échantillons. La deuxième colonne répertorie les exemples C et la troisième colonne répertorie les exemples COBOL qui illustrent chacune des techniques répertoriées dans la première colonne. La quatrième colonne indique si un exécutable C serveur est inclus ou non et la cinquième colonne indique si un exécutable C client est inclus ou non.

(suite)

Technique	**C (source) («1», à la page 1091)**	**COBOL (source) («2», à la page 1091)**	**Serveur (exécutable C)**	**Client (exécutable C)**
Insertion d'un message unique à l'aide de l'appel MQPUT1	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Insertion de messages dans une liste de distribution («3», à la page 1091)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Réponse à un message de demande	amqsinqa	amqminqx amqiinqx	amqsinq	aucun échantillon
Obtention de messages à l'aide de la fonction de navigation (pas d'attente)	amqsgbr0	amq0gbr0	amqsgbr	aucun échantillon
Obtention de messages (attente avec une limite de temps)	amqsget0	amq0get0	amqsget	amqsgetc
Obtention de messages (attente illimitée)	amqstrg0	aucun échantillon	amqstrg	amqstrgc
Obtention de messages (avec conversion de données)	amqsecha	aucun échantillon	amqsech	aucun échantillon
Insertion de messages de référence dans une file d'attente («3», à la page 1091)	amqsprma	aucun échantillon	amqsprm	amqsprmc
Obtention des messages de référence à partir d'une file d'attente («3», à la page 1091)	amqsgrma	aucun échantillon	amqsgrm	amqsgrmc
Exit de canal de message de référence («3», à la page 1091)	amqsqrma amqsxrma	aucun échantillon	amqsxrm	aucun échantillon
Navigation dans les 20 premiers caractères d'un message	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Exploration des messages complets	amqsbcg0	aucun échantillon	amqsbcg	amqsbcgc
Utilisation d'une file d'entrée partagée	amqsinqa	amqminqx amqiinqx	amqsinq	amqsinqc
Utilisation d'une file d'attente en entrée exclusive	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilisation de l'appel MQINQ	amqsinqa	amqminqx amqiinqx	amqsinq	aucun échantillon
Utilisation de l'appel MQSET	amqsseta	amqmsetx amqisetx	amqsset	amqssetc

Tableau 156. Exemples de programmes illustrant l'utilisation de l'interface MQI (C et COBOL) sous AIX and Linux.

Tableau à quatre colonnes. Les premières colonnes répertorient les techniques mises en évidence par les échantillons. La deuxième colonne répertorie les exemples C et la troisième colonne répertorie les exemples COBOL qui illustrent chacune des techniques répertoriées dans la première colonne. La quatrième colonne indique si un exécutable C serveur est inclus ou non et la cinquième colonne indique si un exécutable C client est inclus ou non.

(suite)

Technique	**C (source) («1», à la page 1091)**	**COBOL (source) («2», à la page 1091)**	**Serveur (exécutable C)**	**Client (exécutable C)**
Utilisation d'une file d'attente de réponse	amqsreq0	amq0req0	amqsreq	amqsreqc
Demande d'exceptions de message	amqsreq0	amq0req0	amqsreq	aucun échantillon
Acceptation d'un message tronqué	amqsgbr0	amq0gbr0	amqsgbr	aucun échantillon
Utilisation d'un nom de file d'attente résolue	amqsgbr0	amq0gbr0	amqsgbr	aucun échantillon
Déclenchement d'un processus	amqstrg0	aucun échantillon	amqstrg	amqstrgc
Utilisation de la conversion de données	(«4», à la page 1091)	aucun échantillon	aucun échantillon	aucun échantillon
IBM MQ (coordination des gestionnaires de base de données compatibles XA) accédant à une base de données unique à l'aide de SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	aucun échantillon	aucun échantillon
IBM MQ (coordination des gestionnaires de base de données compatibles XA) accédant à deux bases de données à l'aide de SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	aucun échantillon	aucun échantillon
Transaction CICS («5», à la page 1091)	amqscic0.ccs	aucun échantillon	amqscic0	aucun échantillon
Transaction Encina («3», à la page 1091)	amqsxae0	aucun échantillon	amqsxae0	aucun échantillon
Transaction TUXEDO pour insérer des messages «6», à la page 1091)	amqstxpx	aucun échantillon	aucun échantillon	aucun échantillon
Transaction TUXEDO pour obtenir des messages («6», à la page 1091)	amqstxgx	aucun échantillon	aucun échantillon	aucun échantillon
Serveur pour TUXEDO («6», à la page 1091)	amqstxsx	aucun échantillon	aucun échantillon	aucun échantillon
gestionnaire de files d'attente de rebuts	Répertoire `./tools/c/Samples/dlq` («7», à la page 1092)	aucun échantillon	amqsdlq	aucun échantillon

Tableau 156. Exemples de programmes illustrant l'utilisation de l'interface MQI (C et COBOL) sous AIX and Linux.

Tableau à quatre colonnes. Les premières colonnes répertorient les techniques mises en évidence par les échantillons. La deuxième colonne répertorie les exemples C et la troisième colonne répertorie les exemples COBOL qui illustrent chacune des techniques répertoriées dans la première colonne. La quatrième colonne indique si un exécutable C serveur est inclus ou non et la cinquième colonne indique si un exécutable C client est inclus ou non.

(suite)

Technique	**C (source) («1», à la page 1091)**	**COBOL (source) («2», à la page 1091)**	**Serveur (exécutable C)**	**Client (exécutable C)**
A partir d'un client MQI, insertion d'un message	aucun échantillon	aucun échantillon	aucun échantillon	amqsputc
A partir d'un client MQI, obtention d'un message	aucun échantillon	aucun échantillon	aucun échantillon	amqsgetc
Connexion au gestionnaire de files d'attente à l'aide de MQCONNX	amqscnxc	aucun échantillon	aucun échantillon	amqscnxc
Utilisation des exits API	amqsaxe0	aucun échantillon	amqsaxe	aucun échantillon
Exit d'équilibrage de charge de cluster	amqswlm0	aucun échantillon	amqswlm	aucun échantillon
Insertion de messages de manière asynchrone et obtention du statut à l'aide de l'appel MQSTAT	amqsapt0	aucun échantillon	amqsapt	amqsaptc
Clients reconnectables	amqsphac amqsghac amqsmhac	aucun échantillon	non applicable	amqsphac amqsghac amqsmhac
Utilisation de consommateurs de messages pour consommer de manière asynchrone des messages provenant de plusieurs files d'attente	amqscbf0	aucun échantillon	amqscbf	amqscbfc
Spécification des informations de connexion TLS sur MQCONNX	amqssslc	aucun échantillon	non applicable	amqssslc

Remarques :

1. La version exécutable des exemples IBM MQ MQI client partage la même source que les exemples qui s'exécutent dans un environnement de serveur.
2. Compilez les programmes commençant par'amqm'avec le compilateur Micro Focus COBOL, ceux commençant par'amqi'avec le compilateur IBM COBOL et ceux commençant par'amq0'.
3. AIX Pris en charge sous IBM MQ for AIX uniquement.
4. AIX Sous IBM MQ for AIX , ce programme est appelé `amqsvfc0.c`
5. AIX CICS est pris en charge par IBM MQ for AIX uniquement.
6. Linux TUXEDO n'est pas pris en charge par IBM MQ for Linux sur System p.

7. La source du gestionnaire de files d'attente de rebut se compose de plusieurs fichiers et est fournie dans un répertoire distinct.

Pour plus d'informations sur la prise en charge des systèmes AIX and Linux , voir Configuration système requise pour IBM MQ.

Windows *Exemples pour IBM MQ for Windows*

Les techniques mises en évidence par les exemples de programme pour IBM MQ for Windows.

Le Tableau 157, à la page 1092 répertorie les fichiers source C et COBOL qui sont fournis et indique si un exécutable serveur ou client est inclus.

Tableau 157. Exemples de programmes IBM MQ for Windows illustrant l'utilisation de l'interface MQI (C et COBOL)

Technique	**C (source)**	**COBOL (source)**	**Serveur (exécutable C)**	**Client (exécutable C)**
Utilisation de l'interface de publication / abonnement	amqspuba amqssuba amqssbxa	aucun échantillon	amqspub amqssub amqssbx	aucun échantillon
Insertion de messages à l'aide de l'appel MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Insertion d'un message unique à l'aide de l'appel MQPUT1	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc
Insertion de messages dans une liste de distribution	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Réponse à un message de demande	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Obtention de messages (pas d'attente)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Obtention de messages (attente avec une limite de temps)	amqsget0	amq0get0	amqsget	amqsgetc
Obtention de messages (attente illimitée)	amqstrg0	aucun échantillon	amqstrg	amqstrgc
Obtention de messages (avec conversion de données)	amqsecha	aucun échantillon	amqsech	amqsechc
Insertion de messages de référence dans une file d'attente	amqsprma	aucun échantillon	amqsprm	amqsprmc
Obtention de messages de référence à partir d'une file d'attente	amqsgrma	aucun échantillon	amqsgrm	amqsgrmc
Exit de canal de message de référence	amqsqrma amqsxrma	aucun échantillon	amqsxrm	aucun échantillon
Navigation dans les 20 premiers caractères d'un message	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Exploration des messages complets	amqsbcg0	aucun échantillon	amqsbcg	amqsbcgc
Utilisation d'une file d'entrée partagée	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc

Tableau 157. Exemples de programmes IBM MQ for Windows illustrant l'utilisation de l'interface MQI (C et COBOL) (suite)

Technique	**C (source)**	**COBOL (source)**	**Serveur (exécutable C)**	**Client (exécutable C)**
Utilisation d'une file d'attente en entrée exclusive	amqstrg0	amq0req0	amqstrg	amqstrgc
Utilisation de l'appel MQINQ	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Utilisation de l'appel MQSET	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
Utilisation de l'appel MQINQMP	amqsiqma	aucun échantillon	aucun échantillon	aucun échantillon
Utilisation d'une file d'attente de réponse	amqsreq0	amq0req0	amqsreq	amqsreqc
Demande d'exceptions de message	amqsreq0	amq0req0	amqsreq	amqsreqc
Acceptation d'un message tronqué	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Utilisation d'un nom de file d'attente résolue	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Déclenchement d'un processus	amqstrg0	aucun échantillon	amqstrg	amqstrgc
Utilisation de la conversion de données	amqsvfc0	aucun échantillon	aucun échantillon	aucun échantillon
IBM MQ (coordination des gestionnaires de base de données compatibles XA) accédant à une base de données unique à l'aide de SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	aucun échantillon	aucun échantillon
IBM MQ (coordination des gestionnaires de base de données compatibles XA) accédant à deux bases de données à l'aide de SQL	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	aucun échantillon	aucun échantillon
Transaction TUXEDO permettant d'insérer des messages	amqstxpx	aucun échantillon	aucun échantillon	aucun échantillon
Transaction TUXEDO pour obtenir des messages	amqstxgx	aucun échantillon	aucun échantillon	aucun échantillon
Serveur pour TUXEDO	amqstxsx	aucun échantillon	aucun échantillon	aucun échantillon
gestionnaire de files d'attente de rebuts	Répertoire `./tools/c/Samples/dlq` («1», à la page 1094)	aucun échantillon	amqsdlq	aucun échantillon
A partir d'un IBM MQ MQI client, insertion d'un message	aucun échantillon	aucun échantillon	aucun échantillon	amqsputc

Tableau 157. Exemples de programmes IBM MQ for Windows illustrant l'utilisation de l'interface MQI (C et COBOL) (suite)

Technique	C (source)	COBOL (source)	Serveur (exécutable C)	Client (exécutable C)
A partir d'un IBM MQ MQI client, obtention d'un message	aucun échantillon	aucun échantillon	aucun échantillon	amqsgetc
Connexion au gestionnaire de files d'attente à l'aide de MQCONNX	amqscnxc	aucun échantillon	aucun échantillon	amqscnxc
Utilisation des exits API	amqsaxe0	aucun échantillon	amqsaxe	aucun échantillon
Equilibrage de charge de cluster	amqswlm0	aucun échantillon	amqswlm	aucun échantillon
Routines de sécurité SSPI	amqsspin	aucun échantillon	amqrspin.dll	amqrspin.dll
Insertion de messages de manière asynchrone et obtention du statut à l'aide de l'appel MQSTAT	amqsapt0	aucun échantillon	amqsapt	amqsaptc
Clients reconnectables	amqsphac amqsghac amqsmhac	aucun échantillon	Non applicable	amqsphac amqsghac amqsmhac
Utilisation de consommateurs de messages pour consommer de manière asynchrone des messages provenant de plusieurs files d'attente	amqscbf0	aucun échantillon	amqscbf	amqscbfc
Spécification des informations de connexion TLS sur MQCONNX	amqssslc	aucun échantillon	non applicable	amqssslc

Remarques :

1. La source du gestionnaire de files d'attente de rebut se compose de plusieurs fichiers et est fournie dans un répertoire distinct.

Windows *Exemples Visual Basic pour IBM MQ for Windows*

Techniques mises en évidence par les exemples de programmes pour IBM MQ sur les systèmes Windows .

Le Tableau 158, à la page 1094 présente les techniques mises en évidence par les exemples de programme IBM MQ for Windows .

Un projet peut contenir plusieurs fichiers. Lorsque vous ouvrez un projet dans Visual Basic, les autres fichiers sont chargés automatiquement. Aucun programme exécutable n'est fourni.

Tous les exemples de projet, à l'exception de mqtrivc.vbp, sont configurés pour fonctionner avec le serveur IBM MQ . Pour savoir comment modifier les exemples de projets afin de les utiliser avec les clients IBM MQ , voir «Préparation des programmes Visual Basic dans Windows», à la page 1047.

Tableau 158. Exemples de programmes IBM MQ for Windows illustrant l'utilisation de MQI (Visual Basic)

Technique	Nom du fichier de projet
Insertion de messages à l'aide de l'appel MQPUT	amqsputb.vbp

Tableau 158. Exemples de programmes IBM MQ for Windows illustrant l'utilisation de MQI (Visual Basic) (suite)	
Technique	**Nom du fichier de projet**
Obtention de messages à l'aide de l'appel MQGET	amqsgetb.vbp
Exploration d'une file d'attente à l'aide de l'appel MQGET	amqsbcgb.vbp
Exemple MQGET et MQPUT simple (client)	mqtrivc.vbp
Exemple MQGET et MQPUT simple (serveur)	mqtrivs.vbp
Insertion et obtention de chaînes et de structures définies par l'utilisateur à l'aide de MQPUT et de MQGET	strings.vbp
Utilisation de structures PCF pour démarrer et arrêter un canal	pcfsamp.vbp
Création d'une file d'attente à l'aide de MQAI	amqsaicq.vbp
Affichage de la liste des files d'attente d'un gestionnaire de files d'attente à l'aide de MQAI	amqsailq.vbp
Surveillance des événements à l'aide de MQAI	amqsaiem.vbp

IBM i *Exemples pour IBM i*

Techniques mises en évidence par les exemples de programmes pour IBM MQ sur les systèmes IBM i .

Le Tableau 159, à la page 1095 présente les techniques mises en évidence par les exemples de programme IBM MQ for IBM i . Certaines techniques sont utilisées dans plus d'un programme exemple, mais un seul programme est répertorié dans le tableau.

Tableau 159. Exemples de programmes illustrant l'utilisation de MQI (C et COBOL) sous IBM i				
Technique	**C (source) («1», à la page 1097)**	**COBOL (source) («2», à la page 1097)**	**RPG (source) («3», à la page 1097)**	**Client (exécutable C) (4)**
Insertion de messages à l'aide de l'appel MQPUT	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	amqsputc
Insertion de messages à partir d'un fichier de données à l'aide de l'appel MQPUT	AMQSPUT4	aucun échantillon	aucun échantillon	aucun échantillon
Insertion d'un message unique à l'aide de l'appel MQPUT1	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Insertion de messages dans une liste de distribution	AMQSPTL4	aucun échantillon	aucun échantillon	AMQSPTLC
Réponse à un message de demande	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Obtention de messages (pas d'attente)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Obtention de messages (attente avec une limite de temps)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
Obtention de messages (attente illimitée)	AMQSTRG4	aucun échantillon	AMQ3TRG4	Commande AMQSTRGC
Obtention de messages (avec conversion de données)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Insertion de messages de référence dans une file d'attente	AMQSPRM4	aucun échantillon	aucun échantillon	AMQSPRMC

Tableau 159. Exemples de programmes illustrant l'utilisation de MQI (C et COBOL) sous IBM i (suite)

Technique	**C (source) («1», à la page 1097)**	**COBOL (source) («2», à la page 1097)**	**RPG (source) («3», à la page 1097)**	**Client (exécutable C) (4)**
Obtention de messages de référence à partir d'une file d'attente	AMQSGRM4	aucun échantillon	aucun échantillon	AMQSGRMC
Exit de canal de message de référence	AMQSQRM4, AMQSXRM4	aucun échantillon	aucun échantillon	aucun échantillon
Exit de message	AMQSCMX4	aucun échantillon	aucun échantillon	aucun échantillon
Exploration des 49 premiers caractères d'un message	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Exploration des messages complets	AMQSBCG4	aucun échantillon	aucun échantillon	AMQSBCGC
Utilisation d'une file d'entrée partagée	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilisation d'une file d'attente en entrée exclusive	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Utilisation de l'appel MQINQ	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Utilisation de l'appel MQSET	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
Utilisation d'une file d'attente de réponse	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Demande d'exceptions de message	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Acceptation d'un message tronqué	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Utilisation d'un nom de file d'attente résolue	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Déclenchement d'un processus	AMQSTRG4	aucun échantillon	AMQ3TRG4	Commande AMQSTRGC
Serveur de déclenchement	AMQSERV4	aucun échantillon	AMQ3SRV4	aucun échantillon
Utilisation d'un serveur de déclenchement (y compris les transactions CICS)	AMQSERV4	aucun échantillon	AMQ3SRV4	aucun échantillon
Utilisation de la conversion de données	AMQSVFC4	aucun échantillon	aucun échantillon	aucun échantillon
Utilisation des exits API	AMQSAXE0	aucun échantillon	aucun échantillon	aucun échantillon
Equilibrage de charge de cluster	AMQSWLM0	aucun échantillon	aucun échantillon	aucun échantillon
Insertion de messages de manière asynchrone et obtention du statut à l'aide de l'appel MQSTAT	AMQSAPT0	aucun échantillon	aucun échantillon	AMQSAPTC
Utilisation de l'interface de publication / abonnement	AMQSPUBA, AMQSSUBA, AMQSSBXA	aucun échantillon	aucun échantillon	AMQSPUBC, AMQSSUBC, AMQSSBXC

Tableau 159. Exemples de programmes illustrant l'utilisation de MQI (C et COBOL) sous IBM i (suite)				
Technique	**C (source) («1», à la page 1097)**	**COBOL (source) («2», à la page 1097)**	**RPG (source) («3», à la page 1097)**	**Client (exécutable C) (4)**
Clients reconnectables (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	aucun échantillon	aucun échantillon	aucun échantillon
Utilisation de consommateurs de messages pour consommer de manière asynchrone des messages provenant de plusieurs files d'attente (5)	AMQSCBFO	aucun échantillon	aucun échantillon	aucun échantillon
Spécification des informations de connexion TLS sur MQCONNX	AMQSSSLC	aucun échantillon	aucun échantillon	AMQSSSLC
Connexion au gestionnaire de files d'attente à l'aide de MQCONNX	AMQSCNXC	aucun échantillon	aucun échantillon	AMQSCNXC
Consulter les propriétés d'un descripteur de message, à l'aide de MQINQMP, à partir d'une file d'attente de messages	AMQISQMA	aucun échantillon	aucun échantillon	AMQISQMC
Définition des propriétés d'un descripteur de message à l'aide de MQSETMP et insertion dans une file d'attente de messages	AMQSSQMA	aucun échantillon	aucun échantillon	AMQSSQMC

Remarques :

1. La source des exemples C se trouve dans le fichier QMQMSAMP/QCSRC. Les fichiers d'inclusion existent en tant que membres dans le fichier QMQM/H.
2. La source des exemples COBOL se trouve dans les fichiers QMQMSAMP/QCBLLESRC. Les membres sont nommés AMQ0 xxx 4, où xxx indique l'exemple de fonction.
3. La source des exemples RPG se trouve dans QMQMSAMP/QRPGLESRC. Les membres sont nommés AMQ3 xxx 4, où xxx indique l'exemple de fonction. Des membres de copie existent dans QMQM/QRPGLESRC. Chaque nom de membre possède le suffixe G.
4. La version exécutable des exemples IBM MQ MQI client partage la même source que les exemples qui s'exécutent dans un environnement de serveur. La source des exemples dans l'environnement client est identique à celle du serveur. Les exemples IBM MQ MQI client sont liés à la bibliothèque client LIBMQIC et les exemples de serveur IBM MQ sont liés à la bibliothèque serveur LIBMQM.
5. Si l'exécutable client de l'exemple d'application du client reconnectable et de l'application client asynchrone doit être exécuté, il doit être compilé et lié à la bibliothèque à unités d'exécution LIBMQIC_R. Par conséquent, il doit être exécuté dans un environnement à unités d'exécution. Définissez la variable d'environnement QIBM_MULTI_THREADED sur'Y'et exécutez l'application à partir de qsh.

   Pour plus d'informations, voir Configuration de IBM MQ avec Java et JMS .

Pour plus d'informations, voir «Préparation et exécution d'exemples de programmes sous IBM i», à la page 1100.

En outre, l'exemple d'option IBM MQ for IBM i inclut un exemple de fichier de données que vous utilisez comme entrée pour les exemples de programmes, AMQSDATA et les exemples de programmes CL qui illustrent les tâches d'administration. Les exemples CL sont décrits dans Administration d' IBM i . Vous pouvez utiliser l'exemple de programme CL amqsamp4 pour créer des files d'attente à utiliser avec les exemples de programme décrits dans cette rubrique.

Multi ***Préparation et exécution des exemples de programmes***

Une fois la préparation initiale terminée, vous pouvez exécuter les exemples de programme.

Pourquoi et quand exécuter cette tâche

Avant d'exécuter les exemples de programme, vous devez d'abord créer un gestionnaire de files d'attente, ainsi que les files d'attente dont vous avez besoin. Vous devrez peut-être également effectuer une préparation supplémentaire, par exemple, si vous souhaitez exécuter des exemples COBOL. Après avoir effectué la préparation nécessaire, vous pouvez exécuter les exemples de programme.

Procédure

Pour plus d'informations sur la préparation et l'exécution des exemples de programmes, voir les rubriques suivantes:

- «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098
- «Préparation et exécution d'exemples de programmes sous IBM i», à la page 1100
- «Préparation et exécution d'exemples de programmes sous AIX and Linux», à la page 1102
- «Préparation et exécution d'exemples de programmes sous Windows», à la page 1104

Multi *Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms*

Avant de pouvoir exécuter les modèles d'application, vous devez d'abord créer un gestionnaire de files d'attente. Vous pouvez ensuite configurer le gestionnaire de files d'attente pour qu'il accepte en toute sécurité les demandes de connexion entrantes provenant des applications qui s'exécutent en mode client.

Avant de commencer

Vérifiez que le gestionnaire de files d'attente existe déjà et qu'il a été démarré. Déterminez si les enregistrements d'authentification de canal sont déjà activés en émettant la commande MQSC:

```
DISPLAY QMGR CHLAUTH
```

Important : Cette tâche s'attend à ce que les enregistrements d'authentification de canal soient activés. S'il s'agit d'un gestionnaire de files d'attente utilisé par d'autres utilisateurs et applications, la modification de ce paramètre affectera tous les autres utilisateurs et applications. Si votre gestionnaire de files d'attente n'utilise pas d'enregistrements d'authentification de canal, l'étape 4 peut être remplacée par une autre méthode d'authentification (par exemple, un exit de sécurité) qui définit MCAUSER sur *non-privileged-user-id* que vous obtiendrez à l'étape «1», à la page 1098.

Vous devez savoir quel nom de canal votre application s'attend à utiliser pour que l'application puisse être autorisée à utiliser le canal. Vous devez également savoir quels objets, par exemple des files d'attente ou des rubriques, votre application s'attend à utiliser pour que votre application puisse les utiliser.

Pourquoi et quand exécuter cette tâche

Cette tâche crée un ID utilisateur non privilégié à utiliser pour une application client qui se connecte au gestionnaire de files d'attente. L'accès est accordé à l'application client uniquement pour pouvoir utiliser le canal dont elle a besoin et la file d'attente dont elle a besoin à l'aide de cet ID utilisateur.

Procédure

1. Obtenez un ID utilisateur sur le système sur lequel votre gestionnaire de files d'attente est exécuté. Pour cette tâche, cet ID utilisateur ne doit pas être un administrateur privilégié. Cet ID utilisateur sera l'autorité sous laquelle la connexion client sera exécutée sur le gestionnaire de files d'attente.

2. Démarrez un programme d'écoute à l'aide des commandes suivantes, où:

   qmgr-name est le nom de votre gestionnaire de files d'attente
   nnnn est le numéro de port que vous avez choisi

   a) ALW
   Pour les systèmes AIX, Linux, and Windows :

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

   b) IBM i
   Pour IBM i :

   ```
   STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
   ```

3. Si votre application utilise SYSTEM.DEF.SVRCONN alors ce canal est déjà défini. Si votre application utilise un autre canal, créez-la à l'aide de la commande MQSC suivante:

   ```
   DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   où *channel-name* est le nom de votre canal.
4. Créez une règle d'authentification de canal autorisant uniquement l'adresse IP de votre système client à utiliser le canal en exécutant la commande MQSC suivante:

   ```
   SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
   MCAUSER(' non-privileged-user-id ')
   ```

   Où

   nom_canal est le nom du canal.
   adresse_IP_machine_client est l'adresse IP de votre système client. Si votre exemple d'application client s'exécute sur la même machine que le gestionnaire de files d'attente, utilisez l'adresse IP'127.0.0.1'si votre application va se connecter à l'aide de'localhost'. Si plusieurs machines client différentes vont se connecter, vous pouvez utiliser un modèle ou une plage au lieu d'une adresse IP unique. Pour plus d'informations, voir Adresses IP génériques .
   non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 1098
5. Si votre application utilise SYSTEM.DEFAULT.LOCAL.QUEUE alors cette file d'attente est déjà définie. Si votre application utilise une autre file d'attente, créez-la à l'aide de la commande MQSC suivante:

   ```
   DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
   ```

   où *queue-name* est le nom de votre file d'attente.
6. Accordez l'accès pour vous connecter au gestionnaire de files d'attente et l'interroger en exécutant la commande MQSC suivante:

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

   où *non-privileged-user-id* est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 1098 .
7. Si votre application est une application point à point, c'est-à-dire qu'elle utilise des files d'attente, accordez l'accès pour autoriser l'interrogation et l'insertion et l'obtention de messages à l'aide de votre file d'attente par l'ID utilisateur à utiliser, en émettant les commandes MQSC suivantes:

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

Où

queue-name est le nom de votre file d'attente
non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 1098

8. Si votre application est une application de publication / abonnement, c'est-à-dire qu'elle utilise des rubriques, accordez l'accès pour autoriser la publication et l'abonnement à l'aide de votre rubrique par l'ID utilisateur à utiliser, en émettant les commandes MQSC:

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

Où

non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 1098
Cela permet à *non-privileged-user-id* d'accéder à n'importe quelle rubrique de l'arborescence de rubriques. Vous pouvez également définir un objet de rubrique à l'aide de **DEFINE TOPIC** et accorder des accès uniquement à la partie de l'arborescence de rubriques référencée par cet objet de rubrique. Pour plus d'informations, voir Contrôle de l'accès des utilisateurs aux rubriques .

Que faire ensuite

Votre application client peut désormais se connecter au gestionnaire de files d'attente et insérer ou extraire des messages à l'aide de la file d'attente.

Concepts associés

ALW Octroi de l'accès à un objet IBM MQ sur AIX, Linux, and Windows

Référence associée

SET CHLAUTH
De la définition d'un canal
DEFINE QLOCAL
SET AUTHREC
IBM i Droits IBM MQ sur IBM i

IBM i *Préparation et exécution d'exemples de programmes sous IBM i*

Avant d'exécuter les exemples de programme sous IBM i, vous devez d'abord créer un gestionnaire de files d'attente, ainsi que les files d'attente dont vous avez besoin. Si vous souhaitez exécuter des exemples COBOL, vous devrez peut-être effectuer une préparation supplémentaire.

Pourquoi et quand exécuter cette tâche

La source des exemples de programme IBM MQ for IBM i est fournie dans la bibliothèque QMQMSAMP en tant que membres de QCSRC, QCLSRC, QCBLLESRC et QRPGLESRC.

Vous pouvez utiliser vos propres files d'attente lorsque vous exécutez les exemples ou vous pouvez exécuter l'exemple de programme AMQSAMP4 pour créer des exemples de files d'attente. La source du programme AMQSAMP4 est incluse dans le fichier QCLSRC de la bibliothèque QMQMSAMP. Vous pouvez le compiler à l'aide de la commande CRTCLPGM.

Pour exécuter les exemples, utilisez les versions exécutables C, qui sont fournies dans la bibliothèque QMQM, ou compilez-les de la même manière que pour toute autre application IBM MQ .

V 9.4.0 V 9.4.0 Les exemples de programmes suivants disposent de fonctions d'authentification:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`

- amqsmhac.c
- amqsphac.c
- amqspuba.c
- amqsput0.c
- amqsssl c.c
- amqssuba.c

L'authentification est activée pour les versions exécutables de ces exemples. Toutefois, la compilation des versions source avec l'authentification activée requiert la définition de l'indicateur de compilation **SAMPLE_AUTH_ENABLED** et la compilation du fichier source amqsauth.c avec l'exemple souhaité. Exemple :

- Création du programme amqsssl c sans que l'authentification soit activée:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC) MODULE(MYLIB/AMQSSSLC) BNDSRVPGM(QMQM/LIBMQIC)
```

- Création du amqsssl c avec l'authentification activée:

```
CRTCMOD MODULE(MYLIB/AMQSSSLC) DEFINE('SAMPLE_AUTH_ENABLED') SRCFILE(QMQMSAMP/QCSRC)
CRTCMOD MODULE(MYLIB/AMQSAUTH) SRCFILE(QMQMSAMP/QCSRC)
CRTPGM PGM(MYLIB/AMQSSSLC_AUTH) MODULE(MYLIB/AMQSSSLC MYLIB/AMQSAUTH) BNDSRVPGM(QMQM/LIBMQIC)
```

Procédure

1. Créez un gestionnaire de files d'attente et configurez les définitions par défaut.

   Vous devez effectuer cette opération avant de pouvoir exécuter l'un des exemples de programme. Pour plus d'informations sur la création d'un gestionnaire de files d'attente, voir Administration d' IBM MQ. Pour plus d'informations sur la configuration d'un gestionnaire de files d'attente pour qu'il accepte en toute sécurité les demandes de connexion entrantes provenant d'applications qui s'exécutent en mode client, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.
2. Pour appeler l'un des exemples de programme à l'aide des données du membre PUT dans le fichier AMQSDATA de la bibliothèque QMQMSAMP, utilisez une commande similaire à la suivante:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

   Remarque : Pour qu'un module compilé puisse utiliser le système de fichiers IFS, indiquez l'option SYSIFCOPT (*IFSIO) sur CRTCMOD, puis le nom de fichier, transmis en tant que paramètre, doit être indiqué au format suivant:

   ```
   home/me/myfile
   ```
3. Si vous souhaitez utiliser les versions COBOL des exemples Inquire, Set et Echo, modifiez les définitions de processus avant d'exécuter ces exemples.

   Pour les exemples Inquire, Set et Echo, les exemples de définition déclenchent les versions C de ces exemples. Si vous souhaitez utiliser les versions COBOL, vous devez modifier les définitions de processus:

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

   Sous IBM i, vous pouvez utiliser la commande **CHGMQMPRC** (pour plus de détails, voir Change MQ Process (CHGMQMPRC)) ou éditer et exécuter la commande **AMQSAMP4** avec la définition alternative.
4. Exécutez les exemples de programme.

Pour plus d'informations sur les paramètres attendus par chacun des échantillons, voir les descriptions des échantillons individuels.

Remarque : Pour les exemples de programmes COBOL, lorsque vous transmettez des noms de file d'attente en tant que paramètres, vous devez fournir 48 caractères, en les complétant si nécessaire par des caractères blancs. Tout caractère autre que 48 provoque l'échec du programme avec le code raison 2085.

Référence associée

«Exemples pour IBM i», à la page 1095
Techniques mises en évidence par les exemples de programmes pour IBM MQ sur les systèmes IBM i .

Linux AIX *Préparation et exécution d'exemples de programmes sous AIX and Linux*

Avant d'exécuter les exemples de programme sous AIX and Linux, vous devez d'abord créer un gestionnaire de files d'attente, ainsi que les files d'attente dont vous avez besoin. Si vous souhaitez exécuter des exemples COBOL, vous devrez peut-être effectuer une préparation supplémentaire.

Pourquoi et quand exécuter cette tâche

Les exemples de fichier IBM MQ sur les systèmes AIX and Linux se trouvent dans les répertoires répertoriés dans le Tableau 160, à la page 1102 si les valeurs par défaut ont été utilisées lors de l'installation.

Tableau 160. Où trouver les exemples pour IBM MQ sur les systèmes AIX and Linux

Contenu	**Répertoire**
fichiers source	*MQ_INSTALLATION_PATH*/samp
fichiers source du gestionnaire de files d'attente de rebut	*MQ_INSTALLATION_PATH*/samp/dlq
fichiers exécutables	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Les exemples ont besoin d'un ensemble de files d'attente à utiliser. Vous pouvez utiliser vos propres files d'attente ou exécuter l'exemple de fichier MQSC amqscos0.tst pour créer un ensemble. Pour exécuter les exemples, utilisez les versions exécutables fournies ou compilez les versions source comme vous le feriez pour d'autres applications, à l'aide d'un compilateur ANSI.

V 9.4.0 V 9.4.0 Les exemples de programmes suivants disposent de fonctions d'authentification:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsсslc.c`
- `amqssuba.c`

L'authentification est activée pour les versions exécutables de ces exemples. Toutefois, la compilation des versions source avec l'authentification activée requiert la définition de l'indicateur de compilation

SAMPLE_AUTH_ENABLED et la compilation du fichier source `amqsauth.c` avec l'exemple souhaité. Exemple :

- Compilation de `amqsput0.c` sans authentification activée:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -o /bin/amqsput
amqsput0.c
```

- Compilation de `amqsput0.c` avec l'authentification activée:

```
gcc -m64 -I /opt/mqm/inc -L /opt/mqm/lib64 -lmqic -fsanitize=address -D SAMPLE_AUTH_ENABLED
-o /bin/amqsputc_auth amqsauth.c amqsput0.c
```

Procédure

1. Créez un gestionnaire de files d'attente et configurez les définitions par défaut.

   Vous devez effectuer cette opération avant de pouvoir exécuter l'un des exemples de programme. Pour plus d'informations sur la création d'un gestionnaire de files d'attente, voir Administration d' IBM MQ. Pour plus d'informations sur la configuration d'un gestionnaire de files d'attente pour qu'il accepte en toute sécurité les demandes de connexion entrantes provenant d'applications qui s'exécutent en mode client, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.
2. Si vous n'utilisez pas vos propres files d'attente, exécutez l'exemple de fichier MQSC amqscos0.tst pour créer un ensemble de files d'attente.

   Pour ce faire sur les systèmes AIX and Linux , entrez:

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

   Vérifiez le fichier `sampobj.out` pour vous assurer qu'il n'y a pas d'erreurs.
3. Si vous souhaitez utiliser les versions COBOL des exemples Inquire, Set et Echo, modifiez les définitions de processus avant d'exécuter ces exemples.

   Pour les exemples Inquire, Set et Echo, les exemples de définition déclenchent les versions C de ces exemples. Si vous souhaitez utiliser les versions COBOL, vous devez modifier les définitions de processus:

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

   Sous AIX and Linux, modifiez le fichier `amqscos0.tst` et remplacez les noms des fichiers exécutables C par les noms des fichiers exécutables COBOL avant d'utiliser la commande **runmqsc** pour exécuter ces exemples.
4. Exécutez les exemples de programme.

   Pour exécuter un exemple, entrez son nom suivi de tous les paramètres, par exemple:

   ```
   amqsput myqueue qmanagername
   ```

   où *myqueue* est le nom de la file d'attente dans laquelle les messages seront insérés et *qmanagername* est le gestionnaire de files d'attente propriétaire de *myqueue*.

   Pour plus d'informations sur les paramètres attendus par chacun des échantillons, voir les descriptions des échantillons individuels.

   Remarque : Pour les exemples de programmes COBOL, lorsque vous transmettez des noms de file d'attente en tant que paramètres, vous devez fournir 48 caractères, en les complétant si nécessaire par des caractères blancs. Tout caractère autre que 48 provoque l'échec du programme avec le code raison 2085.

Référence associée

«Exemples pour les systèmes AIX and Linux», à la page 1088

Les techniques mises en évidence par les exemples de programme pour IBM MQ for AIX or Linux.

Windows *Préparation et exécution d'exemples de programmes sous Windows*

Avant d'exécuter les exemples de programme sous Windows, vous devez d'abord créer un gestionnaire de files d'attente, ainsi que les files d'attente dont vous avez besoin. Si vous souhaitez exécuter des exemples COBOL, vous devrez peut-être effectuer une préparation supplémentaire.

Pourquoi et quand exécuter cette tâche

Les exemples de fichier IBM MQ for Windows se trouvent dans les répertoires répertoriés dans le Tableau 161, à la page 1104, si les valeurs par défaut ont été utilisées lors de l'installation. Par défaut, l'unité d'installation est < c: >.

Tableau 161. Où trouver les exemples pour IBM MQ for Windows

Contenu	**Répertoire**
Code source C	*MQ_INSTALLATION_PATH*\Tools\C\Exemples
Code source pour l'exemple de gestionnaire de rebut	*MQ_INSTALLATION_PATH*\Tools\C\Samples\DLQ
Code source COBOL	*MQ_INSTALLATION_PATH*\Tools\Cobol \ Exemples
Fichiers exécutables C [1]	*MQ_INSTALLATION_PATH*\ Tools\C\Samples \ Bin (versions 32 bits) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (versions 64 bits)
Exemples de fichiers MQSC	*MQ_INSTALLATION_PATH*\Tools\MQSC\Exemples
Code source Visual Basic	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
Exemples .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Exemples

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Remarque : Des versions 64 bits de certains exemples de fichiers exécutables C sont disponibles.

Les exemples ont besoin d'un ensemble de files d'attente à utiliser. Vous pouvez utiliser vos propres files d'attente ou exécuter l'exemple de fichier MQSC amqscos0.tst pour créer un ensemble de files d'attente. Pour exécuter les exemples, utilisez les versions d'exécutable fournies ou compilez les versions source comme vous le feriez pour d'autres applications IBM MQ for Windows .

V 9.4.0 V 9.4.0 Les exemples de programmes suivants disposent de fonctions d'authentification:

- `amqsbcg0.c`
- `amqsfhac.c`
- `amqsget0.c`
- `amqsghac.c`
- `amqsmhac.c`
- `amqsphac.c`
- `amqspuba.c`
- `amqsput0.c`
- `amqsssl c.c`
- `amqssuba.c`

L'authentification est activée pour les versions exécutables de ces exemples. Toutefois, la compilation des versions source avec l'authentification activée requiert la définition de l'indicateur de compilation **SAMPLE_AUTH_ENABLED** et la compilation du fichier source `amqsauth.c` avec l'exemple souhaité. Exemple :

- Compilation de `amqsput0.c` sans authentification activée:

```
CL amqsput0.c /link mqic.lib /OUT:Bin\amqsputc.exe
```

- Compilation de `amqsput0.c` avec l'authentification activée:

```
CL /D SAMPLE_AUTH_ENABLED amqsauth.c amqsput0.c /link mqic.lib /OUT:Bin\amqsputc_auth.exe
```

Procédure

1. Créez un gestionnaire de files d'attente et configurez les définitions par défaut.

   Vous devez effectuer cette opération avant de pouvoir exécuter l'un des exemples de programme. Pour plus d'informations sur la création d'un gestionnaire de files d'attente, voir Administration d' IBM MQ. Pour plus d'informations sur la configuration d'un gestionnaire de files d'attente pour qu'il accepte en toute sécurité les demandes de connexion entrantes provenant d'applications qui s'exécutent en mode client, voir «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098.
2. Si vous n'utilisez pas vos propres files d'attente, exécutez l'exemple de fichier MQSC amqscos0.tst pour créer un ensemble de files d'attente.

   Pour ce faire sur les systèmes Windows , entrez:

   ```
   runmqsc QManagerName < amqscos0.tst > sampobj.out
   ```

   Vérifiez le fichier `sampobj.out` pour vous assurer qu'il n'y a pas d'erreurs. Ce fichier se trouve dans votre répertoire de travail.
3. Si vous souhaitez utiliser les versions COBOL des exemples Inquire, Set et Echo, modifiez les définitions de processus avant d'exécuter ces exemples.

   Pour les exemples Inquire, Set et Echo, les exemples de définition déclenchent les versions C de ces exemples. Si vous souhaitez utiliser les versions COBOL, vous devez modifier les définitions de processus:

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

   Sous Windows, modifiez le fichier `amqscos0.tst` et remplacez les noms des fichiers exécutables C par les noms des fichiers exécutables COBOL avant d'utiliser la commande **runmqsc** pour exécuter ces exemples.
4. Exécutez les exemples de programme.

   Pour exécuter un exemple, entrez son nom suivi de tous les paramètres, par exemple:

   ```
   amqsput myqueue qmanagername
   ```

   où *myqueue* est le nom de la file d'attente dans laquelle les messages seront insérés et *qmanagername* est le gestionnaire de files d'attente propriétaire de *myqueue*.

   Pour plus d'informations sur les paramètres attendus par chacun des échantillons, voir les descriptions des échantillons individuels.

   Remarque : Pour les exemples de programmes COBOL, lorsque vous transmettez des noms de file d'attente en tant que paramètres, vous devez fournir 48 caractères, en les complétant si nécessaire par des caractères blancs. Tout caractère autre que 48 provoque l'échec du programme avec le code raison 2085.

Référence associée

«Exemples pour IBM MQ for Windows», à la page 1092
Les techniques mises en évidence par les exemples de programme pour IBM MQ for Windows.

«Exemples Visual Basic pour IBM MQ for Windows», à la page 1094
Techniques mises en évidence par les exemples de programmes pour IBM MQ sur les systèmes Windows .

Exemple de programme d'exit API

L'exemple d'exit API génère une trace MQI dans un fichier spécifié par l'utilisateur avec un préfixe défini dans la variable d'environnement **MQAPI_TRACE_LOGFILE** .

Pour plus d'informations sur les exits API, voir «Ecriture et compilation des exits API sur Multiplatforms», à la page 980.

Source
amqsaxe0.c

Binaire
amqsaxe

Configuration pour l'exemple d'exit

1. Ajoutez les informations suivantes à la section ApiExitLocal du fichier qm.ini .

   Plateformes autres que Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH/samp/bin/amqsaxe
   Name=SampleApiExit
   ```

   où *MQ_INSTALLATION_PATH* représente le répertoire d'installation d'IBM MQ.

   Windows

   ```
   ApiExitLocal:
   Sequence=100
   Function=EntryPoint
   Module= MQ_INSTALLATION_PATH\Tools\c\Samples\bin\amqsaxe
   Name=SampleApiExit
   ```

   où *MQ_INSTALLATION_PATH* représente le répertoire d'installation d'IBM MQ.

2. Définition de la variable d'environnement **MQAPI_TRACE_LOGFILE**

   ```
   MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
   ```

3. Exécutez votre application.

   Les fichiers de sortie sont créés dans le répertoire /tmp avec des noms tels que: MqiTrace.*pid*.*tid*.log.

Exemple de programme de consommation asynchrone

L'exemple de programme amqscbf illustre l'utilisation de MQCB et de MQCTL pour consommer les messages de plusieurs files d'attente de manière asynchrone.

amqscbf est fourni en tant que code source C, et un client binaire et un exécutable serveur sur les plateformes AIX, Linux, and Windows .

Le programme est démarré à partir de la ligne de commande et prend les paramètres facultatifs suivants:

```
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
```

Indiquez plusieurs noms de file d'attente pour lire les messages de plusieurs files d'attente (un maximum de dix files d'attente est pris en charge par l'exemple).

Remarque : Reconnect type n'est valide que pour les programmes client.

Exemple

L'exemple illustre l'exécution d'amqscbf en tant que programme serveur lisant un message de QL1 , puis en cours d'arrêt.

Utilisez IBM MQ Explorer pour insérer un message de test dans QL1. Arrêtez le programme en appuyant sur Entrée.

```
C:\>amqscbf QL1
Sample AMQSCBF0 start

Press enter to end
Message Call (9 Bytes) :
Message 1

Sample AMQSCBF0 end
```

Qu'est-ce qu'amqscbf démontre

L'exemple montre comment lire les messages de plusieurs files d'attente dans l'ordre de leur arrivée. Cela nécessiterait beaucoup plus de code à l'aide de MQGET synchrone. Dans le cas d'une consommation asynchrone, aucune interrogation n'est requise et la gestion des unités d'exécution et du stockage est effectuée par IBM MQ. Un exemple de "monde réel" devrait traiter les erreurs ; dans l'exemple, les erreurs sont écrites sur la console.

L'exemple de code comporte les étapes suivantes:

1. Définissez la fonction de rappel de consommation de message unique,

   ```
   void MessageConsumer(MQHCONN      hConn,
                        MQMD    * pMsgDesc,
                        MQGMO   * pGetMsgOpts,
                        MQBYTE  * Buffer,
                        MQCBC   * pContext)
   { ... }
   ```

2. Connectez-vous au gestionnaire de files d'attente,

   ```
   MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
   ```

3. Ouvrez les files d'attente d'entrée et associez chacune d'elles à la fonction de rappel MessageConsumer .

   ```
   MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
   cbd.CallbackFunction = MessageConsumer;
   MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
   ```

   Il n'est pas nécessaire de définir cbd.CallbackFunction pour chaque file d'attente ; il s'agit d'une zone d'entrée uniquement. Mais vous pouvez associer une fonction de rappel différente à chaque file d'attente.

4. Démarrage de la consommation des messages,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Attendez que l'utilisateur ait appuyé sur Entrée, puis arrêtez la consommation des messages.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Enfin, déconnectez-vous du gestionnaire de files d'attente.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

Exemple de programme d'insertion asynchrone

Découvrez comment exécuter l'exemple amqsapt et la conception de l'exemple de programme Asynchronous Put.

L'exemple de programme d'insertion asynchrone insère des messages dans une file d'attente à l'aide de l'appel MQPUT asynchrone, puis extrait les informations de statut à l'aide de l'appel MQSTAT. Voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 pour le nom de ce programme sur différentes plateformes.

Exécution de l'exemple amqsapt

Ce programme prend jusqu'à 6 paramètres:

1. Nom de la file d'attente cible (obligatoire)
2. Nom du gestionnaire de files d'attente (facultatif)
3. Options d'ouverture (facultatif)
4. Options de fermeture (facultatif)
5. Nom du gestionnaire de files d'attente cible (facultatif)
6. Nom de la file d'attente dynamique (facultatif)

Si aucun gestionnaire de files d'attente n'est spécifié, amqsapt se connecte au gestionnaire de files d'attente par défaut.

Conception de l'exemple de programme Asynchronous Put

Le programme utilise l'appel MQOPEN avec les options de sortie fournies ou avec les options MQOO_OUTPUT et MQOO_FAIL_IF_QUIESCING pour ouvrir la file d'attente cible afin d'insérer des messages.

S'il ne parvient pas à ouvrir la file d'attente, le programme génère un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN. Pour simplifier le programme, sur cet appel MQI et sur les appels MQI suivants, le programme utilise des valeurs par défaut pour de nombreuses options.

Pour chaque ligne d'entrée, le programme lit le texte dans une mémoire tampon et utilise l'appel MQPUT avec MQPMO_ASYNC_RESPONSE pour créer un message de datagramme contenant le texte de cette ligne et l'insérer de manière asynchrone dans la file d'attente cible. Le programme se poursuit jusqu'à ce qu'il atteigne la fin de l'entrée ou que l'appel MQPUT échoue. Si le programme atteint la fin de l'entrée, il ferme la file d'attente à l'aide de l'appel MQCLOSE.

Le programme émet ensuite l'appel MQSTAT, renvoie une structure MQSTS et affiche les messages contenant le nombre de messages insérés avec succès, le nombre de messages insérés avec un avertissement et le nombre d'échecs.

Exemples de programmes de navigation

Les exemples de programme Parcourir parcourent les messages d'une file d'attente à l'aide de l'appel MQGET.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Conception de l'exemple de programme Parcourir

Le programme ouvre la file d'attente cible à l'aide de l'appel MQOPEN avec l'option MQOO_BROWSE. S'il ne parvient pas à ouvrir la file d'attente, le programme génère un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN.

Pour chaque message de la file d'attente, le programme utilise l'appel MQGET pour copier le message à partir de la file d'attente, puis affiche les données contenues dans le message. L'appel MQGET utilise les options suivantes:

MQGMO_BROWSE_NEXT
Après l'appel MQOPEN, le curseur de navigation est positionné de manière logique avant le premier message de la file d'attente. Par conséquent, cette option entraîne le renvoi du ***premier*** message lors de la première exécution de l'appel.

MQGMO_NO_WAIT
Le programme n'attend pas s'il n'y a pas de messages dans la file d'attente.

MQGMO_ACCEPT_TRUNCATED_MSG
L'appel MQGET spécifie une mémoire tampon de taille fixe. Si un message est plus long que cette mémoire tampon, le programme affiche le message tronqué, ainsi qu'un avertissement indiquant que le message a été tronqué.

Le programme montre comment vous devez effacer les zones *MsgId* et *CorrelId* de la structure MQMD après chaque appel MQGET, car l'appel définit ces zones sur les valeurs contenues dans le message qu'il extrait. L'effacement de ces zones signifie que les appels MQGET successifs extraient les messages dans l'ordre dans lequel ils sont placés dans la file d'attente.

Le programme se poursuit jusqu'à la fin de la file d'attente ; l'appel MQGET renvoie le code anomalie MQRC_NO_MSG_AVAILABLE et le programme affiche un message d'avertissement. Si l'appel MQGET échoue, le programme affiche un message d'erreur qui contient le code anomalie.

Le programme ferme ensuite la file d'attente à l'aide de l'appel MQCLOSE.

Exemples de programmes de navigation pour AIX, Linux, and Windows
Pensez à utiliser cette rubrique lorsque vous apprenez à parcourir les exemples de programmes sur AIX, Linux, and Windows.

La version C du programme prend 2 paramètres

1. Nom de la file d'attente source (nécessaire)
2. Nom du gestionnaire de files d'attente (facultatif)

Si aucun gestionnaire de files d'attente n'est spécifié, il se connecte à celui par défaut. Par exemple, entrez l'une des valeurs suivantes:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

où `myqueue` est le nom de la file d'attente à partir de laquelle les messages seront affichés et `qmanagername` est le gestionnaire de files d'attente propriétaire de `myqueue`.

Si vous omettez `qmanagername`, lors de l'exécution de l'exemple C, il suppose que le gestionnaire de files d'attente par défaut est propriétaire de la file d'attente.

La version COBOL ne comporte aucun paramètre. Il se connecte au gestionnaire de files d'attente par défaut et lorsque vous l'exécutez, vous êtes invité à:

```
Please enter the name of the target queue
```

Seuls les 50 premiers caractères de chaque message sont affichés, suivis de `- - - truncated` lorsque c'est le cas.

Les exemples de programmes Parcourir sous IBM i
Chaque programme extrait des copies de tous les messages de la file d'attente que vous indiquez lorsque vous appelez le programme ; les messages restent dans la file d'attente.

Vous pouvez utiliser la file d'attente fournie SYSTEM.SAMPLE.LOCAL; exécutez d'abord l'exemple de programme Put pour insérer des messages dans la file d'attente. Vous pouvez utiliser la file d'attente SYSTEM.SAMPLE.ALIAS, qui est un nom d'alias pour la même file d'attente locale. Le programme se poursuit jusqu'à ce qu'il atteigne la fin de la file d'attente ou qu'un appel MQI échoue.

Les exemples C vous permettent de spécifier le nom du gestionnaire de files d'attente, généralement en tant que second paramètre, de la même manière que les exemples des systèmes Windows . Exemple :

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Si aucun gestionnaire de files d'attente n'est spécifié, il se connecte à celui par défaut. Cela s'applique également aux exemples RPG. Toutefois, avec les exemples RPG, vous devez fournir un nom de gestionnaire de files d'attente au lieu de l'autoriser à utiliser la valeur par défaut.

ALW *Exemple de programme de navigateur*

L'exemple de programme Navigateur lit et écrit à la fois le descripteur de message et les zones de contenu de message de tous les messages d'une file d'attente.

L'exemple de programme est écrit comme un utilitaire, pas seulement pour démontrer une technique. Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Ce programme utilise les paramètres positionnels suivants:

1. Nom de la file d'attente source (obligatoire)
2. Nom du gestionnaire de files d'attente (obligatoire)
3. Paramètre facultatif pour les propriétés (facultatif)

Utilisez les variables d'environnement suivantes pour fournir les données d'identification utilisées pour l'authentification auprès du gestionnaire de files d'attente:

ID_UTILISATEUR_MQM
Définissez l'ID utilisateur à utiliser pour l'authentification de connexion, si vous souhaitez utiliser un ID utilisateur et un mot de passe pour vous authentifier auprès du gestionnaire de files d'attente. Le programme demande le mot de passe qui doit accompagner l'ID utilisateur.

V 9.4.0 Linux AIX **MQSAMP_TOKEN**
Indiquez une valeur non vide si vous souhaitez fournir un jeton d'authentification pour l'authentification auprès du gestionnaire de files d'attente. Le programme demande le jeton d'authentification. Les jetons d'authentification ne peuvent être utilisés que par l'exemple **amqsbcgc** qui utilise des liaisons client.

Pour exécuter ces programmes, entrez l'une des commandes suivantes:

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

où *myqueue* est le nom de la file d'attente dans laquelle les messages seront consultés et *qmanagername* est le gestionnaire de files d'attente propriétaire de *myqueue*.

Il lit chaque message de la file d'attente et écrit ce qui suit dans stdout:

- Zones de descripteur de message formatées
- Données de message (cliché au format hexadécimal et, si possible, au format alphanumérique)

Tableau 162. Valeurs admises pour le paramètre de propriété	
Valeur	**Comportement**
0	Comportement par défaut. Les propriétés qui sont distribuées à l'application dépendent de l'attribut de file d'attente **PropertyControl** à partir duquel le message est extrait.

1	Un descripteur de message est créé et utilisé avec MQGET. Les propriétés du message, à l'exception de celles contenues dans le descripteur de message (ou l'extension), sont affichées de la même manière que le descripteur de message. Exemple :
	`****Message properties****`
	`  property name: property value`
	Ou si aucune propriété n'est disponible:
	`****Message properties****`
	`  None`
	Les valeurs numériques sont affichées à l'aide de `printf`, les valeurs de chaîne sont entourées de guillemets simples et les chaînes d'octets sont entourées de X et de guillemets simples, comme pour le descripteur de message.

2	MQGMO_NO_PROPERTIES est indiqué, de sorte que seules les propriétés de descripteur de message soient renvoyées.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 est spécifié afin que toutes les propriétés soient renvoyées dans les données de message.
4	MQGMO_PROPERTIES_COMPATIBILITY est spécifié, de sorte que toutes les propriétés puissent être renvoyées selon qu'une propriété IBM MQ est incluse ou non, sinon les propriétés sont supprimées.

Le programme est limité à l'impression des 65535 premiers caractères du message et échoue avec la raison `truncated msg` si un message plus long est lu.

Pour un exemple de sortie de cet utilitaire, voir Files d'attente de navigation.

Exemple de transaction CICS

Un exemple de programme de transaction CICS est fourni, nommé amqscic0.ccs pour le code source et amqscic0 pour la version exécutable. Vous pouvez générer des transactions à l'aide des fonctions CICS standard.

Pour plus de détails sur les commandes requises pour votre plateforme, voir «Création d'une application procédurale», à la page 1027 .

La transaction lit les messages de la file d'attente de transmission SYSTEM.SAMPLE.CICS.WORKQUEUE sur le gestionnaire de files d'attente par défaut et les place dans la file d'attente locale, dont le nom est contenu dans l'en-tête de transmission du message. Tous les échecs sont envoyés à la file d'attente SYSTEM.SAMPLE.CICS.File d'attente des messages non livrés

Remarque : Vous pouvez utiliser un exemple de script MQSC amqscic0.tst pour créer ces files d'attente et des exemples de files d'attente d'entrée.

Exemple de programme Connect

L'exemple de programme Connect vous permet d'explorer l'appel MQCONNX et ses options à partir d'un client. L'exemple se connecte au gestionnaire de files d'attente à l'aide de l'appel MQCONNX, demande le nom du gestionnaire de files d'attente à l'aide de l'appel MQINQ et l'affiche. Découvrez également comment exécuter l'exemple amqscnxc.

Remarque : L'exemple de programme Connect est un exemple client. Vous pouvez le compiler et l'exécuter sur un serveur, mais la fonction n'est significative que sur un client et seuls des fichiers exécutables client sont fournis.

Exécution de l'exemple amqscnxc

La syntaxe de ligne de commande de l'exemple de programme Connect est la suivante:

```
amqscnxc [-x ConnName [-c SvrconnChannelName]] [-u User] [QMgrName]
```

Les paramètres sont facultatifs et leur ordre n'est pas important, à l'exception de QMgrName, qui, s'il est spécifié, doit venir en dernier. Les paramètres sont les suivants :

ConnName
Nom de connexion TCP/IP du gestionnaire de files d'attente du serveur

Si vous ne spécifiez pas le nom de connexion TCP/IP, MQCONNX est émis avec *ClientConnPtr* défini sur NULL.

SvrconnChannelNom
Nom du canal de connexion serveur

Si vous indiquez le nom de la connexion TCP/IP mais pas le canal de connexion serveur (l'inverse n'est pas autorisé), l'exemple utilise le nom SYSTEM.DEF.SVRCONN.

Utilisateur
Nom d'utilisateur à utiliser pour l'authentification de connexion

Si vous le spécifiez, le programme vous demandera un mot de passe pour accompagner cet ID utilisateur.

QMgrName
Nom du gestionnaire de files d'attente cible

Si vous ne spécifiez pas le gestionnaire de files d'attente cible, l'exemple se connecte au gestionnaire de files d'attente qui écoute le nom de connexion TCP/IP indiqué.

Remarque : Si vous entrez un point d'interrogation comme seul paramètre, ou si vous entrez des paramètres incorrects, vous obtenez un message expliquant comment utiliser le programme.

Si vous exécutez l'exemple sans option de ligne de commande, le contenu de la variable d'environnement MQSERVER est utilisé pour déterminer les informations de connexion. (Dans cet exemple, MQSERVER est défini sur SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com.) Une sortie similaire à la suivante s'affiche:

```
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 vous exécutez l'exemple et fournissez un nom de connexion TCP/IP et un nom de canal de connexion serveur, mais pas de nom de gestionnaire de files d'attente cible, comme ceci:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

Le nom de gestionnaire de files d'attente par défaut est utilisé et une sortie similaire à celle-ci s'affiche:

```
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 vous exécutez l'exemple et fournissez un nom de connexion TCP/IP et un nom de gestionnaire de files d'attente cible, comme ceci:

```
amqscnxc -x machine.site.company.com MACHINE
```

Une sortie similaire à la suivante s'affiche:

```
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
```

Exemple de programme de conversion de données

L'exemple de programme de conversion de données est un squelette de routine d'exit de conversion de données. Découvrez la conception de l'exemple de conversion de données.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Conception de l'échantillon de conversion de données

Chaque routine d'exit de conversion de données convertit un seul format de message nommé. Ce squelette est conçu comme un encapsuleur pour les fragments de code générés par le programme utilitaire de génération d'exit de conversion de données.

L'utilitaire produit un fragment de code pour chaque structure de données ; plusieurs de ces structures constituent un format, de sorte que plusieurs fragments de code sont ajoutés à ce squelette pour produire une routine pour effectuer la conversion de données du format entier.

Le programme vérifie ensuite si la conversion est un succès ou un échec et renvoie les valeurs requises à l'appelant.

Exemples de coordination de base de données

Deux exemples sont fournis pour illustrer la façon dont IBM MQ peut coordonner les mises à jour de IBM MQ et les mises à jour de base de données au sein de la même unité d'oeuvre.

Ces exemples sont les suivants:

1. AMQSXAS0 (en C) ou AMQ0XAS0 (en COBOL), qui met à jour une base de données unique au sein d'une unité d'oeuvre IBM MQ .
2. AMQSXAG0 (en C) ou AMQ0XAG0 (en COBOL), AMQSXAB0 (en C) ou AMQ0XAB0 (en COBOL) et AMQSXAF0 (en C) ou AMQ0XAF0 (en COBOL), qui mettent à jour ensemble deux bases de données au sein d'une unité d'oeuvre IBM MQ , en indiquant comment accéder à plusieurs bases de données. Ces exemples sont fournis pour illustrer l'utilisation de l'appel MQBEGIN, des appels SQL mixtes et IBM MQ , ainsi que l'emplacement et le moment de la connexion à une base de données.

Figure 128, à la page 1114 montre comment les exemples fournis sont utilisés pour mettre à jour les bases de données:

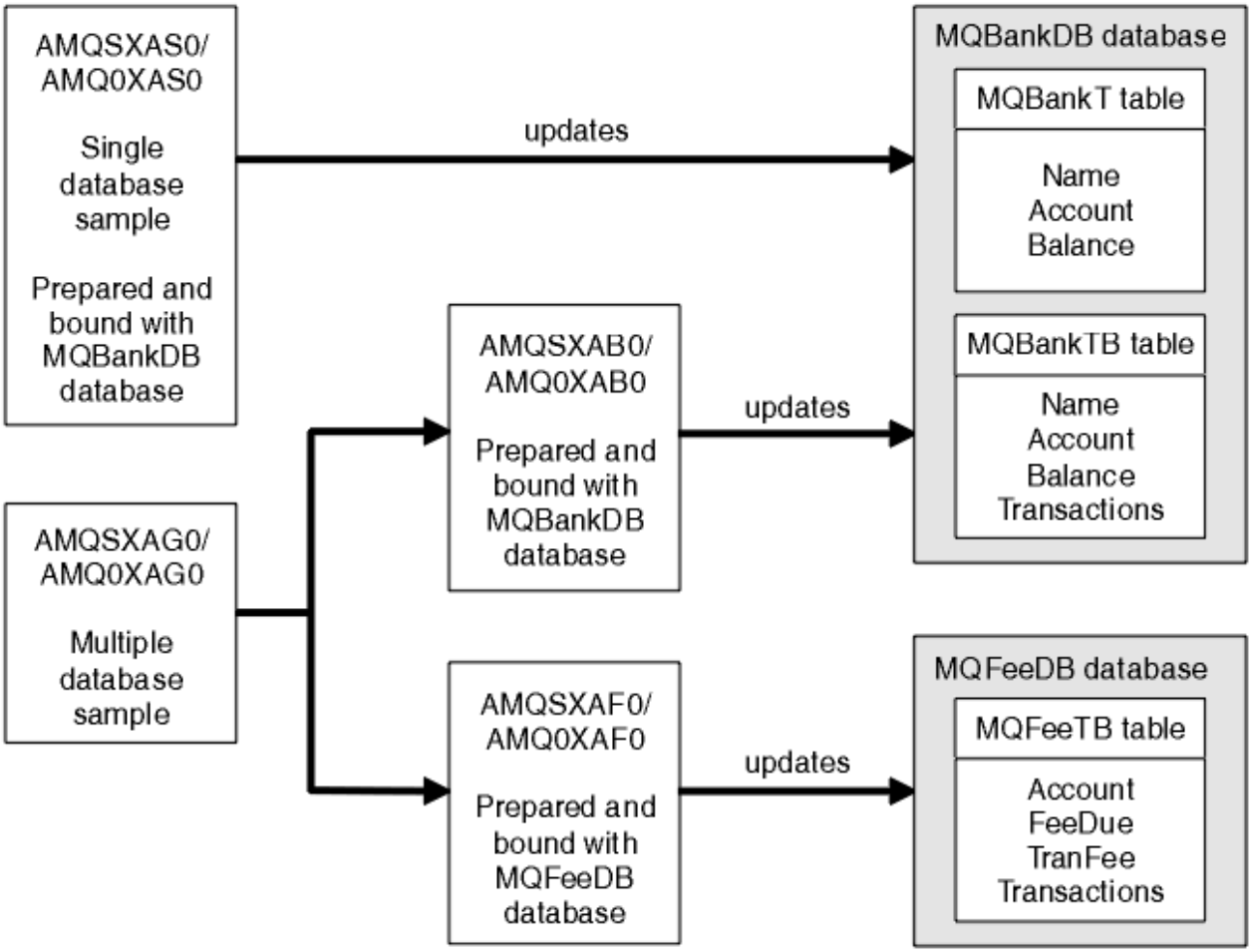


Figure 128. Exemples de coordination de base de données

Les programmes lisent un message à partir d'une file d'attente (sous le point de synchronisation), puis, à l'aide des informations du message, obtiennent les informations appropriées de la base de données et les mettent à jour. Le nouvel état de la base de données est ensuite imprimé.

La logique du programme est la suivante:

1. Utiliser le nom de la file d'entrée à partir de l'argument de programme
2. Connexion au gestionnaire de files d'attente par défaut (ou éventuellement au nom fourni en C) à l'aide de MQCONN
3. Ouvrir une file d'attente (à l'aide de MQOPEN) pour l'entrée alors qu'il n'y a pas d'échec
4. Démarrage d'une unité de travail à l'aide de MQBEGIN
5. Obtenir le message suivant (à l'aide de MQGET) de la file d'attente sous le point de synchronisation
6. Obtention d'informations à partir de bases de données
7. Mettre à jour les informations des bases de données
8. Validation des modifications à l'aide de MQCMIT
9. Imprimer les informations mises à jour (aucun message disponible ne compte comme un échec et la boucle se termine)
10. Fermeture de la file d'attente à l'aide de MQCLOSE
11. Se déconnecter de la file d'attente à l'aide de MQDISC

Les curseurs SQL sont utilisés dans les exemples, de sorte que les lectures à partir des bases de données (c'est-à-dire, plusieurs instances) sont verrouillées pendant le traitement d'un message, ce qui permet

à plusieurs instances de ces programmes de s'exécuter simultanément. Les curseurs sont explicitement ouverts, mais implicitement fermés par l'appel MQCMIT.

L'exemple de base de données unique (AMQSXAS0 ou AMQ0XAS0) ne comporte aucune instruction SQL CONNECT et la connexion à la base de données est implicitement établie par IBM MQ avec l'appel MQBEGIN. L'exemple de plusieurs bases de données (AMQSXAG0 ou AMQ0XAG0, AMQSXAB0 ou AMQ0XAB0et AMQSXAF0 ou AMQ0XAF0) comporte des instructions SQL CONNECT, car certains produits de base de données n'autorisent qu'une seule connexion active. Si tel n'est pas le cas pour votre produit de base de données ou si vous accédez à une seule base de données dans plusieurs produits de base de données, les instructions SQL CONNECT peuvent être supprimées.

Les exemples étant préparés avec le produit de base de données IBM Db2 , vous devrez peut-être les modifier pour les utiliser avec d'autres produits de base de données.

La vérification des erreurs SQL utilise des routines dans UTIL.C et CHECKERR.CBL fourni par Db2. Ils doivent être compilés ou remplacés avant la compilation et la liaison.

Remarque : Si vous utilisez la source Micro Focus COBOL CHECKERR.MFC pour la vérification des erreurs SQL, vous devez mettre l'ID programme en majuscules, c'est-à-dire CHECKERR, pour que AMQ0XAS0 soit correctement lié.

Création des bases de données et des tables
Créez les bases de données et les tables avant de compiler les exemples.

Pour créer les bases de données, utilisez la méthode habituelle pour votre produit de base de données, par exemple:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Créez les tables à l'aide d'instructions SQL comme suit:

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.
```

Entrez des données dans les tables à l'aide d'instructions SQL comme suit:

```
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);
⋮
```

Remarque : Pour COBOL, utilisez les mêmes instructions SQL mais ajoutez END_EXEC à la fin de chaque ligne.

Précompilation, compilation et liaison des exemples
En savoir plus sur la précompilation, la compilation et la liaison d'exemples en C et COBOL.

Précompilez les fichiers `.SQC` (en C) et `.SQB` (en COBOL) et liez-les à la base de données appropriée pour générer les fichiers `.C` ou `.CBL` . Pour ce faire, utilisez la méthode standard de votre produit de base de données.

Précompilation 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
```

Précompilation 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
```

Compilation et liaison

Les exemples de commande suivants utilisent les symboles *DB2TOP* et *MQ_INSTALLATION_PATH*. *DB2TOP* représente le répertoire d'installation du produit Db2 . *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.

- AIX Sous AIX, le chemin de répertoire est:

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

- Windows Sur les systèmes Windows , le chemin de répertoire dépend du chemin choisi lors de l'installation du produit. Si vous avez choisi les paramètres par défaut, le chemin d'accès est le suivant:

```
c:\sqllib
```

Remarque : Avant d'émettre la commande link sur les systèmes Windows , vérifiez que la variable d'environnement LIB contient des chemins d'accès aux bibliothèques Db2 et IBM MQ .

Copiez les fichiers suivants dans un répertoire temporaire:

- Le fichier `amqsxag0.c` de votre installation IBM MQ

  Remarque : Ce fichier se trouve dans les répertoires suivants:

  - Linux AIX Sur les systèmes AIX and Linux :

    ```
    MQ_INSTALLATION_PATH/samp/xatm
    ```

  - Windows Sur les systèmes Windows :

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

- Les fichiers `.c` que vous avez obtenus en précompilant les fichiers source `.sqc` , `amqsxas0.sqc`, `amqsxaf0.sqc`et `amqsxab0.sqc`.
- Les fichiers `util.c` et `util.h` de votre installation Db2 .

  Remarque : Ces fichiers se trouvent dans le répertoire suivant:

  ```
  DB2TOP/samples/c
  ```

Générez les fichiers objet pour chaque fichier `.c` à l'aide de la commande de compilateur suivante pour la plateforme que vous utilisez:

- AIX AIX

  ```
  xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
  FILENAME.o FILENAME.c
  ```

- Windows Systèmes Windows

  ```
  cl /c /I MQ_INSTALLATION_PATH\tools\c\include /I DB2TOP\include
  FILENAME.c
  ```

Générez le fichier exécutable `amqsxag0` à l'aide de la commande de lien suivante pour la plateforme que vous utilisez:

- AIX AIX

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

- Windows Systèmes Windows

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

Générez le fichier exécutable `amqsxas0` à l'aide des commandes de compilation et de liaison suivantes pour la plateforme que vous utilisez:

- AIX AIX

```
xlc_r -H512 -T512 -L DB2TOP/lib -ldb2
-L MQ_INSTALLATION_PATH/lib -lmqm util.o amqsxas0.o -o amqsxas0
```

- Windows Systèmes Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Renseignements supplémentaires

AIX Si vous travaillez sur AIX et souhaitez accéder à Oracle, utilisez le compilateur xlc_r et le lien vers libmqm_r.a.

Exécution des exemples
Utilisez ces informations pour apprendre à configurer le gestionnaire de files d'attente avant d'exécuter des exemples de coordination de base de données sur C et COBOL.

Avant d'exécuter les exemples, configurez le gestionnaire de files d'attente avec le produit de base de données que vous utilisez. Pour plus d'informations sur la procédure à suivre, voir Scénario 1: le gestionnaire de files d'attente effectue la coordination.

Les titres suivants fournissent des informations sur la façon d'exécuter des exemples en C et COBOL:

- «Echantillons C», à la page 1118
- «Exemples COBOL», à la page 1119

Echantillons C

Les messages doivent être au format suivant pour être lus à partir d'une file d'attente:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

AMQSPUT peut être utilisé pour placer les messages dans la file d'attente.

Les exemples de coordination de base de données prennent deux paramètres:

1. Nom de la file d'attente (obligatoire)
2. Nom du gestionnaire de files d'attente (facultatif)

En supposant que vous avez créé et configuré un gestionnaire de files d'attente pour l'exemple de base de données unique appelé singDBQM, avec une file d'attente appelée singDBQ, vous incrémentez le compte de M. Fred Bloggs de 50 comme suit:

```
AMQSPUT singDBQ singDBQM
```

Entrez ensuite le message suivant:

```
UPDATE Balance change=50 WHERE Account=1
```

Vous pouvez placer plusieurs messages dans la file d'attente.

```
AMQSXAS0 singDBQ singDBQM
```

Le statut mis à jour du compte de M. Fred Bloggs est ensuite imprimé.

En supposant que vous avez créé et configuré un gestionnaire de files d'attente pour l'exemple à plusieurs bases de données appelé multDBQM, avec une file d'attente appelée multDBQ, vous décrémentez le compte de Mme Mary Brown de 75 comme suit:

```
AMQSPUT multDBQ multDBQM
```

Entrez ensuite le message suivant:

```
UPDATE Balance change=-75 WHERE Account=3
```

Vous pouvez placer plusieurs messages dans la file d'attente.

```
AMQSXAG0 multDBQ multDBQM
```

Le statut mis à jour du compte de Mme Mary Brown est ensuite imprimé.

Exemples COBOL

Les messages doivent être au format suivant pour être lus à partir d'une file d'attente:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

Pour plus de simplicité, le `Balance change` doit être un nombre de huit caractères signé et le `Account` doit être un nombre de huit caractères.

L'exemple AMQSPUT peut être utilisé pour placer les messages dans la file d'attente.

Les exemples ne prennent aucun paramètre et utilisent le gestionnaire de files d'attente par défaut. Il peut être configuré pour exécuter un seul des exemples à la fois. En supposant que vous avez configuré le gestionnaire de files d'attente par défaut pour l'exemple de base de données unique, avec une file d'attente appelée singDBQ, vous incrémentez le compte de M. Fred Bloggs de 50 comme suit:

```
AMQSPUT singDBQ
```

Entrez ensuite le message suivant:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

Vous pouvez placer plusieurs messages dans la file d'attente:

```
AMQ0XAS0
```

Entrez le nom de la file d'attente:

```
singDBQ
```

Le statut mis à jour du compte de M. Fred Bloggs est ensuite imprimé.

En supposant que vous avez configuré le gestionnaire de files d'attente par défaut pour l'exemple de plusieurs bases de données, avec une file d'attente appelée multDBQ, vous décrémentez le compte de Mme Mary Brown de 75 comme suit:

```
AMQSPUT multDBQ
```

Entrez ensuite le message suivant:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

Vous pouvez placer plusieurs messages dans la file d'attente:

```
AMQ0XAG0
```

Entrez le nom de la file d'attente:

```
multDBQ
```

Le statut mis à jour du compte de Mme Mary Brown est ensuite imprimé.

Exemple de gestionnaire de files d'attente de rebut

Un exemple de gestionnaire de files d'attente de rebut est fourni. Le nom de la version exécutable est `amqsdlq`. Si vous souhaitez un gestionnaire de files d'attente de rebut différent de **RUNMQDLQ**, la source de l'exemple est disponible et vous pouvez l'utiliser comme base.

L'exemple est similaire au gestionnaire de rebut fourni dans le produit, mais les rapports de trace et d'erreur sont différents. Deux variables d'environnement sont disponibles:

TRACE ODQ
: Définissez sur YES ou `yes` pour activer la fonction de trace.

MSG _ODQ
: Indiquez le nom du fichier contenant les messages d'erreur et d'information. Le fichier fourni est appelé `amqsdlq.msg`.

Vous devez faire connaître ces variables à votre environnement à l'aide des commandes **`export`** ou **`set`** , en fonction de votre plateforme ; la trace est désactivée à l'aide de la commande **`unset`** .

Vous pouvez modifier le fichier de messages d'erreur, `amqsdlq.msg`, en fonction de vos besoins. L'exemple insère des messages dans stdout, **pas** dans le fichier journal des erreurs IBM MQ .

Pour plus d'informations sur le fonctionnement du gestionnaire de rebut et sur son exécution, voir Traitement des messages dans une IBM MQ file d'attente de rebut ou le *Guide de gestion des systèmes* de votre plateforme.

Exemple de programme Liste de distribution

L'exemple de liste de distribution amqsptl0 fournit un exemple d'insertion d'un message dans plusieurs files d'attente de messages. Il est basé sur l'exemple MQPUT, amqsput0.

Exécution de l'exemple de liste de distribution amqsptl0

L'exemple Liste de distribution s'exécute de la même manière que les exemples d'insertion.

Il prend les paramètres suivants:

- Noms des files d'attente
- Noms des gestionnaires de files d'attente

Ces valeurs sont entrées sous forme de paires. Exemple :

```
amqsptl0 queue1 qmanagername1 queue2 qmanagername2
```

Les files d'attente sont ouvertes à l'aide de MQOPEN et les messages sont insérés dans les files d'attente à l'aide de MQPUT. Les codes raison sont renvoyés si l'un des noms de file d'attente ou de gestionnaire de files d'attente n'est pas reconnu.

N'oubliez pas de définir des canaux entre les gestionnaires de files d'attente afin que les messages puissent circuler entre eux. L'exemple de programme ne le fait pas pour vous.

Conception de l'échantillon Liste de distribution

Les enregistrements d'insertion de message (MQPMR) spécifient des attributs de message pour chaque destination. L'exemple fournit des valeurs pour *MsgId* et *CorrelId*, qui remplacent les valeurs spécifiées dans la structure MQMD.

La zone *PutMsgRecFields* de la structure MQPMO indique les zones présentes dans les MQPMR:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Ensuite, l'exemple alloue les enregistrements de réponse et les enregistrements d'objet. Les enregistrements d'objet (MQOR) requièrent au moins une paire de noms et un nombre pair de noms, c'est-à-dire *ObjectName* et *ObjectQMgrName*.

L'étape suivante consiste à se connecter aux gestionnaires de files d'attente à l'aide de MQCONN. L'exemple tente de se connecter au gestionnaire de files d'attente associé à la première file d'attente du MQOR ; en cas d'échec, il passe successivement par les enregistrements d'objet. Vous êtes informé s'il n'est pas possible de se connecter à un gestionnaire de files d'attente et que le programme s'arrête.

Les files d'attente cible sont ouvertes à l'aide de MQOPEN et le message est inséré dans ces files d'attente à l'aide de MQPUT. Les problèmes et les échecs sont signalés dans les enregistrements de réponse (MQRRs).

Enfin, les files d'attente cible sont fermées à l'aide de MQCLOSE et le programme se déconnecte du gestionnaire de files d'attente à l'aide de MQDISC. Les mêmes enregistrements de réponse sont utilisés pour chaque appel indiquant *CompCode* et *Reason*.

Exemples de programmes Echo

Les exemples de programme Echo envoient un message d'une file d'attente de messages à la file d'attente de réponses.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Les programmes sont destinés à s'exécuter en tant que programmes déclenchés.

Sur les systèmes IBM i, AIX, Linux, and Windows , leur seule entrée est une structure MQTMC2 (message de déclenchement) qui contient le nom d'une file d'attente cible et le gestionnaire de files d'attente. La version COBOL utilise le gestionnaire de files d'attente par défaut.

IBM i Sous IBM i, pour que le processus de déclenchement fonctionne, assurez-vous que l'exemple de programme Echo que vous souhaitez utiliser est déclenché par les messages arrivant dans la file d'attente SYSTEM.SAMPLE.ECHO. Pour ce faire, indiquez le nom de l'exemple de programme Echo que vous souhaitez utiliser dans la zone *ApplId* de la définition de processus SYSTEM.SAMPLE.ECHOPROCESS. (Pour cela, vous pouvez utiliser la commande CHGMQMPRC ; pour plus de détails, voir Modification du processus MQ (CHGMQMPRC).) Le type de déclencheur de l'exemple de file d'attente est FIRST. Par conséquent, s'il y a déjà des messages dans la file d'attente avant l'exécution de l'exemple Demande, l'exemple Echo n'est pas déclenché par les messages que vous envoyez.

Une fois la définition correctement définie, démarrez d'abord AMQSERV4 dans un travail, puis démarrez AMQSREQ4 dans un autre travail. Vous pouvez utiliser AMQSTRG4 au lieu de AMQSERV4, mais les retards potentiels de soumission des travaux peuvent rendre plus difficile le suivi de ce qui se passe.

Utilisez les exemples de programmes de demande pour envoyer des messages à la file d'attente SYSTEM.SAMPLE.ECHO. Les exemples de programme Echo envoient un message de réponse contenant les données du message de demande à la file d'attente de réponse indiquée dans le message de demande.

Conception des exemples de programmes Echo

Le programme ouvre la file d'attente nommée dans la structure de message de déclenchement qu'il a transmise lors de son démarrage. (Pour plus de clarté, il s'agit de la file d'attente des demandes.) Le programme utilise l'appel MQOPEN pour ouvrir cette file d'attente pour l'entrée partagée.

Le programme utilise l'appel MQGET pour supprimer des messages de cette file d'attente. Cet appel utilise les options MQGMO_ACCEPT_TRUNCATED_MSG, MQGMO_CONVERT et MQGMO_WAIT, avec un intervalle d'attente de 5 secondes. Le programme teste le descripteur de chaque message pour voir s'il s'agit d'un message de demande ; si ce n'est pas le cas, le programme supprime le message et affiche un message d'avertissement.

Pour chaque ligne d'entrée, le programme lit ensuite le texte dans une mémoire tampon et utilise l'appel MQPUT1 pour placer un message de demande, contenant le texte de cette ligne, dans la file d'attente de réponse.

Si l'appel MQGET échoue, le programme insère un message de rapport dans la file d'attente de réponses, en définissant la zone *Feedback* du descripteur de message sur le code anomalie renvoyé par MQGET.

Lorsqu'il ne reste aucun message dans la file d'attente des demandes, le programme ferme cette file d'attente et se déconnecte du gestionnaire de files d'attente.

IBM i Sous IBM i, le programme peut également répondre aux messages envoyés à la file d'attente à partir de plateformes autres que IBM MQ for IBM i, bien qu'aucun exemple ne soit fourni pour cette situation. Pour que le programme ECHO fonctionne:

- Ecrivez un programme en spécifiant correctement les paramètres **Format**, **Encoding**et **CCSID** pour envoyer des messages de demande de texte.

  Le programme ECHO demande au gestionnaire de files d'attente d'effectuer la conversion des données de message, si nécessaire.
- Indiquez CONVERT (*YES) sur le canal émetteur IBM MQ for IBM i si le programme que vous avez écrit ne permet pas une conversion similaire pour la réponse.

Exemples de programmes Get

Les exemples de programme Get extront des messages d'une file d'attente à l'aide de l'appel MQGET.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Conception de l'exemple de programme Get

Le programme ouvre la file d'attente cible à l'aide de l'appel MQOPEN avec l'option MQOO_INPUT_AS_Q_DEF. S'il ne parvient pas à ouvrir la file d'attente, le programme affiche un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN.

Pour chaque message de la file d'attente, le programme utilise l'appel MQGET pour supprimer le message de la file d'attente, puis affiche les données contenues dans le message. L'appel MQGET utilise l'option MQGMO_WAIT, en spécifiant une valeur *WaitInterval* de 15 secondes, de sorte que le programme attend pendant cette période s'il n'y a pas de message dans la file d'attente. Si aucun message n'arrive avant l'expiration de cet intervalle, l'appel échoue et renvoie le code anomalie MQRC_NO_MSG_AVAILABLE.

Le programme montre comment vous devez effacer les zones *MsgId* et *CorrelId* de la structure MQMD après chaque appel MQGET car l'appel définit ces zones sur les valeurs contenues dans le message qu'il

extrait. L'effacement de ces zones signifie que les appels MQGET successifs extraient les messages dans l'ordre dans lequel ils sont placés dans la file d'attente.

L'appel MQGET spécifie une mémoire tampon de taille fixe. Si un message est plus long que cette mémoire tampon, l'appel échoue et le programme s'arrête.

Le programme se poursuit jusqu'à ce que l'appel MQGET renvoie le code anomalie MQRC_NO_MSG_AVAILABLE ou que l'appel MQGET échoue. Si l'appel échoue, le programme affiche un message d'erreur contenant le code anomalie.

Le programme ferme ensuite la file d'attente à l'aide de l'appel MQCLOSE.

Exécution des exemples amqsget et amqsgetc

Ces programmes prennent chacun les paramètres positionnels suivants:

1. Nom de la file d'attente source (obligatoire)
2. Nom du gestionnaire de files d'attente (facultatif)

   Si aucun gestionnaire de files d'attente n'est spécifié, **amqsget** se connecte au gestionnaire de files d'attente par défaut et **amqsgetc** se connecte au gestionnaire de files d'attente identifié par la variable d'environnement *MQSERVER* ou le fichier de définition de canal du client.
3. Les options d'ouverture (facultatif)

   Si les options ouvertes ne sont pas spécifiées, l'exemple utilise la valeur 8193, qui correspond à la combinaison des deux options suivantes:

   - MQOO_INPUT_AS_Q_DEF
   - MQOO_FAIL_IF_QUIESCING
4. Options de fermeture (facultatif)

   Si les options de fermeture ne sont pas spécifiées, l'exemple utilise la valeur 0, qui est MQCO_NONE.

Utilisez les variables d'environnement suivantes pour fournir les données d'identification utilisées pour l'authentification auprès du gestionnaire de files d'attente:

ID_UTILISATEUR_MQM
Définissez l'ID utilisateur à utiliser pour l'authentification de connexion, si vous souhaitez utiliser un ID utilisateur et un mot de passe pour vous authentifier auprès du gestionnaire de files d'attente. Le programme demande le mot de passe qui doit accompagner l'ID utilisateur.

V 9.4.0 Linux AIX **MQSAMP_TOKEN**
Indiquez une valeur non vide si vous souhaitez fournir un jeton d'authentification pour l'authentification auprès du gestionnaire de files d'attente. Le programme demande le jeton d'authentification. Les jetons d'authentification ne peuvent être utilisés que par l'exemple **amqsgetc** qui utilise des liaisons client.

Pour exécuter ces programmes, entrez l'un des éléments suivants:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

où *monfile d'attente* est le nom de la file d'attente à partir de laquelle le programme extrait les messages et *nom_gestionnaire_files* est le gestionnaire de files d'attente qui possède *monfile d'attente*.

Utilisation d'amqsget et d'amqsgetc

Notez que **amqsget** effectue une connexion locale au gestionnaire de files d'attente, à l'aide de la mémoire partagée pour la connexion au gestionnaire de files d'attente. En tant que tel, il ne peut être exécuté que sur le système sur lequel réside le gestionnaire de files d'attente, tandis que **amqsgetc** effectue une connexion de type client (même s'il se connecte à un gestionnaire de files d'attente sur le même système).

Lorsque vous utilisez **amqsgetc** , vous devez fournir les détails de l'application indiquant comment atteindre le gestionnaire de files d'attente, en termes d'hôte du gestionnaire de files d'attente ou d'adresse IP et de port d'écoute du gestionnaire de files d'attente.

Normalement, cette opération est effectuée à l'aide de la variable d'environnement **MQSERVER** ou en définissant les détails de connexion à l'aide d'une table de définition de canal du client, qui peut également être fournie à **amqsgetc** à l'aide de variables d'environnement ; par exemple, voir MQCCDTURL.

Voici un exemple d'utilisation de **MQSERVER**, lors de la connexion à un gestionnaire de files d'attente en local, avec un programme d'écoute s'exécutant sur le port 1414 et utilisant le canal de connexion serveur par défaut:

```
export MQSERVER="SYSTEM.DEF.SVRCONN/TCP/ localhost(1414)"
```

Exemples de programmes à haute disponibilité

Les exemples de programmes à haute disponibilité **amqsghac**, **amqsphac**et **amqsmhac** utilisent la reconnexion client automatisée pour illustrer la reprise après l'échec d'un gestionnaire de files d'attente. **amqsfhac** vérifie qu'un gestionnaire de files d'attente utilisant le stockage en réseau conserve l'intégrité des données suite à une défaillance.

Les programmes **amqsghac**, **amqsphac**et **amqsmhac** sont démarrés à partir de la ligne de commande et peuvent être utilisés conjointement pour démontrer la reconnexion après l'échec d'une instance d'un gestionnaire de files d'attente multi-instance.

Vous pouvez également utiliser les exemples **amqsghac**, **amqsphac**et **amqsmhac** pour illustrer la reconnexion du client à des gestionnaires de files d'attente à instance unique, généralement configurés dans un groupe de gestionnaires de files d'attente.

Pour simplifier l'exemple et faciliter sa configuration, vous voyez les exemples de programmes se reconnecter à un gestionnaire de files d'attente à instance unique qui est démarré, arrêté, puis redémarré ; voir «Configuration et contrôle du gestionnaire de files d'attente», à la page 1126.

Utilisez **amqsfhac** en parallèle avec **amqmfsck** pour vérifier l'intégrité du système de fichiers. Pour plus d'informations, voir **amqmfsck** (vérification du système de fichiers) et Vérification du comportement du système de fichiers partagé .

amqsphac *queueName* [*qMgrNom*]

- **amqsphac** est une application IBM MQ MQI client . Il place une séquence de messages dans une file d'attente avec un délai de deux secondes entre chaque message et affiche les événements envoyés à son gestionnaire d'événements.
- Aucun point de synchronisation n'est utilisé pour insérer des messages dans la file d'attente.
- La reconnexion peut être effectuée à n'importe quel gestionnaire de files d'attente du même groupe de gestionnaires de files d'attente.

amqsghac *queueName* [*qMgr*]

- **amqsghac** est une application IBM MQ MQI client . Il extrait des messages d'une file d'attente et affiche les événements envoyés à son gestionnaire d'événements.
- Aucun point de synchronisation n'est utilisé pour extraire des messages de la file d'attente.
- La reconnexion peut être effectuée à n'importe quel gestionnaire de files d'attente du même groupe de gestionnaires de files d'attente.

amqsmhac -s *sourceQueueNom* -t *targetQueueNom* [-m *qMgrNom*] [-w *waitInterval*]

- **amqsmhac** est une application IBM MQ MQI client . Il copie les messages d'une file d'attente vers une autre avec un intervalle d'attente par défaut de 15 minutes après la réception du dernier message avant la fin du programme.
- Les messages sont copiés dans le point de synchronisation.
- La reconnexion ne peut être effectuée qu'avec le même gestionnaire de files d'attente.

amqsfhac *QueueManagerNom QueueName SideQueueNom InTransactionNombre RepeatCount* (0 | 1 | 2)

- **amqsfhac** est une application IBM MQ MQI client . Il vérifie qu'un gestionnaire de files d'attente multi-instance IBM MQ utilisant le stockage en réseau, tel qu'un NAS ou un système de fichiers de cluster, conserve l'intégrité des données. Suivez les étapes pour exécuter **amqsfhac** dans Vérification du comportement du système de fichiers partagé.
- Il utilise l'option MQCNO_RECONNECT_Q_MGR lors de la connexion à *QueueManagerName*. Il se reconnecte automatiquement en cas de basculement du gestionnaire de files d'attente.
- Elle place les messages persistants *InTransactionCount***RepeatCount* dans *QueueName* , au cours de laquelle vous faites basculer le gestionnaire de files d'attente plusieurs fois. **amqsfhac** se reconnecte au gestionnaire de files d'attente à chaque fois et continue. Le test consiste à s'assurer qu'aucun message n'est perdu.
- *Les messagesInTransactionCount* sont insérés dans chaque transaction. La transaction est répétée *RepeatCount* nombre de fois. Si un incident se produit dans une transaction, **amqsfhac** annule et soumet à nouveau la transaction lorsque **amqsfhac** se reconnecte au gestionnaire de files d'attente.
- Il insère également des messages dans *SideQueueNom*. Il utilise *SideQueueName* pour vérifier si tous les messages sont validés ou annulés à partir de *QueueName* . S'il détecte une incohérence, il écrit un message d'erreur.
- Modifiez la quantité de trace de sortie de **amqsfhac** en définissant le dernier paramètre sur (0|1|2).

  0
  Sortie la plus faible.

  1
  Sortie intermédiaire.

  2
  La plupart des sorties.

Configuration d'une connexion client

Vous devez configurer un canal de connexion client et serveur pour exécuter les exemples. La procédure de vérification du client explique comment configurer un environnement de test client.

Vous pouvez également utiliser la configuration fournie dans l'exemple suivant.

Exemple avec amqsghac, amqsphacet amqsmhac

L'exemple illustre des clients reconnectables à l'aide d'un gestionnaire de files d'attente à instance unique.

Les messages sont placés dans la file d'attente SOURCE par **amqsphac**, transférés à TARGET par **amqsmhac**et extraits de TARGET par **amqsghac** ; voir Figure 129, à la page 1125.

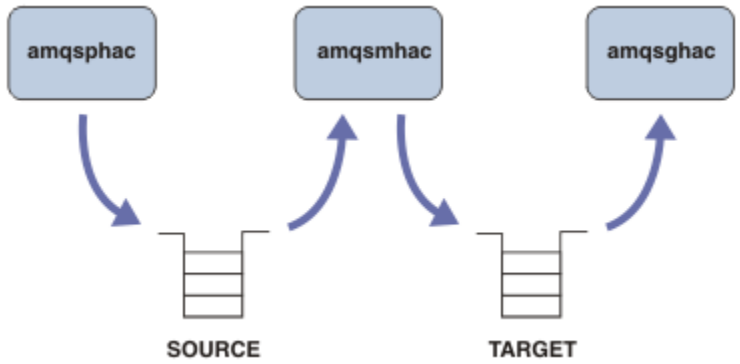


Figure 129. Exemples de client reconnectable

Procédez comme suit pour exécuter les exemples.

1. Créez un fichier `hasamples.tst` contenant les commandes suivantes:

```
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. Entrez les commandes suivantes dans une invite de commande:
   a. `crtmqm QM1`
   b. `strmqm QM1`
   c. `runmqsc QM1 < hasamples.tst`
3. Définissez la variable d'environnement **MQCHLLIB** sur le chemin d'accès au fichier de définition de canal du client `AMQCLCHL.TAB` ; par exemple, SET `MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc`.
4. Ouvrez trois nouvelles fenêtres avec **MQCHLLIB** défini ; par exemple, sous Windows, tapez **start** trois fois à l'invite de commande précédente en démarrant chaque programme dans l'une des fenêtres. Voir l'étape «5», à la page 1127 dans «Configuration et contrôle du gestionnaire de files d'attente», à la page 1126.)
5. Entrez la commande `endmqm -r -p QM1` pour arrêter le gestionnaire de files d'attente, puis autorisez les clients à se reconnecter.
6. Entrez la commande `strmqm QM1` pour redémarrer le gestionnaire de files d'attente.

Les résultats de l'exécution des exemples **amqsghac**, **amqsphac**et **amqsmhac** sous Windows sont présentés dans les exemples suivants.

Configuration et contrôle du gestionnaire de files d'attente

1. Créez le gestionnaire de files d'attente.

```
C:\> crtmqm QM1
IBM MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Mémorisez le répertoire de données pour définir la variable **MQCHLLIB** ultérieurement.
2. Démarrez le gestionnaire de files d'attente.

```
C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

3. Créez les files d'attente et les canaux, modifiez le port d'écoute et démarrez le programme d'écoute et le canal.

```
C:\> runmqsc QM1 < hasamples.tst

5724-H72 (C) Copyright IBM Corp. 1994, 2024.  ALL RIGHTS RESERVED.
```

```
Starting MQSC for queue manager QM1.

     1 : DEFINE QLOCAL(SOURCE) REPLACE
AMQ8006: IBM MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: IBM MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: IBM MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: IBM MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: IBM MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start IBM MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start IBM MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. Rendez la table de canaux client connue des clients.

   Utilisez le répertoire de données renvoyé par la commande **crtmqm** à l'étape «1», à la page 1126et ajoutez-y le répertoire `@ipcc` pour définir la variable **MQCHLLIB** .

```
C:\> SET MQCHLLIB=C:\IBM\MQ\MQ7\Data\qmgrs\QM1\@ipcc
```

5. Démarrer les exemples de programme dans les autres fenêtres

```
C:\> start amqsphac SOURCE QM1
C:\> start amqsmhac -s SOURCE -t TARGET -m QM1
C:\> start amqsghac TARGET QM1
```

6. Arrêtez le gestionnaire de files d'attente et redémarrez-le.

```
C:\> endmqm -r -p QM1

Waiting for queue manager 'QM1' to end.
IBM MQ queue manager 'QM1' ending.
IBM MQ queue manager 'QM1' ended.


C:\> strmqm QM1

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
Message 3
message Message 4
message Message 5
```

amqsmhac

```
Sample AMQSMHA0 start
```

```
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message Message 1
message Message 2
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message Message 3
message Message 4
message Message 5
```

Tâches associées

Vérification du comportement du système de fichiers partagé

Référence associée

amqmfsck (vérification du système de fichiers)

Exemples de programmes Inquire

Les exemples de programme d'interrogation s'interrogent sur certains attributs d'une file d'attente à l'aide de l'appel MQINQ.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Ces programmes étant destinés à être exécutés en tant que programmes déclenchés, leur seule entrée est une structure MQTMC2 (message de déclenchement) pour IBM MQ for Multiplatforms. Cette structure contient le nom d'une file d'attente cible dont les attributs doivent être renseignés. La version C utilise également le nom du gestionnaire de files d'attente. La version COBOL utilise le gestionnaire de files d'attente par défaut.

Pour que le processus de déclenchement fonctionne, assurez-vous que l'exemple de programme Inquire que vous souhaitez utiliser est déclenché par les messages arrivant dans la file d'attente SYSTEM.SAMPLE.INQ. Pour ce faire, indiquez le nom de l'exemple de programme Inquire que vous souhaitez utiliser dans la zone *ApplicId* de la définition de processus SYSTEM.SAMPLE.INQPROCESS.

IBM i Pour IBM i, vous pouvez utiliser la commande CHGMQMPRC pour cela ; pour plus de détails, voir Change MQ Process (CHGMQMPRC). Le type de déclencheur de l'exemple de file d'attente est FIRST ; s'il existe déjà des messages dans la file d'attente avant l'exécution de l'exemple de demande, l'exemple d'interrogation n'est pas déclenché par les messages que vous envoyez.

Lorsque vous avez correctement défini la définition:

- ALW Pour AIX, Linux, and Windows, démarrez le programme **runmqtrm** dans une session, puis démarrez le programme `amqsreq` dans une autre.
- IBM i Pour IBM i, démarrez le programme AMQSERV4 dans une session, puis démarrez le programme AMQSREQ4 dans une autre. Vous pouvez utiliser AMQSTRG4 au lieu de AMQSERV4, mais les retards potentiels de soumission des travaux peuvent rendre plus difficile le suivi de ce qui se passe.

Utilisez les exemples de programmes de demande pour envoyer des messages de demande, chacun contenant uniquement un nom de file d'attente, à la file d'attente SYSTEM.SAMPLE.INQ. Pour chaque message de demande, les exemples de programme Inquire envoient un message de réponse contenant des informations sur la file d'attente indiquée dans le message de demande. Les réponses sont envoyées à la file d'attente de réponse indiquée dans le message de demande.

IBM i Sous IBM i, si l'exemple de membre de fichier d'entrée est QMQMSAMP.AMQSDATA(INQ) est utilisé, la dernière file d'attente nommée n'existe pas, de sorte que l'exemple renvoie un message de rapport avec un code raison pour l'échec.

Conception de l'exemple de programme Inquire

Le programme ouvre la file d'attente nommée dans la structure de message de déclenchement qu'il a transmise lors de son démarrage. (Pour plus de clarté, nous appellerons cela la *file d'attente des demandes*.) Le programme utilise l'appel MQOPEN pour ouvrir cette file d'attente pour l'entrée partagée.

Le programme utilise l'appel MQGET pour supprimer des messages de cette file d'attente. Cet appel utilise les options MQGMO_ACCEPT_TRUNCATED_MSG et MQGMO_WAIT, avec un intervalle d'attente de 5 secondes. Le programme teste le descripteur de chaque message pour voir s'il s'agit d'un message de demande ; si ce n'est pas le cas, le programme supprime le message et affiche un message d'avertissement.

Pour chaque message de demande supprimé de la file d'attente des demandes, le programme lit le nom de la file d'attente (que nous appellerons la *file d'attente cible*) contenues dans les données et ouvre cette file d'attente à l'aide de l'appel MQOPEN avec l'option MQOOO_INQ. Le programme utilise ensuite l'appel MQINQ pour consulter les valeurs des attributs *InhibitGet*, **CurrentQDepth**et **OpenInputCount** de la file d'attente cible.

Si l'appel MQINQ aboutit, le programme utilise l'appel MQPUT1 pour placer un message de réponse dans la file d'attente de réponse. Ce message contient les valeurs des trois attributs.

Si l'appel MQOPEN ou MQINQ échoue, le programme utilise l'appel MQPUT1 pour placer un message de rapport dans la file d'attente de réponse. Dans la zone *Feedback* du descripteur de message de ce message de rapport, le code anomalie est renvoyé par l'appel MQOPEN ou MQINQ, selon celui qui a échoué.

Après l'appel MQINQ, le programme ferme la file d'attente cible à l'aide de l'appel MQCLOSE.

Lorsqu'il ne reste aucun message dans la file d'attente des demandes, le programme ferme cette file d'attente et se déconnecte du gestionnaire de files d'attente.

Exemple de programme d'interrogation des propriétés d'un descripteur de message

AMQSIQMA est un exemple de programme C permettant d'interroger les propriétés d'un descripteur de message à partir d'une file d'attente de messages et est un exemple d'utilisation de l'appel API MQINQMP.

Cet exemple crée un descripteur de message et le place dans la zone MsgHandle de la structure MQGMO. L'exemple obtient ensuite un message et demande et imprime toutes les propriétés avec lesquelles le descripteur de message a été rempli.

```
C:\Program Files\IBM\MQ\tools\c\Samples\Bin >amqsiqm Q QM1
Sample AMQSIQMA start
property name MyProp value MyValue
message text Hello world!
Sample AMQSIQMA end
```

Exemples de programmes de publication / abonnement

Les exemples de programmes de publication / abonnement illustrent l'utilisation des fonctions de publication et d'abonnement dans IBM MQ.

Il existe trois exemples de programmes en langage C qui illustrent comment programmer l'interface de publication / abonnement IBM MQ . Il existe des exemples C qui utilisent des interfaces plus anciennes et des exemples Java . Les exemples Java utilisent l'interface de publication / abonnement IBM MQ dans com.ibm.mq.jar et l'interface de publication / abonnement JMS dans com.ibm.mqjms. Les exemples JMS ne sont pas traités dans cette rubrique.

C

Recherchez l'exemple de diffuseur de publications amqspub dans le dossier des exemples C . Exécutez-le avec le nom de rubrique de votre choix comme premier paramètre, suivi d'un nom de gestionnaire de files d'attente facultatif. Par exemple, `amqspub mytopic QM3` . Il existe également une version client appelée amqspubc. Si vous choisissez d'exécuter la version du client, consultez d'abord «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098 pour plus de détails.

Le diffuseur de publications se connecte au gestionnaire de files d'attente par défaut et répond avec la sortie `target topic is mytopic` . Chaque ligne que vous entrez dans cette fenêtre à partir de maintenant est publiée dans `mytopic` .

Ouvrez une autre fenêtre de commande dans le même répertoire et exécutez le programme d'abonné, amqssub, en indiquant le même nom de rubrique et un nom de gestionnaire de files d'attente facultatif. Par exemple, `amqssub mytopic QM3` .

L'abonné répond avec la sortie, `Calling MQGET : 30 seconds wait time` . A partir de maintenant, les lignes que vous entrez dans le diffuseur de publications apparaissent dans la sortie de l'abonné.

Démarrez un autre abonné dans une autre fenêtre de commande et observez les deux abonnés recevoir des publications.

Pour une documentation complète des paramètres, y compris les options de définition, reportez-vous à l'exemple de code source. Les valeurs de la zone des options d'abonné sont décrites dans la rubrique suivante: Options (MQLONG).

Il existe un autre exemple d'abonné amqssbx, qui offre des options d'abonnement supplémentaires en tant que commutateurs de ligne de commande.

Entrez `amqssbx -d mysub -t mytopic -k` pour appeler l'abonné à l'aide d'abonnements durables qui sont conservés après l'arrêt de l'abonné.

Testez l'abonnement en publiant un autre élément à l'aide du diffuseur de publications. Attendez 30 secondes que l'abonné s'arrête. Publiez d'autres éléments sous la même rubrique. Redémarrez l'abonné. Le dernier élément publié alors que l'abonné n'était pas en cours d'exécution est affiché par l'abonné dès qu'il est redémarré.

C existant

Il existe un ensemble supplémentaire d'échantillons C qui illustrent les commandes mises en file d'attente. Certains de ces exemples ont été fournis à l'origine dans le cadre du Supportpac MQ0C . Les fonctionnalités des exemples sont entièrement prises en charge, pour des raisons de compatibilité.

Nous vous déconseillons d'utiliser l'interface de commande en file d'attente. Il est beaucoup plus complexe que l'API de publication / abonnement, et il n'y a aucune raison fonctionnelle impérieuse de programmer des commandes en file d'attente complexes. Toutefois, vous pouvez trouver l'approche mise en file d'attente plus appropriée, peut-être parce que vous utilisez déjà l'interface, ou parce que votre environnement de programmation facilite la génération d'un message complexe et l'appel d'un MQPUT générique, plutôt que la construction d'appels différents à MQSUB.

Les exemples supplémentaires se trouvent dans le sous-répertoire pubsub du dossier `samples` .

Il existe six types d'exemple répertoriés dans le Tableau 163, à la page 1130.

Tableau 163. Catégories d'exemples de programmes C de publication / abonnement existants

Catégorie	Programmes	Commentaires
RFH1	amqssr1a.c amqspr1a.c	Exemple de publication / abonnement simple généré à l'aide de messages au format RFH1 .

Tableau 163. Catégories d'exemples de programmes C de publication / abonnement existants (suite)		
Catégorie	**Programmes**	**Commentaires**
RFH2	`amqssr2a.c` `amqspr2a.c`	Exemple simple de publication / abonnement généré à l'aide de messages au format RFH2 .
Exemples MQAI	`amqsppca.c` `amqsspca.c`	Exemple de publication / abonnement simple généré à l'aide de commandes PCF et de l'interface de commande MQAI.
MA0C Service de résultats utilisant RFH1	`amqsgama.c` `amqsresa.c`	Service de résultats généré à l'aide des en-têtes RFH1 1. Requiert les files d'attente définies dans `amqsgama.tst` et `amqsresa.tst` 2. `amqsresa` doit être démarré avant `amqsgama`
MA0C Service de résultats utilisant RFH2	`amqsgr2a.c` `amqsrr2a.c`	Service de résultats généré à l'aide des en-têtes RFH2 1. Requiert les files d'attente définies dans `amqsgama.tst` et `amqsresa.tst` 2. `amqsresa` doit être démarré avant `amqsgama`
Exemple de publication / abonnement d'exit de routage	`amqspsra.c`	Montre comment modifier la destination de file d'attente ou de gestionnaire de files d'attente pour un message de publication / abonnement dans un exit de routage.

Exemple de programme pour Java

L' Java exemple `MQPubSubApiSample.java` combine le diffuseur de publications et les abonnés dans un seul programme. Ses fichiers de classe source et compilés se trouvent dans le dossier des exemples `wmqjava` .

Si vous choisissez de l'exécuter en mode client, consultez d'abord «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098 pour plus de détails.

Exécutez l'exemple à partir de la ligne de commande à l'aide de la commande Java , si un environnement Java est configuré. Vous pouvez également exécuter l'exemple à partir de l'espace de travail IBM MQ Explorer Eclipse dans lequel un plan de travail de programmation Java est déjà configuré.

Vous devrez peut-être modifier certaines des propriétés de l'exemple de programme pour l'exécuter. Pour ce faire, vous devez fournir des paramètres à la machine virtuelle Java ou éditer la source.

Les instructions de la rubrique «Exécution de l'exemple MQPubSubApiSample Java», à la page 1131 montrent comment exécuter l'exemple à partir de l'espace de travail Eclipse .

Exécution de l'exemple MQPubSubApiSample Java
Comment exécuter MQPubSubApiSample à l'aide des outils de développement Java à partir de la plateforme Eclipse .

Avant de commencer

Ouvrez le plan de travail Eclipse . Créez un répertoire d'espace de travail et sélectionnez-le. Fermez la fenêtre de bienvenue.

Suivez les étapes de la rubrique «Configuration d'un gestionnaire de files d'attente pour l'acceptation des connexions client sur Multiplatforms», à la page 1098 avant de l'exécuter en tant que client.

Pourquoi et quand exécuter cette tâche

L'exemple de programme de publication / abonnement Java est un programme IBM MQ MQI client Java . L'exemple s'exécute sans modification à l'aide d'un gestionnaire de files d'attente par défaut à l'écoute sur le port 1414. La tâche décrit ce cas simple et indique en termes généraux comment fournir des paramètres et modifier l'exemple pour l'adapter à différentes configurations IBM MQ . L'exemple est illustré sous Windows. Les chemins d'accès aux fichiers diffèrent sur les autres plateformes.

Procédure

1. Importation des exemples de programme Java
   a) Dans le plan de travail, cliquez sur **Fenêtre** > **Ouvrir la perspective** > **Autre** > **Java** et cliquez sur **OK**.
   b) Accédez à la vue **Explorateur de packages** .
   c) Cliquez avec le bouton droit de la souris dans l'espace blanc de la vue **Explorateur de packages** . Cliquez sur **Nouveau** > **Java projet**.
   d) Dans la zone **`Project name`** , entrez `MQ Java Samples`. Cliquez sur **Suivant**.
   e) Dans le panneau **`Java Settings`** , accédez à l'onglet **Bibliothèques** .
   f) Cliquez sur **Ajouter des fichiers JAR externes**.
   g) Accédez à *MQ_INSTALLATION_PATH* `\java\lib` où *MQ_INSTALLATION_PATH* est le dossier d'installation de IBM MQ et sélectionnez `com.ibm.mq.jar` et `com.ibm.mq.jmqi.jar`
   h) Cliquez sur **Ouvrir** > **Terminer.**
   i) Cliquez avec le bouton droit de la souris sur `src` dans la vue **Explorateur de packages** .
   j) Sélectionnez **Importer ...** > **Général** > **Système de fichiers** > **Suivant** > **Parcourir**... et accédez au chemin *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ .
   k) Dans le panneau **Importer** , Figure 130, à la page 1133, cliquez sur `samples` (ne cochez pas la case).
   l) Sélectionnez `MQPubSubApiSample.java`. La zone **`Into folder`** doit contenir `MQ Java Samples/src`. Cliquez sur **Terminer**.

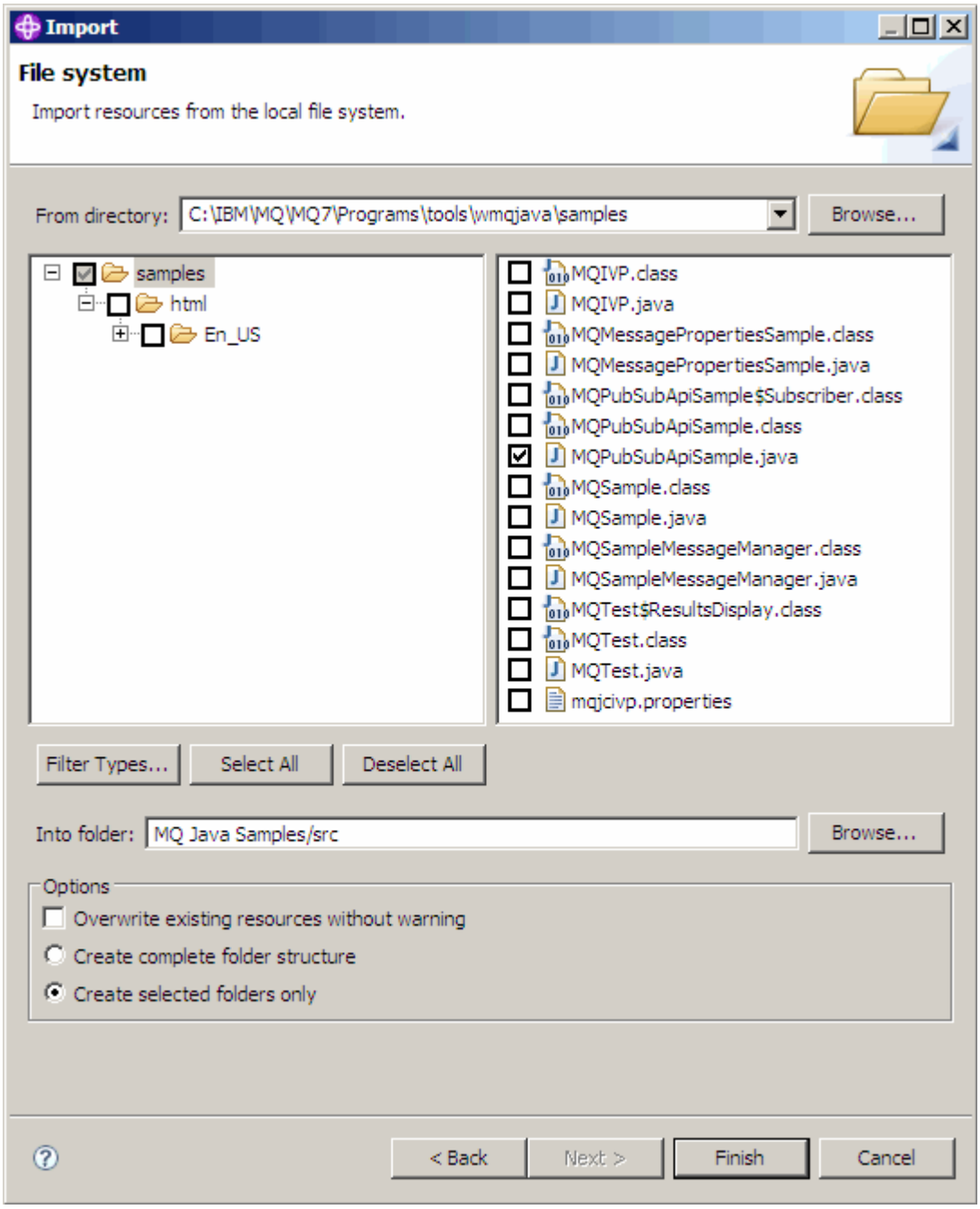


Figure 130. Importation du système de fichiers

2. Exécutez l'exemple de programme de publication / abonnement.

   Il existe deux façons d'exécuter le programme, selon que vous devez ou non modifier les paramètres par défaut.

   - Le premier choix consiste à exécuter le programme sans apporter de modifications:
     - Dans le menu principal de l'espace de travail, développez le dossier `src` . Cliquez avec le bouton droit de la souris sur **MQPubSubApiSample.java Run-as** > **1. Java Application**
   - La deuxième option exécute le programme avec des paramètres ou avec du code source modifié pour votre environnement:
     - Ouvrez `MQPubSubApiSample.java` et étudiez le constructeur `MQPubSubApiSample` .

– Modifiez les attributs du programme.

  Ces attributs sont modifiables à l'aide du commutateur JVM -D ou en fournissant une valeur par défaut pour la propriété System pen éditant le code source.

  - topicObject
  - QueueManagerName
  - subscriberCount

  Ces attributs ne sont modifiables qu'en éditant le code source dans le constructeur.

  - nom d'hôte
  - port
  - canal

  Pour définir des opérandes System p, codez une valeur par défaut dans le manipulateur, par exemple:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  Vous pouvez également fournir le paramètre à la machine virtuelle Java à l'aide de l'option `-D` , comme indiqué dans les étapes suivantes:

a. Copiez le nom complet de la propriété System.Property que vous souhaitez définir, par exemple: `com.ibm.mq.pubSubSample.queueManagerName`.
b. Dans l'espace de travail, cliquez avec le bouton droit de la souris sur **Exécuter** > **Ouvrir la boîte de dialogue Exécuter**. Cliquez deux fois sur Java Application dans **Créer, gérer et exécuter des applications** et cliquez sur l'onglet **(x) = Arguments** .
c. Dans le panneau **VM arguments:** , entrez `-D` et collez le nom System.property , `com.ibm.mq.pubSubSample.queueManagerName`, suivi de `=QM3`. Cliquez sur **Appliquer** > **Exécuter**.
d. Ajoutez d'autres arguments sous forme de liste séparée par des virgules ou sous forme de lignes supplémentaires dans le panneau, sans séparateurs de virgules.

   Par exemple : `-Dcom.ibm.mq.pubSubSample.queueManagerName=QM3,` `-Dcom.ibm.mq.pubSubSample.subscriberCount=6`.

Exemple de programme d'exit de publication

AMQSPSE0 est un exemple de programme C d'un exit permettant d'intercepter une publication avant qu'elle ne soit distribuée à un abonné. L'exit peut ensuite, par exemple, modifier les en-têtes de message, la charge ou la destination, ou empêcher la publication du message sur un abonné.

Pour exécuter l'exemple, effectuez les tâches suivantes:

1. Configurez le gestionnaire de files d'attente:
   - Linux AIX Sur les systèmes AIX and Linux , ajoutez une strophe similaire à la suivante dans le fichier `qm.ini` :

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     où le module est *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé.
   - Windows Sous Windows , définissez les attributs équivalents dans le registre.
2. Assurez-vous que le module est accessible à IBM MQ.
3. Redémarrez le gestionnaire de files d'attente pour récupérer la configuration.

4. Dans le processus d'application à tracer, indiquez où les fichiers de trace doivent être écrits. Exemple :
   - Linux AIX Sur les systèmes AIX and Linux , vérifiez que le répertoire `/var/mqm/trace` existe et exportez la variable d'environnement **MQPSE_TRACE_LOGFILE** :

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows Sous Windows, vérifiez que le répertoire `C:\temp` existe et définissez la variable d'environnement **MQPSE_TRACE_LOGFILE** :

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Exemples de programmes Put

Les exemples de programme d'insertion placent des messages dans une file d'attente à l'aide de l'appel MQPUT.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Conception de l'exemple de programme Put

Le programme utilise l'appel MQOPEN avec l'option MQOO_OUTPUT pour ouvrir la file d'attente cible afin d'insérer des messages.

S'il ne parvient pas à ouvrir la file d'attente, le programme génère un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN. Pour simplifier le programme, sur cet appel MQI et sur les appels MQI suivants, le programme utilise des valeurs par défaut pour de nombreuses options.

Pour chaque ligne d'entrée, le programme lit le texte dans une mémoire tampon et utilise l'appel MQPUT pour créer un message de datagramme contenant le texte de cette ligne. Le programme se poursuit jusqu'à ce qu'il atteigne la fin de l'entrée ou que l'appel MQPUT échoue. Si le programme atteint la fin de l'entrée, il ferme la file d'attente à l'aide de l'appel MQCLOSE.

Exécution des exemples de programme Put

Exécution des exemples amqsput et amqsputc

ALW

L'exemple **amqqsput** est le programme d'insertion de messages à l'aide de liaisons locales et l'exemple **amqsputc** est le programme d'insertion de messages à l'aide de liaisons client. Ces programmes prennent chacun les paramètres positionnels suivants:

1. Nom de la file d'attente cible (obligatoire)
2. Nom du gestionnaire de files d'attente (facultatif)

   Si aucun gestionnaire de files d'attente n'est spécifié, **amqsput** se connecte au gestionnaire de files d'attente par défaut et **amqsputc** se connecte au gestionnaire de files d'attente identifié par la variable d'environnement *MQSERVER* ou le fichier de définition de canal du client.
3. Les options d'ouverture (facultatif)

   Si les options d'ouverture ne sont pas spécifiées, l'exemple utilise la valeur 8208, qui est la combinaison de ces deux options:

   - MQOO_SORTIE
   - MQOO_FAIL_IF_QUIESCING
4. Options de fermeture (facultatif)

   Si les options de fermeture ne sont pas spécifiées, l'exemple utilise la valeur 0, qui est MQCO_NONE.

5. Nom du gestionnaire de files d'attente cible (facultatif)

   Si aucun gestionnaire de files d'attente cible n'est spécifié, la zone `ObjectQMgrName` du MQOD reste vide.
6. Nom de la file d'attente dynamique (facultatif)

   Si aucun nom de file d'attente dynamique n'est spécifié, la zone `DynamicQName` du MQOD reste vide.

Utilisez les variables d'environnement suivantes pour fournir les données d'identification utilisées pour l'authentification auprès du gestionnaire de files d'attente:

ID_UTILISATEUR_MQM
Définissez l'ID utilisateur à utiliser pour l'authentification de connexion, si vous souhaitez utiliser un ID utilisateur et un mot de passe pour vous authentifier auprès du gestionnaire de files d'attente. Le programme demande le mot de passe qui doit accompagner l'ID utilisateur.

V 9.4.0 Linux AIX **MQSAMP_TOKEN**
Indiquez une valeur non vide si vous souhaitez fournir un jeton d'authentification pour l'authentification auprès du gestionnaire de files d'attente. Le programme demande le jeton d'authentification. Les jetons d'authentification ne peuvent être utilisés que par l'exemple **amqsputc** qui utilise des liaisons client.

Pour exécuter ces programmes, entrez l'une des commandes suivantes:

- `amqsput` *`myqueue qmanagername`*
- `amqsputc` *`myqueue qmanagername`*

où *monfile d'attente* est le nom de la file d'attente dans laquelle les messages vont être insérés et *qmanagername* est le gestionnaire de files d'attente qui possède *monfile d'attente*.

Exécution de l'exemple amq0put

ALW

La version COBOL ne comporte aucun paramètre. Il se connecte au gestionnaire de files d'attente par défaut et lorsque vous l'exécutez, vous êtes invité à:

```
Please enter the name of the target queue
```

Il prend l'entrée de StdIn et ajoute chaque ligne d'entrée à la file d'attente cible. Une ligne vide indique qu'il n'y a plus de données.

Exécution de l'exemple C AMQSPUT4 (IBM i)

IBM i

Le programme C AMQSPUT4, disponible uniquement pour la plateforme IBM i , crée des messages en lisant les données d'un membre d'un fichier source.

Vous devez indiquer le nom du fichier en tant que paramètre lorsque vous démarrez le programme. La structure du fichier doit être:

```
queue name
text of message 1
text of message 2
:
text of message n
blank line
```

Un exemple d'entrée pour les exemples d'insertion est fourni dans le fichier QMQMSAMP de la bibliothèque AMQSDATA member PUT.

Remarque : N'oubliez pas que les noms de file d'attente sont sensibles à la casse. Toutes les files d'attente créées par le programme de création d'exemple de fichier AMQSAMP4 ont des noms créés en majuscules.

Le programme C place les messages dans la file d'attente nommée sur la première ligne du fichier ; vous pouvez utiliser la file d'attente fournie SYSTEM.SAMPLE.LOCAL. Le programme place le texte de chacune des lignes suivantes du fichier dans des messages de datagramme distincts et s'arrête lorsqu'il lit une ligne vide à la fin du fichier.

A l'aide de l'exemple de fichier de données, la commande est la suivante:

```
CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
```

Exécution de l'exemple COBOL AMQ0PUT4 (IBM i)

IBM i

Le programme COBOL AMQ0PUT4, disponible uniquement sur la plateforme IBM i , crée des messages en acceptant des données à partir du clavier.

Pour démarrer le programme, appelez le programme et indiquez le nom de votre file d'attente cible comme paramètre de programme. Le programme accepte les entrées du clavier dans une mémoire tampon et crée un message de datagramme pour chaque ligne de texte. Le programme s'arrête lorsque vous entrez une ligne vide au clavier.

Exemples de programmes de message de référence

Les exemples de message de référence permettent de transférer un objet volumineux d'un noeud à un autre (généralement sur des systèmes différents) sans qu'il soit nécessaire que l'objet soit stocké dans des files d'attente IBM MQ sur les noeuds source ou de destination.

Un ensemble d'exemples de programmes est fourni pour illustrer la façon dont les messages de référence peuvent être placés dans une file d'attente, reçus par des exits de message et extraits d'une file d'attente. Les exemples de programme utilisent des messages de référence pour déplacer des fichiers. Si vous souhaitez déplacer d'autres objets, tels que des bases de données, ou si vous souhaitez effectuer des contrôles de sécurité, définissez votre propre exit, basé sur l'exemple, amqsxrm.

La version de l'exemple de programme d'exit de message de référence à utiliser dépend de la plateforme sur laquelle le canal s'exécute:

- Sur toutes les plateformes, utilisez amqsxrma à la fin de l'envoi.
- Utilisez amqsxrma à l'extrémité réceptrice si le récepteur s'exécute sous une plateforme autre que IBM i.
- IBM i Si le récepteur s'exécute sous IBM i, utilisez amqsxrm4.

.

IBM i *Remarques pour les utilisateurs IBM i*

Pour recevoir un message de référence à l'aide de l'exemple d'exit de message, spécifiez un fichier dans le système de fichiers racine d'IFS ou dans un sous-répertoire de sorte qu'un fichier STREAM puisse être créé.

L'exemple d'exit de message sur IBM i crée le fichier, convertit les données en EBCDIC et définit la page de codes sur votre page de codes système. Vous pouvez ensuite copier ce fichier dans QSYS.LIB à l'aide de la commande CPYFRMSTMF. Exemple :

```
CPYFRMSTMF FROMSTMF('JANEP/TEST.TXT')
TOMBR('qsys.lib.janep.lib/test.fie/test.mbr') MBROPT(*REPLACE)
CVTDTA(*NONE)
```

La commande CPYFRMSTMF ne crée pas le fichier. Vous devez le créer avant d'exécuter cette commande.

Si vous envoyez un fichier à partir de QSYS.LIB, aucune modification n'est requise pour les exemples. Pour tout autre système de fichiers, vérifiez que le CCSID spécifié dans la zone CodedCharSetId de la structure MQRMH correspond aux données non formatées que vous envoyez.

Lorsque vous utilisez le système de fichiers intégré, créez des modules de programme avec l'option SYSIFCOPT (*IFSIO) définie. Si vous souhaitez déplacer des fichiers d'enregistrement de base de données ou de longueur fixe, définissez votre propre exit en fonction de l'exemple fourni AMQSXRM4.

La méthode recommandée pour transférer un fichier de base de données consiste à le convertir en structure IFS, à l'aide de la commande CPYTOSTMF, puis à envoyer le message de référence joint au fichier IFS. Si vous choisissez de transférer un fichier de base de données en y faisant référence depuis IFS, mais que vous ne le convertissez pas en structure IFS, vous devez spécifier le nom du membre. L'intégrité des données n'est pas garantie si vous choisissez cette méthode.

Exécution des exemples de message de référence

Utilisez cet exemple pour savoir comment exécuter l'exemple d'application de message de référence AMQSPRM sur AIX, Linux, and Windowsou AMQSPRMA sur IBM i. L'exemple montre comment les messages de référence peuvent être placés dans une file d'attente, reçus par des exits de message et extraits d'une file d'attente.

Les exemples de message de référence s'exécutent comme suit:

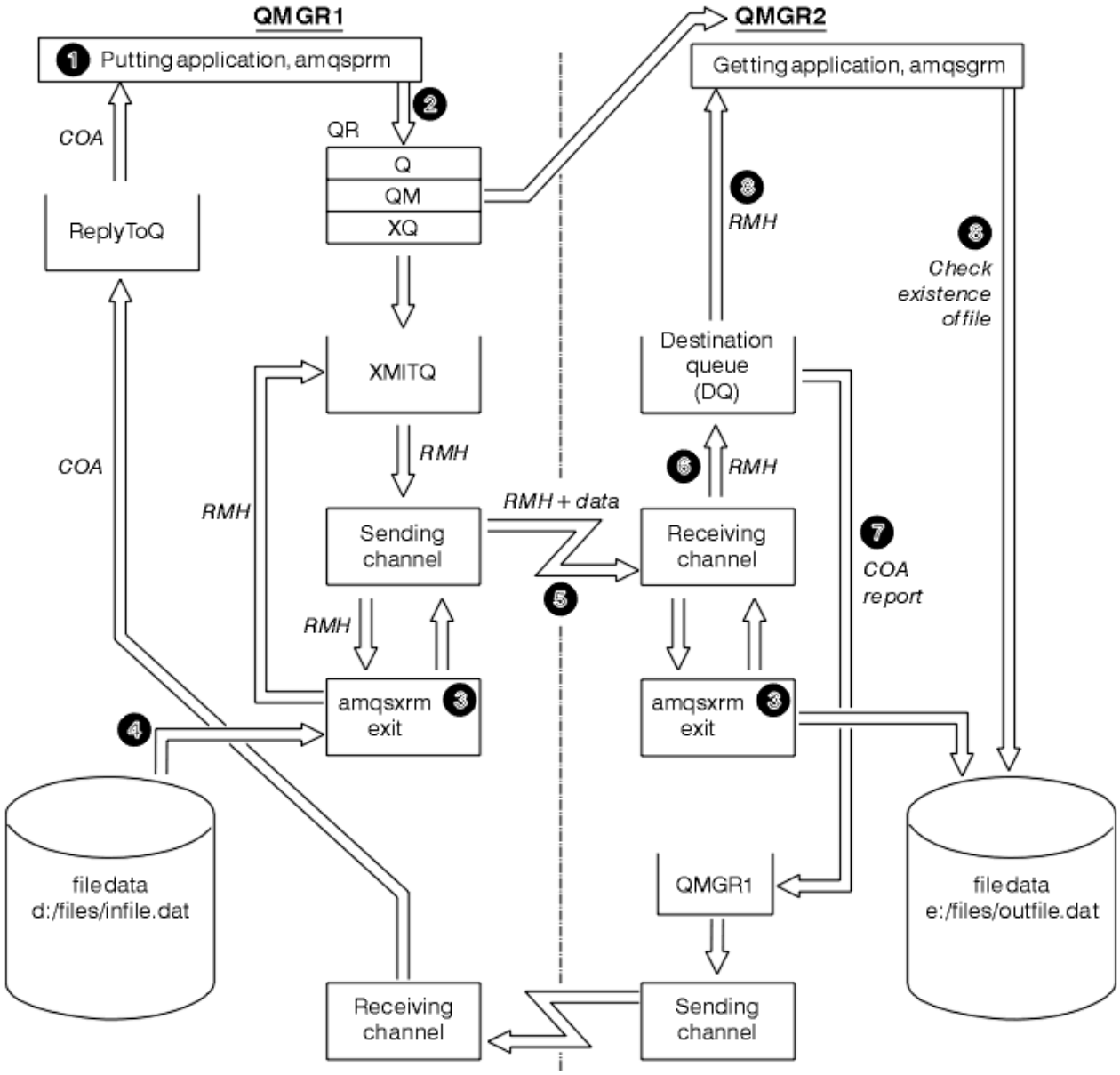


Figure 131. Exécution des exemples de message de référence

1. Configurez l'environnement pour démarrer les programmes d'écoute, les canaux et les moniteurs de déclenchement, et définissez vos canaux et vos files d'attente.

   Pour décrire la configuration du message de référence, cet exemple fait référence à la machine émettrice en tant que MACHINE1 avec un gestionnaire de files d'attente appelé QMGR1 et à la machine réceptrice en tant que MACHINE2 avec un gestionnaire de files d'attente appelé QMGR2.

   Remarque : Les définitions suivantes permettent de générer un message de référence pour envoyer un fichier avec un type d'objet FLATFILE du gestionnaire de files d'attente QMGR1 à QMGR2 et de recréer le fichier comme défini dans l'appel à AMQSPRM (ou AMQSPRMA sous IBM i). Le message de référence (y compris les données de fichier) est envoyé à l'aide du canal CHL1 et de la file d'attente de transmission XMITQ et placé dans la file d'attente DQ. Les rapports d'exception et COA sont renvoyés à QMGR1 à l'aide du canal REPORT et de la file d'attente de transmission QMGR1.

   L'application qui reçoit le message de référence (AMQSGRM ou AMQSGRMA sur IBM i) est déclenchée à l'aide de la file d'attente d'initialisation INITQ et du processus PROC. Vérifiez que les zones CONNAME sont définies correctement et que la zone MSGEXIT reflète votre structure de répertoire, en fonction du type de machine et de l'emplacement d'installation du produit IBM MQ .

IBM i Les définitions MQSC ont utilisé un style AIX pour définir les exits. Par conséquent, si vous utilisez MQSC sous IBM i, vous devez les modifier en conséquence. Il est important de noter que les données de message FLATFILE sont sensibles à la casse et que l'exemple ne fonctionnera pas sauf s'il est en majuscules.

Sur la machine MACHINE1, le gestionnaire de files d'attente QMGR1

Syntaxe MQSC

```
define chl(chl1) chltype(sdr) trptype(tcp) conname('machine2') xmitq(xmitq)
msgdata(FLATFILE) msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)
')

define ql(xmitq) usage(xmitq)

define chl(report) chltype(rcvr) trptype(tcp) replace

define qr(qr) rname(dq) rqmname(qmgr2) xmitq(xmitq) replace
```

IBM i **Syntaxe de commande IBM i**

Remarque : Si vous ne spécifiez pas de nom de gestionnaire de files d'attente, le système utilise le gestionnaire de files d'attente par défaut.

```
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)
```

Sur la machine MACHINE2, gestionnaire de files d'attente QMGR2

Syntaxe MQSC

```
define chl(chl1) chltype(rcvr) trptype(tcp)
msgexit('/usr/lpp/mqm/samp/bin/amqsxrm(MsgExit)')
        msgdata(flatfile)

define chl(report) chltype(sdr) trptype(tcp) conname('MACHINE1')
        xmitq(qmgr1)

define ql(initq)

define ql(qmgr1) usage(xmitq)

define pro(proc) applicid('/usr/lpp/mqm/samp/bin/amqsgrm')

define ql(dq) initq(initq) process(proc) trigger trigtype(first)
```

IBM i **IBM i syntaxe de commande**

Remarque : IBM i Sur IBM i, si vous ne spécifiez pas de nom de gestionnaire de files d'attente, le système utilise le gestionnaire de files d'attente par défaut.

```
CRTMQMCHL CHLNAME(CHL1) CHLTYPE(*RCVR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
MSGEXIT(QMQM/AMQSXRM4) MSGUSRDATA(FLATFILE)

CRTMQMCHL CHLNAME(REPORT) CHLTYPE(*SDR) MQMNAME(QMGR2) +
REPLACE(*YES) TRPTYPE(*TCP) +
```

```
CONNAME('MACHINE1(60500)') TMQNAME(QMGR1)

CRTMQMQ  QNAME(INITQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*NORMAL)

CRTMQMQ  QNAME(QMGR1) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) USAGE(*TMQ)

CRTMQMPRC PRCNAME(PROC) MQMNAME(QMGR2) REPLACE(*YES) +
APPID('QMQM/AMQSGRM4')

CRTMQMQ  QNAME(DQ) QTYPE(*LCL) MQMNAME(QMGR2) +
REPLACE(*YES) PRCNAME(PROC) TRGENBL(*YES) +
INITQNAME(INITQ)
```

2. Une fois les objets IBM MQ créés:
   a. Si applicable à la plateforme, démarrez le programme d'écoute pour les gestionnaires de files d'attente d'envoi et de réception
   b. Démarrez les canaux CHL1 et REPORT
   c. Sur le gestionnaire de files d'attente de réception, démarrez le moniteur de déclenchement pour la file d'attente d'initialisation INITQ
3. Appelez l'exemple de programme de message de référence d'insertion AMQSPRM (AMQSPRMA sous IBM i) à partir de la ligne de commande à l'aide des paramètres suivants:

   -m
   Nom du gestionnaire de files d'attente local ; par défaut, il s'agit du gestionnaire de files d'attente par défaut

   -i
   Nom et emplacement du fichier source

   -o
   Nom et emplacement du fichier de destination

   -q
   Nom de la file d'attente

   -g
   Nom du gestionnaire de files d'attente dans lequel la file d'attente, définie dans le paramètre -q, existe. La valeur par défaut est le gestionnaire de files d'attente spécifié dans le paramètre -m.

   -t
   Type d'objet

   -w
   Intervalle d'attente, c'est-à-dire le temps d'attente des rapports d'exception et COA du gestionnaire de files d'attente de réception

   Par exemple, pour utiliser l'exemple avec les objets définis précédemment, vous devez utiliser les paramètres suivants:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   L'augmentation du temps d'attente permet à un fichier volumineux d'être envoyé sur un réseau avant que le programme qui place les messages n'arrive à expiration.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

Utilisateurs IBM i : 

Sous IBM i, procédez comme suit:

   a. Utilisez la commande suivante :

```
CALL    PGM(QMQM/AMQSPRM4) PARM('-mQMGR1' +
'-i/refmsgs/rmsg1' +
```

```
'-o/refmsgs/rmsgx' '-qQR' +
'-gQMGR1' '-tFLATFILE' '-w15')
```

Cela suppose que le fichier d'origine `rmsg1` se trouve dans le répertoire IFS `/refmsgs` et que vous souhaitez que le fichier de destination soit `rmsgx` dans le répertoire IFS `/refmsgs` sur le système cible.

b. Créez votre propre répertoire à l'aide de la commande CRTDIR au lieu d'utiliser le répertoire racine.

c. Lorsque vous appelez le programme qui insère des données, n'oubliez pas que le nom du fichier de sortie doit refléter la convention de dénomination IFS ; par exemple, /TEST/FILENAME crée un fichier appelé FILENAME dans le répertoire TEST.

Remarque :

IBM i Sous IBM i, vous pouvez utiliser une barre oblique (/) ou un tiret (-) lorsque vous spécifiez des paramètres. Exemple :

```
amqsprm /i d:\files\infile.dat /o e:\files\outfile.dat /q QR
/m QMGR1 /w 30 /t FLATFILE
```

Linux AIX Pour les plateformes AIX and Linux , vous devez utiliser deux barres obliques inversées (\\) au lieu d'une pour indiquer le répertoire de fichiers de destination. Par conséquent, la commande **amqsprm** se présente comme suit:

```
amqsprm -i /files/infile.dat -o e:\\files\\outfile.dat -q QR
-m QMGR1 -w 30 -t FLATFILE
```

L'exécution du programme d'insertion de message de référence effectue les opérations suivantes:

- Le message de référence est inséré dans la file d'attente QR du gestionnaire de files d'attente QMGR1.
- Le fichier source et le chemin d'accès sont `d:\files\infile.dat` et existent sur le système où l'exemple de commande est émis.
- Si la file d'attente QR est une file d'attente éloignée, le message de référence est envoyé à un autre gestionnaire de files d'attente, sur un autre système, où un fichier est créé avec le nom et le chemin `e:\files\outfile.dat`. Le contenu de ce fichier est identique à celui du fichier source.
- amqsprm attend pendant 30 secondes un rapport COA provenant du gestionnaire de files d'attente de destination.
- Le type d'objet étant `flatfile`, le canal utilisé pour déplacer des messages de la file d'attente QR doit le spécifier dans la zone *MsgData* .

4. Lorsque vous définissez vos canaux, sélectionnez l'exit de message aux extrémités émettrice et réceptrice pour qu'il soit amqsxrm.

Windows Il est défini sur Windows comme suit:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Linux AIX Il est défini sur AIX and Linux comme suit:

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Si vous indiquez un nom de chemin, indiquez le nom complet. Si vous omettez le nom de chemin, il est supposé que le programme se trouve dans le chemin indiqué dans le fichier `qm.ini` (ou, sous IBM MQ for Windows, dans le registre).

5. L'exit de canal lit l'en-tête du message de référence et trouve le fichier auquel il fait référence.
6. L'exit de canal peut ensuite segmenter le fichier avant de l'envoyer sur le canal avec l'en-tête.

Linux AIX Sous AIX and Linux, remplacez le propriétaire de groupe du répertoire cible par'mqm'afin que l'exemple d'exit de message puisse créer le fichier dans ce répertoire. Modifiez également les droits d'accès du répertoire cible pour permettre aux membres du groupe mqm d'y écrire des données. Les données de fichier ne sont pas stockées dans les files d'attente IBM MQ .

7. Lorsque le dernier segment du fichier est traité par l'exit de message de réception, le message de référence est placé dans la file d'attente de destination spécifiée par amqsprm. Si cette file d'attente est déclenchée (c'est-à-dire si la définition spécifie les attributs de file d'attente **Trigger**, **InitQ**et **Process**), le programme spécifié par le paramètre PROC de la file d'attente de destination est déclenché. Le programme à déclencher doit être défini dans la zone ApplId de l'attribut **Process** .
8. Lorsque le message de référence atteint la file d'attente de destination (DQ), un rapport COA est renvoyé à l'application d'insertion (amqsprm).
9. L'exemple d'extraction de message de référence, amqsgrm, extrait les messages de la file d'attente spécifiée dans le message de déclenchement d'entrée et vérifie l'existence du fichier.

Conception de l'exemple Put Reference Message (amqsprma.c, AMQSPRM4)
Cette rubrique fournit une description détaillée d'un exemple de message de référence d'insertion.

Cet exemple crée un message de référence qui fait référence à un fichier et le place dans une file d'attente spécifiée:

1. L'exemple se connecte à un gestionnaire de files d'attente local à l'aide de MQCONN.
2. Il ouvre ensuite (MQOPEN) une file d'attente modèle utilisée pour recevoir des messages de rapport.
3. L'exemple génère un message de référence contenant les valeurs requises pour déplacer le fichier, par exemple, les noms de fichier source et de destination et le type d'objet. Par exemple, l'exemple fourni avec IBM MQ génère un message de référence pour envoyer le fichier `d:\x\file.in` de QMGR1 à QMGR2 et pour recréer le fichier en tant que `d:\y\file.out` à l'aide des paramètres suivants:

   ```
   amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
   ```

   Où QR est une définition de file d'attente éloignée qui fait référence à une file d'attente cible sur QMGR2.

   Remarque : Pour les plateformes AIX and Linux , utilisez deux barres obliques inversées (\\) au lieu d'une pour indiquer le répertoire de fichiers de destination. Par conséquent, la commande **amqsprm** se présente comme suit:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```
4. Le message de référence est inséré (sans données de fichier) dans la file d'attente spécifiée par le paramètre /q. S'il s'agit d'une file d'attente éloignée, le message est inséré dans la file d'attente de transmission correspondante.
5. L'exemple attend, pendant la durée spécifiée dans le paramètre /w (qui par défaut est de 15 secondes), les rapports COA qui, avec les rapports d'exception, sont renvoyés à la file d'attente dynamique créée sur le gestionnaire de files d'attente local (QMGR1).

Conception de l'exemple d'exit de message de référence (amqsxrma.c, AMQSXRM4)
Cet exemple reconnaît les messages de référence avec un type d'objet qui correspond au type d'objet dans la zone de données utilisateur de l'exit de message de la définition de canal.

Pour ces messages, les événements suivants se produisent:

- Au niveau du canal émetteur ou serveur, la longueur de données spécifiée est copiée à partir du décalage indiqué du fichier spécifié dans l'espace restant dans la mémoire tampon de l'agent après le message de référence. Si la fin du fichier n'est pas atteinte, le message de référence est replacé dans la file d'attente de transmission après la mise à jour de la zone *DataLogicalOffset* .
- Sur le canal demandeur ou récepteur, si la zone *DataLogicalOffset* est à zéro et que le fichier spécifié n'existe pas, il est créé. Les données qui suivent le message de référence sont ajoutées à la fin

du fichier spécifié. Si le message de référence n'est pas le dernier du fichier spécifié, il est supprimé. Sinon, il est renvoyé à l'exit de canal, sans les données ajoutées, pour être placé dans la file d'attente cible.

Pour les canaux émetteur et serveur, si la zone *DataLogicalLength* du message de référence d'entrée est égale à zéro, la partie restante du fichier, de *DataLogicalOffset* à la fin du fichier, doit être envoyée via le canal. S'il n'est pas égal à zéro, seule la longueur indiquée est envoyée.

Si une erreur se produit (par exemple, si l'exemple ne peut pas ouvrir un fichier), MQCXP. *ExitResponse* est défini sur MQXCC_SUPPRESS_FUNCTION de sorte que le message en cours de traitement soit inséré dans la file d'attente de rebut au lieu de passer à la file d'attente de destination. Un code retour est renvoyé dans MQCXP. *Feedback* et renvoyé à l'application qui a inséré le message dans la zone *Feedback* du descripteur de message d'un message de rapport. En effet, l'application d'insertion a demandé des rapports d'exception en définissant MQRO_EXCEPTION dans la zone *Report* du MQMD.

Si le codage ou le *CodedCharacterSetId* (CCSID) du message de référence est différent de celui du gestionnaire de files d'attente, le message de référence est converti en codage local et CCSID. Dans notre exemple, amqsprm, le format de l'objet est MQFMT_STRING. Par conséquent, amqsxrm convertit les données de l'objet en CCSID local à l'extrémité réceptrice avant que les données ne soient écrites dans le fichier.

Ne spécifiez pas le format du fichier transféré en tant que MQFMT_STRING si le fichier contient des caractères multi-octets (par exemple, DBCS ou Unicode). En effet, un caractère multioctet peut être fractionné lorsque le fichier est segmenté à la fin de l'envoi. Pour transférer et convertir un fichier de ce type, indiquez un format autre que MQFMT_STRING de sorte que l'exit de message de référence ne le convertisse pas et convertisse le fichier à la fin de la réception lorsque le transfert est terminé.

Compilation de l'exemple d'exit de message de référence
Pour compiler l'exemple d'exit de message de référence, utilisez la commande correspondant à la plateforme sur laquelle IBM MQ est installé.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM MQ est installé.

Pour compiler amqsxrma, utilisez les commandes suivantes:

sur AIX


```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

sur IBM i


```
CRTCMOD MODULE(MYLIB/AMQSXRMA) SRCFILE(QMQMSAMP/QCSRC)
TERASPACE(*YES *TSIFC)
```

Remarque :

1. Pour créer votre module afin qu'il utilise le système de fichiers IFS, ajoutez l'option SYSIFCOPT(*IFSIO)
2. Pour créer le programme à utiliser avec des canaux sans unités d'exécution, utilisez la commande suivante: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)
3. Pour créer le programme à utiliser avec les canaux à unités d'exécution, utilisez la commande suivante: CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM_R)

sur Linux

Linux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -Wl,-rpath= MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-lmqm_r
```

sur Windows

Windows IBM MQ fournit désormais la bibliothèque mqm avec des packages client ainsi que des packages serveur, de sorte que l'exemple suivant utilise `mqm.lib` à la place de `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Concepts associés

«Ecriture de programmes d'exit de canal», à la page 991
Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal.

Conception de l'exemple Obtenir un message de référence (amqsgrma.c, AMQSGRM4)
Cette rubrique explique la conception de l'exemple Obtenir un message de référence.

La logique du programme est la suivante:

1. L'exemple est déclenché et extrait les noms de file d'attente et de gestionnaire de files d'attente du message de déclenchement d'entrée.
2. Il se connecte ensuite au gestionnaire de files d'attente spécifié à l'aide de MQCONN et ouvre la file d'attente spécifiée à l'aide de MQOPEN.
3. L'exemple émet une commande MQGET avec un intervalle d'attente de 15 secondes dans une boucle pour extraire des messages de la file d'attente.
4. Si un message est un message de référence, l'exemple vérifie l'existence du fichier qui a été transféré.
5. Il ferme ensuite la file d'attente et se déconnecte du gestionnaire de files d'attente.

Exemples de programmes de demande

Les exemples de programmes de demande illustrent le traitement client-serveur. Les exemples sont les clients qui placent des messages de demande dans une file d'attente de serveur cible traitée par un programme serveur. Ils attendent que le programme serveur place un message de réponse dans une file d'attente de réponse.

Les exemples de demande placent une série de messages de demande dans la file d'attente du serveur cible à l'aide de l'appel MQPUT. Ces messages spécifient la file d'attente locale, SYSTEM.SAMPLE.REPLY comme file d'attente de réponse, qui peut être une file d'attente locale ou éloignée. Les programmes attendent les messages de réponse, puis les affichent. Les réponses sont envoyées uniquement si la file d'attente du serveur cible est traitée par une application serveur ou si une application est déclenchée à cette fin (les exemples de programme Inquire, Set et Echo sont conçus pour être déclenchés). L'exemple C attend 1 minute (l'exemple COBOL attend 5 minutes), que la première réponse arrive (pour permettre le déclenchement d'une application serveur) et 15 secondes pour les réponses suivantes, mais les deux exemples peuvent se terminer sans obtenir de réponse. Pour connaître les noms des exemples de programme de demande, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Exécution des exemples de programmes de demande

Exécution des exemples amqsreq0.c, amqsreq et amqsreqc

La version C du programme utilise trois paramètres:

1. Nom de la file d'attente du serveur cible (nécessaire)

2. Nom du gestionnaire de files d'attente (facultatif)
3. La file d'attente de réponses (facultatif)

Par exemple, entrez l'une des valeurs suivantes:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

où `myqueue` est le nom de la file d'attente du serveur cible, `qmanagername` est le nom du gestionnaire de files d'attente propriétaire de `myqueue`et `replyqueue` est le nom de la file d'attente de réponses.

Si vous omettez le nom du gestionnaire de files d'attente, il est supposé que le gestionnaire de files d'attente par défaut est propriétaire de la file d'attente. Si vous omettez le nom de la file d'attente de réponses, la file d'attente de réponses par défaut est fournie.

Exécution de l'exemple amq0req0.cbl

La version COBOL ne comporte aucun paramètre. Il se connecte au gestionnaire de files d'attente par défaut et lorsque vous l'exécutez, vous êtes invité à:

```
Please enter the name of the target server queue
```

Le programme extrait son entrée de StdIn et ajoute chaque ligne à la file d'attente du serveur cible, en prenant chaque ligne de texte comme contenu d'un message de demande. Le programme se termine lorsqu'une ligne nulle est lue.

Exécution de l'exemple AMQSREQ4

Le programme C crée des messages en prenant les données de stdin (le clavier) avec une heure de fin d'entrée à blanc. Le programme prend jusqu'à trois paramètres: le nom de la file d'attente cible (obligatoire), le nom du gestionnaire de files d'attente (facultatif) et le nom de la file d'attente de réponse (facultatif). Si aucun nom de gestionnaire de files d'attente n'est spécifié, le gestionnaire de files d'attente par défaut est utilisé. Si aucune file d'attente de réponse n'est spécifiée, SYSTEM.SAMPLE.REPLY est utilisée.

Voici un exemple d'appel de l'exemple de programme C, en spécifiant la file d'attente de réponse, mais en laissant la valeur par défaut du gestionnaire de files d'attente:

```
CALL PGM(QMQM/AMQSREQ4) PARM('SYSTEM.SAMPLE.LOCAL' '' 'SYSTEM.SAMPLE.REPLY')
```

Remarque : N'oubliez pas que les noms de file d'attente sont sensibles à la casse. Toutes les files d'attente créées par le programme de création d'exemple de fichier AMQSAMP4 ont des noms créés en majuscules.

Exécution de l'exemple AMQ0REQ4

Le programme COBOL crée des messages en acceptant les données du clavier. Pour démarrer le programme, appelez le programme et indiquez le nom de votre file d'attente cible en tant que paramètre. Le programme accepte les entrées du clavier dans une mémoire tampon et crée un message de demande pour chaque ligne de texte. Le programme s'arrête lorsque vous entrez une ligne vide au clavier.

Exécution de l'exemple Demande à l'aide du déclenchement

Si l'exemple est utilisé avec le déclenchement et l'un des exemples de programmes Inquire, Set ou Echo, la ligne d'entrée doit être le nom de la file d'attente à laquelle le programme déclenché doit accéder.

ALW *Exécution de l'exemple Demande à l'aide du déclenchement sous AIX, Linux, and Windows*

Sous AIX, Linux, and Windows, démarrez le programme de moniteur de déclenchement RUNMQTRM dans une session, puis démarrez le programme amqsreq dans une autre session.

Pour exécuter les exemples à l'aide du déclenchement:

1. Démarrez le programme de moniteur de déclenchement RUNMQTRM dans une session (file d'attente d'initialisation SYSTEM.SAMPLE.TRIGGER est disponible pour que vous puissiez l'utiliser).
2. Démarrez le programme amqsreq dans une autre session.
3. Vérifiez que vous avez défini une file d'attente de serveur cible.

   Les exemples de files d'attente que vous pouvez utiliser comme file d'attente du serveur cible pour l'exemple de demande d'insertion de messages sont les suivants:

   - SYSTEM.SAMPLE.INQ -pour l'exemple de programme Inquire
   - SYSTEM.SAMPLE.SET -pour l'exemple de programme Set
   - SYSTEM.SAMPLE.ECHO -pour l'exemple de programme Echo

   Ces files d'attente ont un type de déclencheur FIRST. Par conséquent, s'il existe déjà des messages dans les files d'attente avant l'exécution de l'exemple Demande, les applications serveur ne sont pas déclenchées par les messages que vous envoyez.
4. Vérifiez que vous avez défini une file d'attente pour l'exemple de programme Inquire, Set ou Echo à utiliser.

Cela signifie que le moniteur de déclenchement est prêt lorsque l'exemple de demande envoie un message.

Remarque : Les exemples de définitions de processus créés à l'aide de RUNMQSC et du fichier amqscos0.tst déclenchent les exemples C. Modifiez les définitions de processus dans amqscos0.tst et utilisez RUNMQSC avec ce fichier mis à jour pour utiliser les versions COBOL.

Figure 132, à la page 1148 montre comment utiliser les exemples Request et Inquire ensemble.

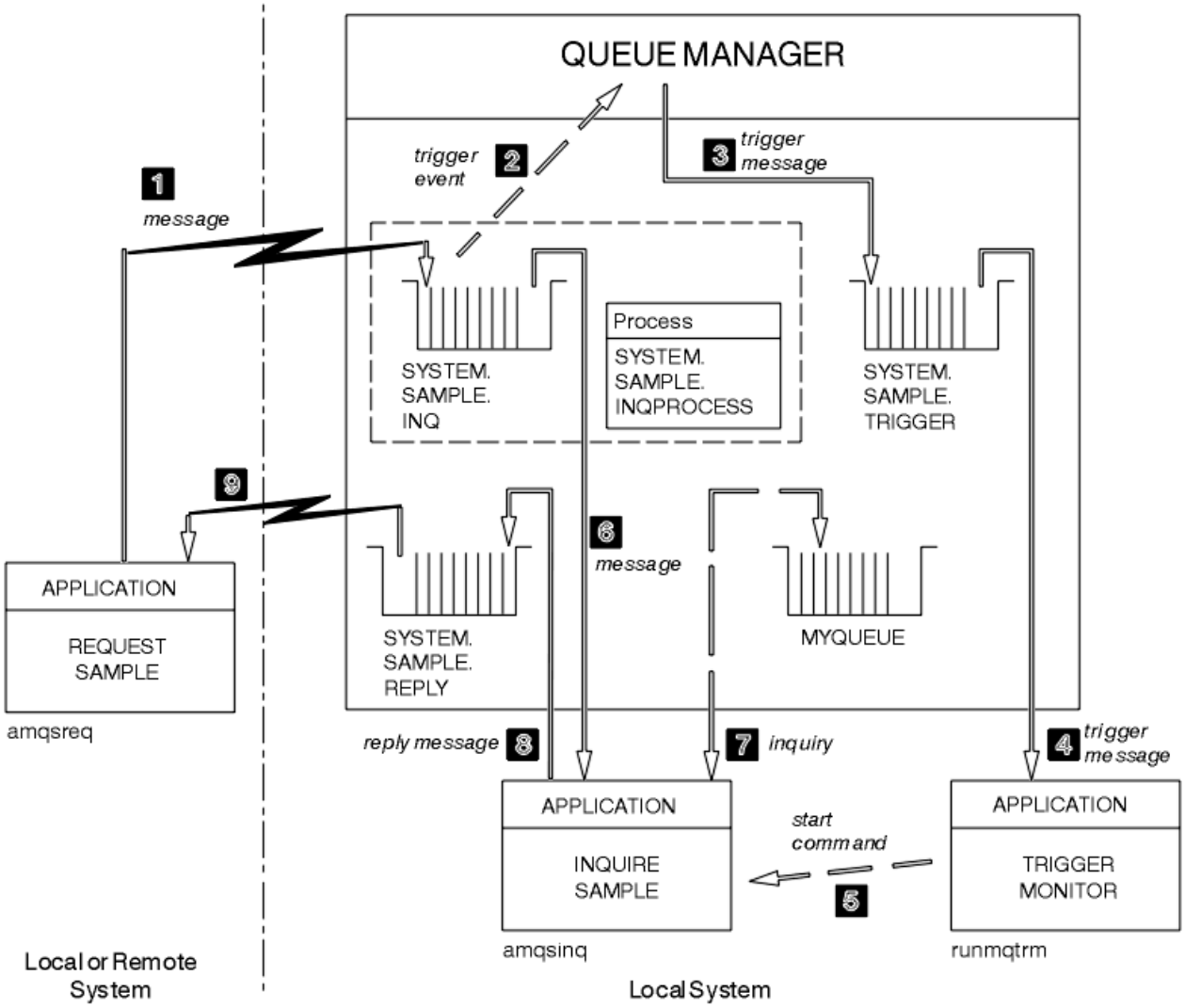


Figure 132. Demander et demander des exemples à l'aide du déclenchement

Dans Figure 132, à la page 1148 , l'exemple Demande insère des messages dans la file d'attente du serveur cible, SYSTEM.SAMPLE.INQet l'exemple Inquire interroge la file d'attente MYQUEUE. Vous pouvez également utiliser l'un des exemples de files d'attente définis lorsque vous avez exécuté amqscos0.tst, ou toute autre file d'attente que vous avez définie, pour l'exemple Inquire.

Remarque : Les nombres dans Figure 132, à la page 1148 indiquent la séquence des événements.

Pour exécuter les exemples de demande et d'interrogation à l'aide du déclenchement:

1. Vérifiez que les files d'attente que vous souhaitez utiliser sont définies. Exécutez amqscos0.tstpour définir les exemples de files d'attente et une file d'attente MYQUEUE.
2. Exécutez la commande RUNMQTRM du moniteur de déclenchement:

   ```
   RUNMQTRM -m qmanagername -q SYSTEM.SAMPLE.TRIGGER
   ```

3. Exécuter l'exemple de demande

   ```
   amqsreq SYSTEM.SAMPLE.INQ
   ```

   Remarque : L'objet de processus définit ce qui doit être déclenché. Si le client et le serveur ne sont pas en cours d'exécution sur la même plateforme, tous les processus démarrés par le moniteur de

déclenchement doivent définir *ApplType*, sinon le serveur prend ses définitions par défaut (c'est-à-dire le type d'application normalement associé à la machine du serveur) et provoque un échec.

Pour obtenir la liste des types d'application, voir ApplType.

4. Entrez le nom de la file d'attente que l'exemple Inquire doit utiliser:

```
MYQUEUE
```

5. Entrez une ligne vide (pour mettre fin au programme de demande).
6. L'échantillon de requête affichera alors un message contenant les données du programme Inquire obtenues à partir de MYQUEUE.

Vous pouvez utiliser plusieurs files d'attente ; dans ce cas, entrez les noms des autres files d'attente à l'étape «4», à la page 1149.

Pour plus d'informations sur le déclenchement, voir «Démarrage des applications IBM MQ à l'aide de déclencheurs», à la page 888.

IBM i *Exécution de l'exemple Demande à l'aide du déclenchement sous IBM i*

Sous IBM i, démarrez l'exemple de serveur de déclenchement, AMQSERV4, dans un travail, puis démarrez AMQSREQ4 dans un autre travail. Cela signifie que le serveur de déclenchement est prêt lorsque l'exemple de programme de demande envoie un message.

Remarque :

1. Les exemples de définition créés par AMQSAMP4 déclenchent les versions C des exemples. Si vous souhaitez déclencher les versions COBOL, modifiez les définitions de processus SYSTEM.SAMPLE.ECHOPROCESS, SYSTEM.SAMPLE.INQPROCESSet SYSTEM.SAMPLE.SETPROCESS. Vous pouvez utiliser la commande CHGMQMPRC (pour plus de détails, voir Modification du processus MQ (CHGMQMPRC)) pour ce faire, ou éditez et exécutez votre propre version de AMQSAMP4.
2. Le code source de AMQSERV4 est fourni pour la langue C uniquement. Toutefois, une version compilée (que vous pouvez utiliser avec les exemples COBOL) est fournie dans la bibliothèque QMQM.

Vous pouvez placer vos messages de demande dans les files d'attente du serveur d'exemples suivantes:

- SYSTEM.SAMPLE.ECHO (pour les exemples de programmes Echo)
- SYSTEM.SAMPLE.INQ (pour les exemples de programmes Inquire)
- SYSTEM.SAMPLE.SET (pour les exemples de programme Set)

Un diagramme de flux pour SYSTEM.SAMPLE.ECHO est présenté dans la Figure 133, à la page 1151. A l'aide de l'exemple de fichier de données, la commande permettant d'émettre la demande de programme C sur ce serveur est la suivante:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Remarque : Cet exemple de file d'attente a un type de déclencheur FIRST. Par conséquent, s'il existe déjà des messages dans la file d'attente avant l'exécution de l'exemple Demande, les applications serveur ne sont pas déclenchées par les messages que vous envoyez.

Si vous souhaitez essayer d'autres exemples, vous pouvez essayer les variantes suivantes:

- Utilisez AMQSTRG4 (ou son équivalent en ligne de commande STRMQMTRM, pour plus de détails, voir Start MQ Trigger Monitor (STRMQMTRM)) au lieu de AMQSERV4 pour soumettre le travail à la place, mais les retards de soumission de travail potentiels pourraient rendre plus difficile le suivi de ce qui se passe.
- Exécutez SYSTEM.SAMPLE.INQUIRE et SYSTEM.SAMPLE.SET . A l'aide de l'exemple de fichier de données, les commandes permettant d'émettre les demandes de programme C vers ces serveurs sont, respectivement:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Ces exemples de files d'attente ont également un type de déclencheur FIRST.

Conception de l'exemple de programme de demande
Le programme ouvre la file d'attente du serveur cible pour qu'elle puisse insérer des messages. Il utilise l'appel MQOPEN avec l'option MQOO_OUTPUT. S'il ne parvient pas à ouvrir la file d'attente, le programme affiche un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN.

Le programme ouvre ensuite la file d'attente de réponses appelée SYSTEM.SAMPLE.REPLY pour obtenir des messages de réponse. Pour cela, le programme utilise l'appel MQOPEN avec l'option MQOO_INPUT_EXCLUSIVE. S'il ne parvient pas à ouvrir la file d'attente, le programme affiche un message d'erreur contenant le code anomalie renvoyé par l'appel MQOPEN.

Pour chaque ligne d'entrée, le programme lit ensuite le texte dans une mémoire tampon et utilise l'appel MQPUT pour créer un message de demande contenant le texte de cette ligne. Dans cet appel, le programme utilise l'option de rapport MQRO_EXCEPTION_WITH_DATA pour demander que les messages de rapport envoyés à propos du message de demande incluent les 100 premiers octets des données de message. Le programme se poursuit jusqu'à ce qu'il atteigne la fin de l'entrée ou que l'appel MQPUT échoue.

Le programme utilise ensuite l'appel MQGET pour supprimer les messages de réponse de la file d'attente et affiche les données contenues dans les réponses. L'appel MQGET utilise les options MQGMO_WAIT, MQGMO_CONVERT et MQGMO_ACCEPT_TRUNCATED. *WaitInterval* correspond à 5 minutes dans la version COBOL et à 1 minute dans la version C pour la première réponse (afin de permettre le déclenchement d'une application serveur) et à 15 secondes pour les réponses suivantes. Le programme attend ces périodes s'il n'y a pas de message dans la file d'attente. Si aucun message n'arrive avant l'expiration de cet intervalle, l'appel échoue et renvoie le code anomalie MQRC_NO_MSG_AVAILABLE. L'appel utilise également l'option MQGMO_ACCEPT_TRUNCATED_MSG, de sorte que les messages dépassant la taille de mémoire tampon déclarée soient tronqués.

Le programme montre comment effacer les zones *MsgId* et *CorrelId* de la structure MQMD après chaque appel MQGET car l'appel définit ces zones sur les valeurs contenues dans le message qu'il extrait. L'effacement de ces zones signifie que les appels MQGET successifs extraient les messages dans l'ordre dans lequel ils sont placés dans la file d'attente.

Le programme se poursuit jusqu'à ce que l'appel MQGET renvoie le code anomalie MQRC_NO_MSG_AVAILABLE ou que l'appel MQGET échoue. Si l'appel échoue, le programme affiche un message d'erreur contenant le code anomalie.

Le programme ferme ensuite la file d'attente du serveur cible et la file d'attente de réponse à l'aide de l'appel MQCLOSE.

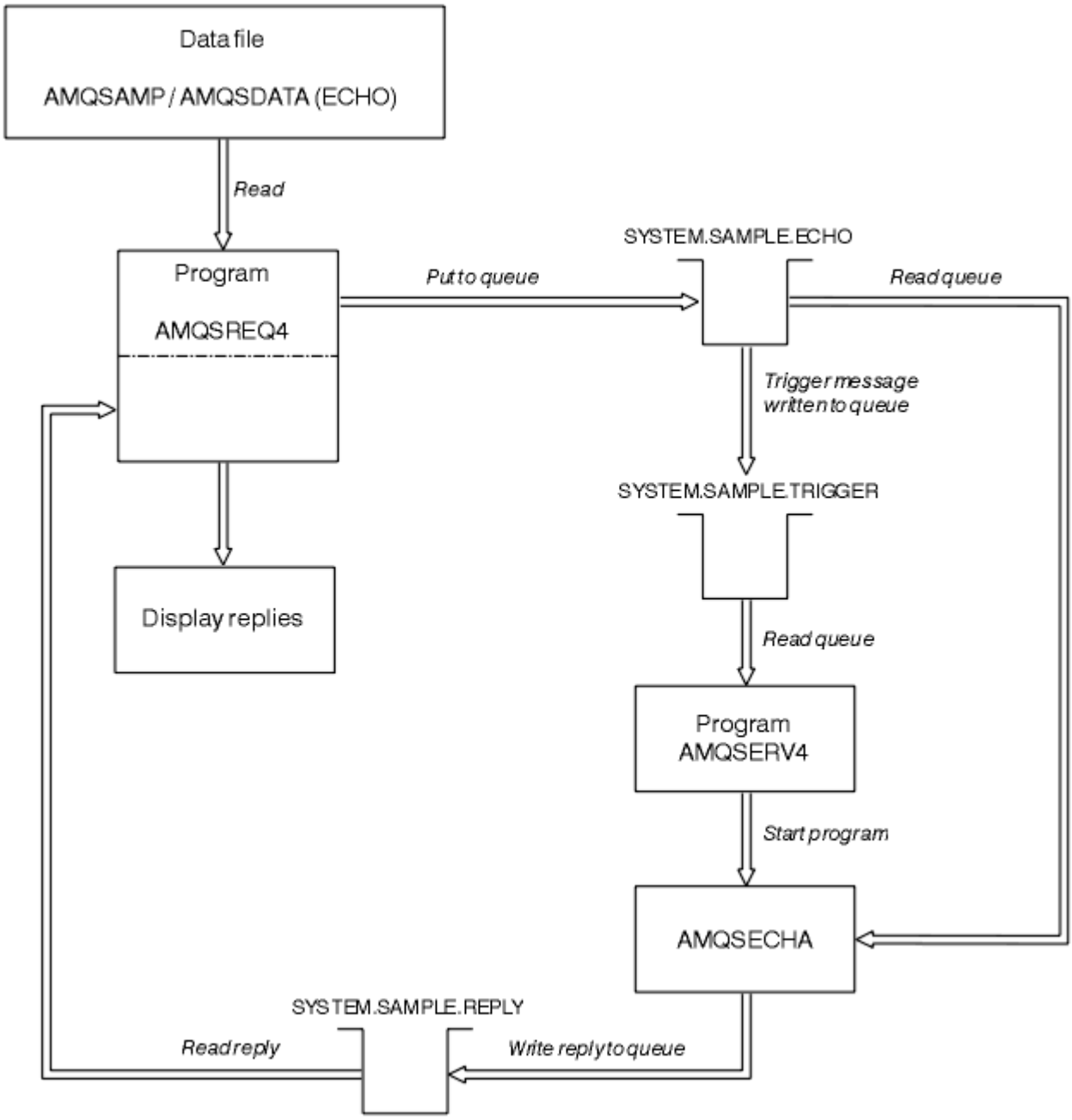


Figure 133. Exemple de diagramme de programme client-serveur IBM i (Echo)

Exemples de programme Set

Les exemples de programme Set inhibent les opérations d'insertion dans une file d'attente en utilisant l'appel MQSET pour modifier l'attribut **InhibitPut** de la file d'attente. Découvrez également la conception des exemples de programmes Set.

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Les programmes étant destinés à être exécutés en tant que programmes déclenchés, leur seule entrée est une structure MQTMC2 (message de déclenchement) qui contient le nom d'une file d'attente cible avec les attributs à appeler. La version C utilise également le nom du gestionnaire de files d'attente. La version COBOL utilise le gestionnaire de files d'attente par défaut.

Pour que le processus de déclenchement fonctionne, assurez-vous que l'exemple de programme Set que vous souhaitez utiliser est déclenché par les messages arrivant dans la file d'attente

SYSTEM.SAMPLE.SET. Pour ce faire, indiquez le nom de l'exemple de programme Set que vous souhaitez utiliser dans la zone *ApplicId* de la définition de processus SYSTEM.SAMPLE.SETPROCESS. Le type de déclencheur de l'exemple de file d'attente est FIRST ; s'il y a déjà des messages dans la file d'attente avant l'exécution de l'exemple de demande, l'exemple de définition n'est pas déclenché par les messages que vous envoyez.

Lorsque vous avez correctement défini la définition:

- ALW Pour les systèmes AIX, Linux, and Windows , démarrez le programme **runmqtrm** dans une session, puis démarrez le programme amqsreq dans une autre.
- IBM i Pour IBM i, démarrez le programme AMQSERV4 dans une session, puis démarrez le programme AMQSREQ4 dans une autre. Vous pouvez utiliser AMQSTRG4 au lieu de AMQSERV4, mais les retards potentiels de soumission des travaux peuvent rendre plus difficile le suivi de ce qui se passe.

Utilisez les exemples de programme de demande pour envoyer des messages de demande, chacun contenant uniquement un nom de file d'attente, à la file d'attente SYSTEM.SAMPLE.SET. Pour chaque message de demande, les exemples de programme Set envoient un message de réponse contenant une confirmation que les opérations d'insertion ont été interdites dans la file d'attente indiquée. Les réponses sont envoyées à la file d'attente de réponse indiquée dans le message de demande.

Conception de l'exemple de programme Set

Le programme ouvre la file d'attente nommée dans la structure de message de déclenchement qu'il a transmise lors de son démarrage. (Pour plus de clarté, nous appellerons cela la *file d'attente des demandes*.) Le programme utilise l'appel MQOPEN pour ouvrir cette file d'attente pour l'entrée partagée.

Le programme utilise l'appel MQGET pour supprimer des messages de cette file d'attente. Cet appel utilise les options MQGMO_ACCEPT_TRUNCATED_MSG et MQGMO_WAIT, avec un intervalle d'attente de 5 secondes. Le programme teste le descripteur de chaque message pour voir s'il s'agit d'un message de demande ; dans le cas contraire, le programme supprime le message et affiche un message d'avertissement.

Pour chaque message de demande supprimé de la file d'attente des demandes, le programme lit le nom de la file d'attente (que nous appellerons la *file d'attente cible*) contenues dans les données et ouvre cette file d'attente à l'aide de l'appel MQOPEN avec l'option MQOO_SET. Le programme utilise ensuite l'appel MQSET pour définir la valeur de l'attribut **InhibitPut** de la file d'attente cible sur MQQA_PUT_INHIBÉ.

Si l'appel MQSET aboutit, le programme utilise l'appel MQPUT1 pour placer un message de réponse dans la file d'attente de réponse. Ce message contient la chaîne `PUT inhibited`.

Si l'appel MQOPEN ou MQSET échoue, le programme utilise l'appel MQPUT1 pour placer un message `report` dans la file d'attente de réponse. Dans la zone *Feedback* du descripteur de message de ce message de rapport, il s'agit du code anomalie renvoyé par l'appel MQOPEN ou MQSET, selon celui qui a échoué.

Après l'appel MQSET, le programme ferme la file d'attente cible à l'aide de l'appel MQCLOSE.

Lorsqu'il ne reste aucun message dans la file d'attente des demandes, le programme ferme cette file d'attente et se déconnecte du gestionnaire de files d'attente.

Exemple de programme TLS

AMQSSSLC est un exemple de programme C qui explique comment utiliser les structures MQCNO et MQSCO pour fournir des informations de connexion client TLS sur l'appel MQCONNX. Cela permet à une application MQI client de fournir la définition de son canal de connexion client et des paramètres TLS lors de l'exécution sans table de définition de canal du client (CCDT).

Si un nom de connexion est fourni, le programme construit une définition de canal de connexion client dans une structure MQCD.

Si le nom de radical du fichier de référentiel de clés est fourni, le programme construit une structure MQSCO ; si une URL de répondeur OCSP est également fournie, le programme construit une structure MQAIR d'enregistrement des informations d'authentification.

Le programme se connecte ensuite au gestionnaire de files d'attente à l'aide de MQCONNX. Il demande et imprime le nom du gestionnaire de files d'attente auquel il s'est connecté.

Ce programme est destiné à être lié en tant qu'application client MQI. Toutefois, il peut être lié en tant qu'application MQI standard, auquel cas il se connecte simplement à un gestionnaire de files d'attente local et ignore les informations de connexion client.

Si la phrase passe permettant d'accéder au référentiel de clés n'est pas stockée dans un fichier, vous devez la fournir à **amqssslc** lors de l'exécution de l'application. Vous pouvez fournir la phrase passe en:

- Demande à **amqssslc** de demander la phrase passe, ou
- Utilisation de la variable d'environnement *MQKEYRPWD* , ou
- Utilisation de l'attribut **SSLKeyRepositoryPassword** dans le fichier de configuration du client

Pour plus d'informations sur la fourniture du mot de passe du référentiel de clés aux applications IBM MQ MQI client , voir Fourniture du mot de passe du référentiel de clés pour un IBM MQ MQI client sur AIX, Linux, and Windows.

amqssslc accepte les paramètres suivants, qui sont tous facultatifs:

-m QmgrName
Nom du gestionnaire de files d'attente auquel se connecter

-c ChannelName
Nom du canal à utiliser

-x ConnName
Nom de la connexion au serveur

Paramètres TLS:

-k KeyReposFileName
Nom du fichier de référentiel de clés. Si l'extension de fichier n'est pas fournie, elle est supposée être `.kdb`. Exemple :

```
/home/user/client.kdb
C:\User\client.p12
```

-s CipherSpec
Chaîne CipherSpec du canal TLS correspondant à **SSLCIPH** sur la définition de canal SVRCONN du gestionnaire de files d'attente.

-f
Indique que seuls les algorithmes certifiés FIPS 140-2 doivent être utilisés.

-b VALUE1[,VALUE2...]
Indique que seuls les algorithmes compatibles Suite B doivent être utilisés. Ce paramètre est une liste séparée par des virgules d'une ou de plusieurs des valeurs suivantes: NONE,128_BIT,192_BIT. Ces valeurs ont la même signification que celles de la variable d'environnement **MQSUITEB** et du paramètre **EncryptionPolicySuiteB** équivalent dans la section SSL du fichier de configuration du client.

-p Stratégie
Indique la règle de validation de certificat à utiliser. Il peut s'agir de l'une des valeurs suivantes:

ANY
Appliquez chacune des règles de validation de certificat prises en charge par la bibliothèque de sockets sécurisés et acceptez la chaîne de certificats si l'une des règles considère que la chaîne de certificats est valide. Ce paramètre peut être utilisé pour une compatibilité en amont maximale avec les anciens certificats numériques qui ne sont pas conformes aux normes de certificat modernes.

RFC5280
Appliquez uniquement la règle de validation de certificat conforme à la norme RFC 5280. Ce paramètre fournit une validation plus stricte que le paramètre ANY, mais rejette certains certificats numériques plus anciens.

La valeur par défaut est Indifférent.

-l CertLabel
Libellé de certificat à utiliser pour la connexion sécurisée.

Remarque : Vous devez spécifier la valeur en minuscules.

-w
Indique que **amqssslc** vous invite à indiquer la phrase passe du référentiel de clés.

-i
Indique que **amqssslc** vous invite à indiquer la clé initiale utilisée pour chiffrer la phrase passe du référentiel de clés à fournir.

Spécifiez cette option si un fichier de clés initial a été spécifié lorsque la phrase passe du référentiel de clés a été chiffrée à l'aide de l'utilitaire **runmqicred** .

Paramètre de révocation de certificat OCSP:

-o URL
URL du répondeur OCSP

Vous pouvez également définir l'une des variables d'environnement suivantes pour fournir les données d'identification utilisées pour l'authentification auprès du gestionnaire de files d'attente:

ID_UTILISATEUR_MQM
Définissez l'ID utilisateur à utiliser pour l'authentification de connexion, si vous souhaitez utiliser un ID utilisateur et un mot de passe pour vous authentifier auprès du gestionnaire de files d'attente. Le programme demande le mot de passe qui doit accompagner l'ID utilisateur.

V 9.4.0 Linux AIX **MQSAMP_TOKEN**
Indiquez une valeur non vide si vous souhaitez fournir un jeton d'authentification pour l'authentification auprès du gestionnaire de files d'attente. Le programme demande le jeton d'authentification.

Exécution de l'exemple de programme TLS
Pour exécuter l'exemple de programme TLS, vous devez d'abord configurer votre environnement TLS. Vous exécutez ensuite l'exemple à partir de la ligne de commande, en fournissant un certain nombre de paramètres.

Pourquoi et quand exécuter cette tâche

Les instructions suivantes exécutent l'exemple de programme à l'aide de certificats personnels. En faisant varier la commande, vous pouvez, par exemple, utiliser des certificats de l'autorité de certification et vérifier leur statut à l'aide d'un répondeur OCSP. Consultez les instructions de l'exemple.

Procédure

1. Créez un gestionnaire de files d'attente nommé QM1. Pour plus d'informations, voir crtmqm.
2. Créez un référentiel de clés pour le gestionnaire de files d'attente. Pour plus d'informations, voir Setting up a key repository on AIX, Linux, and Windows.
3. Créez un référentiel de clés pour le client. Appelez-le *clientkey.kdb*.

   Stockez le mot de passe du référentiel de clés dans un fichier lorsque vous créez le référentiel de clés.
4. Créez un certificat personnel pour le gestionnaire de files d'attente. Pour plus d'informations, voir Creating a self-signed personal certificate on AIX, Linux, and Windows.
5. Créez un certificat personnel pour le client.

6. Procédez à l'extraction du certificat personnel depuis le référentiel de clés du serveur et ajoutez-le dans le référentiel du client. Pour plus d'informations, voir Extraction de la partie publique d'un certificat auto-signé à partir d'un référentiel de clés sous AIX, Linux, and Windows, et Ajout d'un certificat d'autorité de certification (ou de la partie publique d'un certificat autosigné) dans un référentiel de clés, sur les systèmesAIX, Linux, and Windows.
7. Procédez à l'extraction du certificat personnel depuis le référentiel de clés du client et ajoutez-le dans le référentiel de clés du serveur.
8. Créez un canal de connexion serveur à l'aide de la commande MQSC:

   ```
   DEFINE CHANNEL(QM1SVRCONN) CHLTYPE(SVRCONN) TRPTYPE(TCP)
   SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
   ```

   Pour plus d'informations, voir Canal de connexion serveur
9. Définissez et démarrez un programme d'écoute de canal sur le gestionnaire de files d'attente. Pour plus d'informations, voir DEFINE LISTENER et START LISTENER.
10. Exécutez l'exemple de programme à l'aide de la commande suivante:

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "C:\Program Files\IBM\MQ\clientkey.kdb" -s TLS_RSA_WITH_AES_128_CBC_SHA256
    -o http://dummy.OCSP.responder
    ```

Résultats

L'exemple de programme effectue les actions suivantes:

1. Se connecte à un gestionnaire de files d'attente spécifié ou au gestionnaire de files d'attente par défaut, à l'aide des options spécifiées.
2. Ouvre le gestionnaire de files d'attente et recherche son nom.
3. Ferme le gestionnaire de files d'attente.
4. Se déconnecte du gestionnaire de files d'attente.

Si l'exemple de programme s'exécute correctement, il affiche une sortie similaire à l'exemple suivant:

```
Sample AMQSSSLC start
Connecting to queue manager QM1
Using the server connection channel QM1SVRCONN
on connection name localhost.
Using TLS CipherSpec TLS_RSA_WITH_AES_128_CBC_SHA256
Using TLS key repository stem C:\Program Files\IBM\MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Si l'exemple de programme rencontre un problème, il affiche un message d'erreur approprié. Par exemple, si vous spécifiez une URL de répondeur OCSP non valide, vous recevez le message suivant:

```
MQCONNX ended with reason code 2553
```

Pour obtenir la liste des codes anomalie, voir Codes anomalie et achèvement d'API.

Exemples de programmes de déclenchement

La fonction fournie dans l'exemple de déclenchement est un sous-ensemble de celle fournie dans le moniteur de déclenchement dans le programme **runmqtrm** .

Pour connaître les noms de ces programmes, voir «Fonctions démontrées dans les exemples de programmes sur Multiplatforms», à la page 1088 .

Conception de l'échantillon déclencheur

L'exemple de programme de déclenchement ouvre la file d'attente d'initialisation à l'aide de l'appel MQOPEN avec l'option MQOO_INPUT_AS_Q_DEF. Il extrait les messages de la file d'attente d'initialisation à l'aide de l'appel MQGET avec les options MQGMO_ACCEPT_TRUNCATED_MSG et MQGMO_WAIT, en spécifiant un intervalle d'attente illimité. Le programme efface les zones *MsgId* et *CorrelId* avant chaque appel MQGET pour obtenir les messages dans l'ordre.

Lorsqu'il a extrait un message de la file d'attente d'initialisation, le programme le teste en vérifiant la taille du message pour s'assurer qu'il est de la même taille qu'une structure MQTM. Si ce test échoue, le programme affiche un avertissement.

Pour les messages de déclenchement valides, l'exemple de déclenchement copie les données des zones suivantes: *ApplicId*, *EnvrData*, *Version*et *ApplType*. Les deux dernières de ces zones étant numériques, le programme crée des remplacements de caractères à utiliser dans une structure MQTMC2 pour les systèmes IBM i, AIX, Linux, and Windows .

L'exemple de déclenchement émet une commande de démarrage vers l'application spécifiée dans la zone *ApplicId* du message de déclenchement et transmet une structure MQTMC2 ou MQTMC (version alphanumérique du message de déclenchement).

- ALW Dans les systèmes AIX, Linux, and Windows , la zone *EnvData* est utilisée comme extension de la chaîne de commande appelante.
- IBM i Dans IBM i, il est utilisé en tant que paramètres de soumission de travail, par exemple, la priorité de travail ou la description de travail.

Enfin, le programme ferme la file d'attente d'initialisation.

Arrêt des exemples de programme de déclenchement sous IBM i

IBM i

Un programme de moniteur de déclenchement peut être arrêté par l'option 2 de sysrequest (ENDRQS) ou en inhibant les extractions de la file d'attente de déclenchement.

Si l'exemple de file d'attente de déclenchement est utilisé, la commande est la suivante:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') MQMNAME GETENBL(*NO)
```

Important : Avant de relancer le déclenchement sur cette file d'attente, vous devez entrer la commande suivante:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Exécution des exemples de programmes de déclenchement
Cette rubrique contient des informations sur l'exécution des exemples de programme Déclenchement.

Exécution des exemples amqstrg0.c, amqstrg et amqstrgc

Le programme prend 2 paramètres:

1. Nom de la file d'attente d'initialisation (nécessaire)
2. Nom du gestionnaire de files d'attente (facultatif)

Si aucun gestionnaire de files d'attente n'est spécifié, il se connecte à celui par défaut. Un exemple de file d'attente d'initialisation a été défini lorsque vous avez exécuté amqscos0.tst; le nom de cette file d'attente est SYSTEM.SAMPLE.TRIGGERet vous pouvez l'utiliser lorsque vous exécutez ce programme.

Remarque : La fonction de cet exemple est un sous-ensemble de la fonction de déclenchement complet fournie dans le programme runmqtrm.

Exécution de l'exemple AMQSTRG4

IBM i

Il s'agit d'un moniteur de déclenchement pour l'environnement IBM i . Il soumet un travail IBM i pour chaque application à démarrer. Cela signifie qu'un traitement supplémentaire est associé à chaque message de déclenchement.

AMQSTRG4 (dans QCSRC) prend deux paramètres: le nom de la file d'attente d'initialisation qu'il doit servir et le nom du gestionnaire de files d'attente (facultatif). AMQSAMP4 (dans QCLSRC) définit un exemple de file d'attente d'initialisation, SYSTEM.SAMPLE.TRIGGER, que vous pouvez utiliser lorsque vous essayez les exemples de programme.

A l'aide de l'exemple de file d'attente de déclenchement, la commande à émettre est la suivante:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Vous pouvez également utiliser l'équivalent CL STRMQMTRM ; pour plus de détails, voir Démarrer le moniteur de déclenchement MQ (STRMQMTRM).

Exécution de l'exemple AMQSERV4

IBM i

Il s'agit d'un serveur de déclenchement pour l'environnement IBM i . Pour chaque message de déclenchement, ce serveur exécute la commande de démarrage dans son propre travail pour démarrer l'application spécifiée. Le serveur de déclenchement peut appeler des transactions CICS .

AMQSERV4 prend deux paramètres: le nom de la file d'attente d'initialisation qu'il doit servir et le nom du gestionnaire de files d'attente (facultatif). AMQSAMP4 définit un exemple de file d'attente d'initialisation, SYSTEM.SAMPLE.TRIGGER, que vous pouvez utiliser lorsque vous essayez les exemples de programme.

A l'aide de l'exemple de file d'attente de déclenchement, la commande à émettre est la suivante:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Conception du serveur de déclenchement
La conception du serveur de déclenchement est similaire à celle du moniteur de déclenchement, à quelques exceptions près.

La conception du serveur de déclenchement est similaire à celle du moniteur de déclenchement, sauf que le serveur de déclenchement:

- Autorise les applications MQAT_CICS et MQAT_OS400 .
- IBM i Appelle des applications IBM i dans son propre travail (ou utilise STRCICSUSR pour démarrer des applications CICS) au lieu de soumettre un travail IBM i .
- Pour les applications CICS , remplace *EnvData*, par exemple, pour spécifier la région CICS , à partir du message de déclenchement dans la commande STRCICSUSR.
- Ouvre la file d'attente d'initialisation pour l'entrée partagée, de sorte que de nombreux serveurs de déclenchement puissent s'exécuter en même temps.

Remarque : Les programmes démarrés par AMQSERV4 ne doivent pas utiliser l'appel MQDISC car cela arrête le serveur de déclenchement. Si les programmes démarrés par AMQSERV4 utilisent l'appel MQCONN, ils obtiennent le code anomalie MQRC_ALREADY_CONNECTED.

ALW *Utilisation des exemples TUXEDO sur AIX, Linux, and Windows*

Découvrez les exemples de programmes Put et Get pour TUXEDO et générez l'environnement de serveur dans TUXEDO.

Avant de commencer

Avant d'exécuter ces exemples, vous devez générer l'environnement de serveur.

Pourquoi et quand exécuter cette tâche

Remarque : Dans cette section, la barre oblique inversée (\) est utilisée pour fractionner les commandes longues sur plusieurs lignes. N'entrez pas ce caractère. Entrez chaque commande sur une seule ligne.

ALW *Génération de l'environnement de serveur*
Informations sur la génération de l'environnement de serveur pour IBM MQ pour différentes plateformes.

Avant de commencer

Il est supposé que vous disposez d'un environnement TUXEDO fonctionnel.

AIX *Génération de l'environnement de serveur pour AIX (32 bits)*
Comment générer l'environnement de serveur pour IBM MQ for AIX (32 bits).

Procédure

1. Créez un répertoire (par exemple, APPDIR) dans lequel l'environnement de serveur est généré et exécutez toutes les commandes de ce répertoire.
2. Exportez les variables d'environnement suivantes, où TUXDIR est le répertoire racine de TUXEDO, et *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est installé:

```
$ 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. Ajoutez la ligne suivante au fichier TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa -lmqm
```

4. Exécutez les commandes suivantes :

```
$ 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. Editez ubbstxcx.cfg et ajoutez des détails sur le nom de la machine, les répertoires de travail et le gestionnaire de files d'attente, si nécessaire:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Créez TLOGDEVICE:

```
$tmadmin -c
```

Une invite s'affiche. A cette invite, entrez:

```
> crdl -z /APPDIR/TLOG1
```

7. Démarrez le gestionnaire de files d'attente :

```
$ strmqm
```

8. Démarrez Tuxedo:

```
$ tmboot -y
```

Que faire ensuite

Vous pouvez maintenant utiliser les programmes doputs et dogets pour placer des messages dans une file d'attente et les extraire d'une file d'attente.

AIX *Génération de l'environnement de serveur pour AIX (64 bits)*

Comment générer l'environnement de serveur pour IBM MQ for AIX (64 bits).

Procédure

1. Créez un répertoire (par exemple, APPDIR) dans lequel l'environnement de serveur est généré et exécutez toutes les commandes de ce répertoire.
2. Exportez les variables d'environnement suivantes, où TUXDIR représente le répertoire racine de TUXEDO, et *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM MQ est 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. Ajoutez la ligne suivante au fichier TUXEDO udataobj/RM:

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: -lmqmxa64 -lmqm
```

4. Exécutez les commandes suivantes :

```
$ 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. Editez ubbstxcx.cfg et ajoutez des détails sur le nom de la machine, les répertoires de travail et le gestionnaire de files d'attente, si nécessaire:

```
$ tmloadcf  -y MQ_INSTALLATION_PATH/samp/ubbstxcx.cfg
```

6. Créez TLOGDEVICE:

```
$tmadmin -c
```

   Une invite s'affiche. A cette invite, entrez:

```
> crdl -z /APPDIR/TLOG1
```

7. Démarrez le gestionnaire de files d'attente :

```
$ strmqm
```

8. Démarrez Tuxedo:

```
$ tmboot -y
```

Que faire ensuite

Vous pouvez maintenant utiliser les programmes doputs et dogets pour placer des messages dans une file d'attente et les extraire d'une file d'attente.

Génération de l'environnement de serveur pour Windows (32 bits)
Génération de l'environnement de serveur pour IBM MQ for Windows (32 bits).

Pourquoi et quand exécuter cette tâche

Remarque : Remplacez les zones identifiées en tant que *VARIABLES* dans ce qui suit par les chemins de répertoire:

Tableau 164. Zones à modifier en chemins de répertoire

Zone	Chemin du répertoire
REP_MQM	Chemin de répertoire spécifié lors de l'installation de IBM MQ , par exemple `g:\Program Files\IBM\MQ`.
TUXDIR	Chemin de répertoire spécifié lors de l'installation de TUXEDO, par exemple `f:\tuxedo`.
APPDIR	Chemin de répertoire à utiliser pour l'exemple d'application, par exemple `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Program Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figure 134. Exemple de fichier ubbstxcn.cfg pour IBM MQ for Windows

Remarque : Modifiez le nom de machine *MachineName*et les chemins de répertoire en fonction de votre installation. Remplacez également le nom du gestionnaire de files d'attente *MYQUEUEMANAGER* par le nom du gestionnaire de files d'attente auquel vous souhaitez vous connecter.

L'exemple de fichier ubbconfig pour IBM MQ for Windows est répertorié dans Figure 134, à la page 1161. Il est fourni sous la forme ubbstxcn.cfg dans le répertoire des exemples IBM MQ .

L'exemple de fichier makefile (voir Figure 135, à la page 1162) fourni pour IBM MQ for Windows est appelé ubbstxmn.maket se trouve dans le répertoire des exemples IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
$(TUXDIR)\bin\buildclient  -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figure 135. Exemple de fichier makefile TUXEDO pour IBM MQ for Windows

Pour générer l'environnement de serveur et les exemples, procédez comme suit.

Procédure

1. Créez un répertoire d'application dans lequel générer le modèle d'application, par exemple:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copiez les exemples de fichier suivants depuis le répertoire d'exemple IBM MQ vers le répertoire d'application:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Editez chacun de ces fichiers pour définir les noms de répertoire et les chemins de répertoire utilisés sur votre installation.
4. Editez `ubbstxcn.cfg` (voir Figure 134, à la page 1161) pour ajouter des détails sur le nom de la machine et le gestionnaire de files d'attente auquel vous souhaitez vous connecter.
5. Ajoutez la ligne suivante au fichier TUXEDO *TUXDIR*`udataobj\rm`:

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib\mqmxa.lib MQMDIR\tools\lib\mqm.lib
   ```

   La nouvelle entrée doit comporter une ligne dans le fichier.
6. Définissez les variables d'environnement suivantes :

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Créez une unité TLOG pour TUXEDO.

   Pour ce faire, appelez `tmadmin -c`et entrez la commande suivante:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Définissez le répertoire de travail sur *APPDIR*et appelez l'exemple de fichier makefile `amqstxmn.mak` en tant que fichier makefile de projet externe. Par exemple, avec Microsoft Visual C++, exécutez la commande suivante:

```
msvc amqstxmn.mak
```

   Sélectionnez **build** pour générer tous les exemples de programme.

Windows *Génération de l'environnement de serveur pour Windows (64 bits)*
Comment générer l'environnement de serveur pour IBM MQ for Windows (64 bits).

Pourquoi et quand exécuter cette tâche

Remarque : Remplacez les zones identifiées en tant que *VARIABLES* dans ce qui suit par les chemins de répertoire:

Tableau 165. Zones à modifier en chemins de répertoire

Zone	Chemin du répertoire
REP_MQM	Chemin de répertoire spécifié lors de l'installation de IBM MQ , par exemple `g:\Program Files\IBM\MQ`.
TUXDIR	Chemin de répertoire spécifié lors de l'installation de TUXEDO, par exemple `f:\tuxedo`.
APPDIR	Chemin de répertoire à utiliser pour l'exemple d'application, par exemple `f:\tuxedo\apps\mqapp`.

```
*RESOURCES
IPCKEY       99999
UID          0
GID          0
MAXACCESSERS 20
MAXSERVERS   20
MAXSERVICES  50
MASTER       SITE1
MODEL        SHM
LDBAL        N

*MACHINES
MachineName LMID=SITE1
            TUXDIR="f:\tuxedo"
            APPDIR="f:\tuxedo\apps\mqapp;g:\Programï¿½Files\IBM\WebSphere MQ\bin"
            ENVFILE="f:\tuxedo\apps\mqapp\amqstxen.env"
            TUXCONFIG="f:\tuxedo\apps\mqapp\tuxconfig"
            ULOGPFX="f:\tuxedo\apps\mqapp\ULOG"
            TLOGDEVICE="f:\tuxedo\apps\mqapp\TLOG"
            TLOGNAME=TLOG
            TYPE="i386NT"
            UID=0
            GID=0

*GROUPS
GROUP1
        LMID=SITE1 GRPNO=1
        TMSNAME=MQXA
        OPENINFO="MQSERIES_XA_RMI:MYQUEUEMANAGER"

*SERVERS
DEFAULT: CLOPT="-A -- -m MYQUEUEMANAGER"

MQSERV1   SRVGRP=GROUP1 SRVID=1
MQSERV2   SRVGRP=GROUP1 SRVID=2

*SERVICES
MPUT1
MGET1
MPUT2
MGET2
```

Figure 136. Exemple de fichier ubbstxcn.cfg pour IBM MQ for Windows

Remarque : Modifiez le nom de machine *MachineName*et les chemins de répertoire en fonction de votre installation. Remplacez également le nom du gestionnaire de files d'attente *MYQUEUEMANAGER* par le nom du gestionnaire de files d'attente auquel vous souhaitez vous connecter.

L'exemple de fichier ubbconfig pour IBM MQ for Windows est répertorié dans Figure 136, à la page 1164. Il est fourni sous la forme `ubbstxcn.cfg` dans le répertoire des exemples IBM MQ .

L'exemple de fichier makefile (voir Figure 137, à la page 1165) fourni pour IBM MQ for Windows est appelé `ubbstxmn.mak`et se trouve dans le répertoire des exemples IBM MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSERIES_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSERIES_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figure 137. Exemple de fichier makefile TUXEDO pour IBM MQ for Windows

Pour générer l'environnement de serveur et les exemples, procédez comme suit.

Procédure

1. Créez un répertoire d'application dans lequel générer le modèle d'application, par exemple:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Copiez les exemples de fichier suivants depuis le répertoire d'exemple IBM MQ vers le répertoire d'application:

   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`

3. Editez chacun de ces fichiers pour définir les noms de répertoire et les chemins de répertoire utilisés sur votre installation.
4. Editez `ubbstxcn.cfg` (voir Figure 136, à la page 1164) pour ajouter des détails sur le nom de machine et le gestionnaire de files d'attente auquel vous souhaitez vous connecter.
5. Ajoutez la ligne suivante au fichier TUXEDO *TUXDIR*`udataobj\rm`

   ```
   MQSERIES_XA_RMI;MQRMIXASwitchDynamic;MQMDIR\tools\lib64\mqmxa64.lib
   MQMDIR\tools\lib64\mqm.lib
   ```

   La nouvelle entrée doit comporter une ligne dans le fichier.
6. Définissez les variables d'environnement suivantes :

   ```
   TUXDIR=TUXDIR
   TUXCONFIG=APPDIR\tuxconfig
   FIELDTBLS=MQMDIR\tools\c\samples\amqstxvx.fld
   LANG=C
   ```

7. Créez une unité TLOG pour TUXEDO. Pour ce faire, appelez `tmadmin -c`et entrez la commande suivante:

```
crdl -z APPDIR\TLOG
```

8. Définissez le répertoire de travail sur *APPDIR*et appelez l'exemple de fichier makefile `amqstxmn.mak` en tant que fichier makefile de projet externe. Par exemple, avec Microsoft Visual C++, exécutez la commande suivante:

   ```
   msvc amqstxmn.mak
   ```

   Sélectionnez **build** pour générer tous les exemples de programme.

ALW *Exemple de programme serveur pour TUXEDO*

Le programme de serveur d'exemples (amqstxsx) est conçu pour s'exécuter avec les exemples de programme Put (amqstxpx.c) et Get (amqstxgx.c). Le programme du serveur d'exemples s'exécute automatiquement lorsque TUXEDO est démarré.

Remarque : Vous devez démarrer votre gestionnaire de files d'attente avant de démarrer TUXEDO.

Le serveur d'exemples fournit deux services TUXEDO, MPUT1 et MGET1:

- Le service MPUT1 est géré par l'exemple PUT et utilise MQPUT1 en point de synchronisation pour insérer un message dans une unité de travail contrôlée par TUXEDO. Il prend les paramètres QName et Message Text, qui sont fournis par l'exemple PUT.
- Le service MGET1 ouvre et ferme la file d'attente chaque fois qu'il reçoit un message. Il prend les paramètres QName et Message Text, qui sont fournis par l'exemple GET.

Les messages d'erreur, les codes anomalie et les messages d'état sont consignés dans le fichier journal TUXEDO.

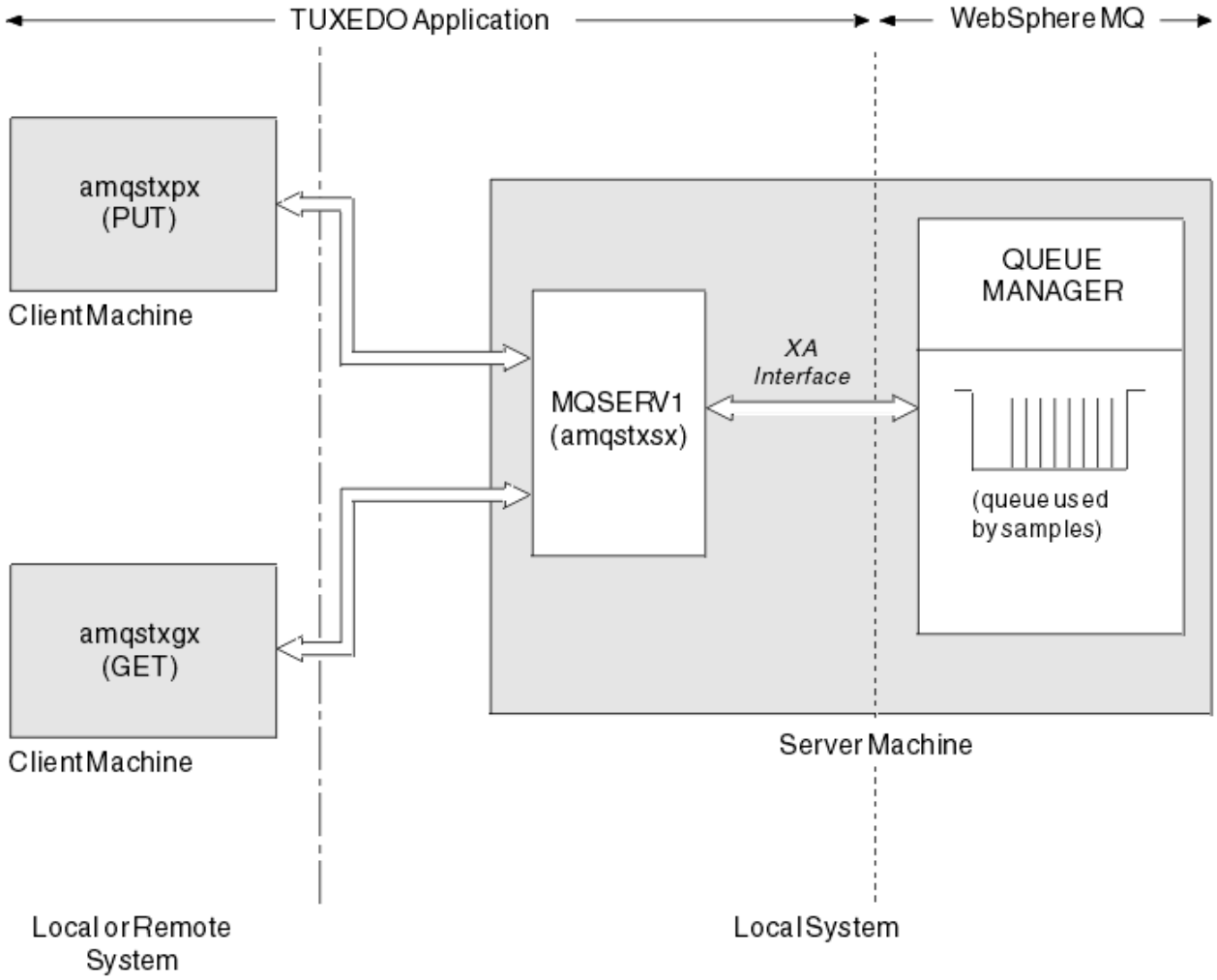


Figure 138. Comment les exemples TUXEDO fonctionnent ensemble

ALW *Exemple de programme d'insertion pour TUXEDO*

Cet exemple vous permet de placer un message dans une file d'attente plusieurs fois, par lots, en démontrant la synchronisation en utilisant TUXEDO comme gestionnaire de ressources.

L'exemple de programme serveur `amqstxsx` doit être en cours d'exécution pour que l'exemple d'insertion aboutisse ; l'exemple de programme serveur se connecte au gestionnaire de files d'attente et utilise l'interface XA. Pour exécuter l'exemple, entrez:

- `doputs -n queuename -b batchsize -c trancount -t message`

Exemple :

- `doputs -n myqueue -b 5 -c 6 -t "Hello World"`

Cela place 30 messages dans la file d'attente nommée `myqueue`, en six lots, chacun contenant cinq messages. S'il y a des problèmes, il renvoie un lot de messages, sinon il les valide.

Tous les messages d'erreur sont écrits dans le fichier journal TUXEDO et dans stderr. Tous les codes raison sont écrits dans stderr.

ALW *Obtenir un exemple pour TUXEDO*

Cet exemple permet d'extraire des messages d'une file d'attente par lots.

Le programme du serveur d'exemples `amqstxsx` doit être en cours d'exécution pour que l'exemple Get aboutisse ; le programme du serveur d'exemples se connecte au gestionnaire de files d'attente et utilise l'interface XA. Pour exécuter l'exemple, entrez la commande suivante:

- `dogets -n queuename -b batchsize -c trancount`

Exemple :

- `dogets -n myqueue -b 6 -c 4`

Cela permet de retirer 24 messages de la file d'attente nommée `myqueue`, en six lots, chacun contenant quatre messages. Si vous l'exécutez après l'exemple d'insertion, qui insère 30 messages sur `myqueue`, vous ne disposez que de six messages sur `myqueue`. Le nombre de lots et la taille de lot peuvent varier entre l'insertion des messages et leur obtention.

Tous les messages d'erreur sont écrits dans le fichier journal TUXEDO et dans stderr. Tous les codes raison sont écrits dans stderr.

Windows *Utilisation de l'exit de sécurité SSPI sous Windows*

Cette rubrique explique comment utiliser les programmes d'exit de canal SSPI sur les systèmes Windows . Le code de sortie fourni est dans deux formats: objet et source.

Code objet

Le fichier de code d'objet est appelé `amqrspin.dll`. Pour le client et le serveur, il est installé en tant que partie standard de IBM MQ for Windows dans le dossier *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* . Par exemple, `C:\Program Files\IBM\MQ\exits\installation2`. Il est chargé en tant qu'exit utilisateur standard. Vous pouvez exécuter l'exit de canal de sécurité fourni et utiliser les services d'authentification dans votre définition du canal.

Pour ce faire, spécifiez l'une des options suivantes:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Pour assurer la prise en charge d'un canal restreint, indiquez ce qui suit sur le canal SVRCONN:

```
SCYDATA('remote_principal_name')
```

où *remote_principal_name* est au format *DOMAIN**user*. Le canal sécurisé est établi uniquement si le nom du principal distant correspond à *remote_principal_name*.

Pour utiliser les programmes d'exit de canal fournis entre les systèmes qui fonctionnent dans un domaine de sécurité Kerberos , créez un **servicePrincipalName** pour le gestionnaire de files d'attente.

Code source

Le fichier de code source d'exit est appelé `amqsspin.c`. Il se trouve dans `C:\Program Files\IBM\MQ\Tools\c\Samples`.

Si vous modifiez le code source, vous devez recompiler la source modifiée.

Vous le compilez et le liez de la même manière que n'importe quel autre exit de canal pour la plateforme appropriée, sauf que les en-têtes SSPI doivent être accessibles lors de la compilation, et que les bibliothèques de sécurité SSPI, ainsi que les bibliothèques associées recommandées, doivent être accessibles lors de la liaison.

Avant d'exécuter la commande suivante, assurez-vous que `cl.exe`, la bibliothèque Visual C++ et le dossier `include` sont disponibles dans votre chemin. Exemple :

```
cl /VERBOSE /LD /MT /Ipath_to_Microsoft_platform_SDK\include
/Ipath_to_IBM_MQ\tools\c\include amqsspin.c /DSECURITY_WIN32
-link /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

Remarque : Le code source n'inclut aucune mise à disposition pour le traçage ou le traitement des erreurs. Si vous modifiez et utilisez le code source, ajoutez vos propres routines de traçage et de traitement des erreurs.

Exécution des exemples à l'aide de files d'attente distantes

Vous pouvez illustrer la mise en file d'attente à distance en exécutant les exemples sur les gestionnaires de files d'attente connectés.

Le programme amqscos0.tst fournit une définition locale d'une file d'attente éloignée (SYSTEM.SAMPLE.REMOTE) qui utilise un gestionnaire de files d'attente éloignées nommé OTHER. Pour utiliser cet exemple de définition, remplacez OTHER par le nom du deuxième gestionnaire de files d'attente que vous souhaitez utiliser. Vous devez également configurer un canal de transmission de messages entre vos deux gestionnaires de files d'attente. Pour plus d'informations sur cette procédure, voir Définition des canaux.

Les exemples de programme de demande placent leur propre nom de gestionnaire de files d'attente local dans la zone *ReplyToQMgr* des messages qu'ils envoient. Les exemples d'interrogation et d'interrogation envoient des messages de réponse à la file d'attente et au gestionnaire de files d'attente de messages nommés dans les zones *ReplyToQ* et *ReplyToQMgr* des messages de demande qu'ils traitent.

Exemple de programme de surveillance de file d'attente de cluster (AMQSCLM)

Cet exemple utilise les fonctions d'équilibrage de charge de cluster IBM MQ intégrées pour diriger les messages vers les instances de files d'attente auxquelles des applications consommatrices sont connectées. Cette direction automatique empêche l'accumulation de messages sur une instance d'une file d'attente de cluster à laquelle aucune application consommatrice n'est connectée.

Présentation

Vous pouvez configurer un cluster comportant plusieurs définitions pour la même file d'attente sur des gestionnaires de files d'attente différents. Cette configuration offre l'avantage d'une disponibilité et d'un équilibrage de la charge de travail accrus. Toutefois, aucune fonction n'est intégrée à IBM MQ pour modifier de manière dynamique la distribution des messages dans un cluster en fonction de l'état des applications associées. Pour cette raison, une application consommatrice doit toujours être associée à chaque instance d'une file d'attente pour s'assurer que les messages sont traités.

L'exemple de programme de surveillance de file d'attente de cluster surveille l'état des applications connectées. Le programme ajuste dynamiquement la configuration d'équilibrage de charge intégrée pour diriger les messages vers les instances d'une file d'attente en cluster avec des applications consommatrices connectées. Dans certaines situations, ce programme peut être utilisé pour réduire la nécessité pour une application consommatrice d'être toujours connectée à chaque instance d'une file d'attente. Il renvoie également les messages qui sont mis en file d'attente sur une instance d'une file d'attente à laquelle aucune application consommatrice n'est connectée. Le renvoi de messages permet d'acheminer les messages autour d'une application consommatrice temporairement arrêtée.

Le programme est conçu pour être utilisé lorsque les applications consommatrices sont des applications à exécution longue, plutôt que d'attacher et de détacher fréquemment des applications.

L'exemple de programme de surveillance de file d'attente de cluster est le programme exécutable compilé de l'exemple de fichier C `amqsclma.c`.

Pour plus d'informations sur les clusters et la charge de travail, voir Utilisation de clusters pour la gestion de la charge de travail .

AMQSCLM: Conception et planification de l'utilisation de l'exemple

Informations sur le fonctionnement de l'exemple de programme de surveillance de file d'attente de cluster, points à prendre en compte lors de la configuration d'un système pour l'exécution de l'exemple de programme et modifications pouvant être apportées à l'exemple de code source.

Conception

L'exemple de programme de surveillance des files d'attente de cluster surveille les files d'attente en cluster locales auxquelles des applications consommatrices sont connectées. Le programme surveille les files d'attente spécifiées par l'utilisateur. Le nom de la file d'attente peut être spécifique, par exemple APP.TEST01, ou générique. Les noms génériques doivent être dans un format conforme au format PCF (Programmable Command Format). Exemples de noms génériques: APP.TEST*ou APP*.

Chaque gestionnaire de files d'attente d'un cluster qui possède une instance d'une file d'attente locale à surveiller requiert qu'une instance de l'exemple de programme de surveillance de file d'attente de cluster lui soit connectée.

Routage dynamique des messages

L'exemple de programme de surveillance de file d'attente de cluster utilise la valeur **IPPROCS** (ouvert pour le nombre de processus d'entrée) d'une file d'attente pour déterminer si cette file d'attente comporte des consommateurs. Une valeur supérieure à 0 indique qu'au moins une application consommatrice est connectée à la file d'attente. Ces files d'attente sont actives. La valeur 0 indique que la file d'attente n'est associée à aucun programme consommateur. Ces files d'attente sont inactives.

Pour une file d'attente en cluster avec plusieurs instances dans un cluster, IBM MQ utilise la propriété de priorité de charge de travail de cluster **CLWLPRTY** de chaque instance de file d'attente pour déterminer à quelles instances envoyer des messages. IBM MQ envoie des messages aux instances disponibles d'une file d'attente dont la valeur **CLWLPRTY** est la plus élevée.

L'exemple de programme de surveillance de file d'attente de cluster active une file d'attente de cluster en définissant la valeur locale **CLWLPRTY** sur 1. Le programme désactive une file d'attente de cluster en définissant sa valeur **CLWLPRTY** sur 0.

La technologie de mise en cluster IBM MQ propage la propriété **CLWLPRTY** mise à jour d'une file d'attente de cluster à tous les gestionnaires de files d'attente pertinents du cluster. Exemple :

- Gestionnaire de files d'attente avec une application connectée qui insère des messages dans la file d'attente.
- Gestionnaire de files d'attente qui possède une file d'attente locale du même nom dans le même cluster.

La propagation est effectuée à l'aide des gestionnaires de files d'attente de référentiel complet du cluster. Les nouveaux messages de la file d'attente de cluster sont dirigés vers les instances ayant la valeur **CLWLPRTY** la plus élevée dans le cluster.

Transfert de messages en file d'attente

La modification dynamique de la valeur de **CLWLPRTY** influence le routage des nouveaux messages. Cette modification dynamique n'affecte pas les messages déjà mis en file d'attente sur une instance de file d'attente sans destinataires connectés, ni les messages qui ont été transmis via le mécanisme d'équilibrage de charge avant qu'une valeur **CLWLPRTY** modifiée ne soit propagée dans le cluster. Par conséquent, les messages restent dans une file d'attente inactive et ne sont pas traités par une application consommatrice. Pour résoudre ce problème, l'exemple de programme de surveillance de file d'attente de cluster est capable d'extraire des messages d'une file d'attente locale sans destinataires et d'envoyer ces messages à des instances distantes de la même file d'attente à laquelle les destinataires sont connectés.

L'exemple de programme de surveillance de file d'attente de cluster transfère des messages d'une file d'attente locale inactive vers une ou plusieurs files d'attente éloignées actives en obtenant des messages (à l'aide de **MQGET**) et insertion de messages (à l'aide de **MQPUT**) dans la même file d'attente en cluster. Ce transfert permet à la gestion de charge de travail du cluster IBM MQ de sélectionner une instance cible différente, en fonction d'une valeur CLWLPRTY supérieure à celle de l'instance de file d'attente locale. La persistance et le contexte des messages sont préservés lors du transfert des messages. L'ordre des messages et les options de liaison ne sont pas conservées.

Planification

L'exemple de programme de surveillance de file d'attente de cluster modifie la configuration du cluster en cas de modification de la connectivité des applications consommatrices. Les modifications sont transmises des gestionnaires de files d'attente dans lesquels l'exemple de programme de surveillance de file d'attente de cluster surveille les files d'attente vers les gestionnaires de files d'attente de référentiel complet du cluster. Les gestionnaires de files d'attente de référentiel complet traitent les mises à jour de configuration et les réappliquent à tous les gestionnaires de files d'attente pertinents du cluster. Les gestionnaires de files d'attente appropriés incluent les gestionnaires de files d'attente qui possèdent des files d'attente en cluster du même nom (où une instance de l'exemple de programme de surveillance des files d'attente de cluster est en cours d'exécution) et tout gestionnaire de files d'attente dans lequel une application a ouvert la file d'attente de cluster pour y insérer des messages au cours des 30 derniers jours.

Les modifications sont traitées de manière asynchrone dans le cluster. Par conséquent, après chaque modification, les différents gestionnaires de files d'attente du cluster peuvent avoir des vues différentes de la configuration pendant un certain temps.

L'exemple de programme de surveillance de file d'attente de cluster ne convient qu'aux systèmes sur lesquels les applications consommatrices sont rarement connectées ou déconnectées ; par exemple, les applications consommatrices à exécution longue. Lorsqu'il est utilisé pour surveiller les systèmes sur lesquels les applications consommatrices ne sont connectées que pour de courtes périodes, le temps d'attente lors de la distribution des mises à jour de configuration peut entraîner une vue incorrecte des gestionnaires de files d'attente du cluster sur les files d'attente sur lesquelles les consommateurs sont connectés. Ce temps d'attente peut entraîner des messages mal acheminés.

Lors de la surveillance de nombreuses files d'attente, un taux de modification relativement faible des consommateurs connectés dans toutes les files d'attente peut augmenter le trafic de configuration du cluster dans le cluster. L'augmentation du trafic de configuration de cluster peut entraîner une charge excessive sur un ou plusieurs des gestionnaires de files d'attente suivants.

- Les gestionnaires de files d'attente dans lesquels l'exemple de programme de surveillance de file d'attente de cluster est en cours d'exécution
- Les gestionnaires de files d'attente du référentiel complet.
- Un gestionnaire de files d'attente avec une application connectée qui insère des messages dans la file d'attente
- Un gestionnaire de files d'attente qui possède une file d'attente locale du même nom dans le même cluster

L'utilisation du processeur sur les gestionnaires de files d'attente de référentiel complet doit être évaluée. Une utilisation supplémentaire du processeur est visible sous forme de trafic de messages dans la file d'attente de référentiel complète SYSTEM.CLUSTER.COMMAND.QUEUE. Si des messages s'accumulent dans cette file d'attente, cela indique que les gestionnaires de files d'attente de référentiel complet ne peuvent pas suivre le taux de changement de configuration du cluster dans le système.

Lorsque de nombreuses files d'attente sont surveillées par l'exemple de programme de surveillance de file d'attente de cluster, le travail effectué par l'exemple de programme et le gestionnaire de files d'attente est important. Ce travail est effectué, même lorsque les consommateurs associés ne sont pas modifiés. L'argument **-i** peut être modifié pour réduire l'utilisation du processeur de l'exemple de programme sur le système local, en diminuant la fréquence du cycle de surveillance.

Pour aider à détecter une activité excessive, l'exemple de programme de surveillance de file d'attente de cluster indique le temps de traitement moyen par intervalle d'interrogation, le temps de traitement écoulé et le nombre de modifications de configuration. Les rapports sont distribués dans un message d'information, **CLM0045I**, toutes les 30 minutes ou tous les 600 intervalles d'interrogation, la date la plus proche étant retenue.

Exigences d'utilisation de la surveillance des files d'attente de

L'exemple de programme de surveillance de file d'attente de cluster comporte des exigences et des restrictions. Vous pouvez modifier l'exemple de code source fourni pour modifier certaines de ces restrictions dans la façon dont il peut être utilisé. Les exemples répertoriés dans cette section détaillent les modifications qui peuvent être apportées.

- L'exemple de programme de surveillance des files d'attente de cluster est conçu pour être utilisé pour surveiller les files d'attente dans lesquelles les applications consommatrices sont connectées ou non connectées. Si le système possède des applications consommatrices qui se connectant et se déconnectant fréquemment, l'exemple de programme peut générer une activité excessive de configuration de cluster sur l'ensemble du cluster. Cela peut avoir un impact sur les performances des gestionnaires de files d'attente du cluster.
- L'exemple de programme de surveillance de file d'attente de cluster dépend du système IBM MQ sous-jacent et de la technologie de cluster. Le nombre de files d'attente surveillées, la fréquence de surveillance et la fréquence de changement de l'état de chaque file d'attente affectent la charge sur le système global. Ces facteurs doivent être pris en compte lors de la sélection des files d'attente à surveiller et de l'intervalle d'interrogation de la surveillance.
- Une instance de l'exemple de programme de surveillance de file d'attente de cluster doit être connectée à chaque gestionnaire de files d'attente du cluster qui possède une instance d'une file d'attente à surveiller. Il n'est pas nécessaire de connecter l'exemple de programme aux gestionnaires de files d'attente du cluster qui ne possèdent pas les files d'attente.
- L'exemple de programme de surveillance de file d'attente de cluster doit être exécuté avec les autorisations appropriées pour accéder à toutes les ressources IBM MQ requises. Exemple :
  - Gestionnaire de files d'attente auquel la connexion doit être établie
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - Toutes les files d'attente à surveiller lors du transfert de messages
- Le serveur de commandes doit être en cours d'exécution pour chaque gestionnaire de files d'attente avec l'exemple de programme de surveillance de file d'attente de cluster connecté.
- Chaque instance de l'exemple de programme de surveillance de file d'attente de cluster nécessite l'utilisation exclusive d'une file d'attente locale (non en cluster) sur le gestionnaire de files d'attente auquel elle est connectée. Cette file d'attente locale permet de contrôler l'exemple de programme et de recevoir des messages de réponse des interrogations effectuées sur le serveur de commandes du gestionnaire de files d'attente.
- Toutes les files d'attente à surveiller par une seule instance de l'exemple de programme de surveillance de file d'attente de cluster doivent se trouver dans le même cluster. Si un gestionnaire de files d'attente comporte des files d'attente dans plusieurs clusters nécessitant une surveillance, plusieurs instances de l'exemple de programme sont requises. Chaque instance a besoin d'une file d'attente locale pour les messages de contrôle et de réponse.
- Toutes les files d'attente à surveiller doivent se trouver dans un seul cluster. Les files d'attente configurées pour utiliser une liste de noms de cluster ne sont pas surveillées.
- L'activation du transfert de messages à partir de files d'attente inactives est facultative. Elle s'applique à toutes les files d'attente surveillées par l'instance de l'exemple de programme de surveillance de file d'attente de cluster. Si seul un sous-ensemble des files d'attente surveillées requiert l'activation du transfert de messages, deux instances de l'exemple de programme de surveillance de file d'attente de cluster sont nécessaires. Dans un exemple de programme, le transfert de message est activé et dans l'autre, le transfert de message est désactivé. Chaque instance de l'exemple de programme a besoin d'une file d'attente locale pour les messages de contrôle et de réponse.
- L'équilibrage de charge de cluster IBM MQ envoie par défaut des messages aux instances des files d'attente de cluster qui résident sur le même gestionnaire de files d'attente auquel une application d'insertion est connectée. Cette option doit être désactivée lorsque la file d'attente locale est inactive dans les cas suivants:
  - Le placement d'applications se connecte à des gestionnaires de files d'attente qui possèdent des instances d'une file d'attente inactive surveillée

– Les messages en file d'attente sont transférés des files d'attente inactives vers les files d'attente actives.

La préférence d'équilibrage de charge locale sur la file d'attente peut être désactivée de manière statique en définissant la valeur CLWLUSEQ sur ANY. Dans cette configuration, les messages placés dans les files d'attente locales sont distribués aux instances de file d'attente locales et distantes afin d'équilibrer la charge de travail, même lorsqu'il existe des applications consommatrices locales. L'exemple de programme de surveillance de file d'attente de cluster peut également être configuré pour définir temporairement la valeur **CLWLUSEQ** sur ANY alors que la file d'attente n'a aucun consommateur connecté, ce qui entraîne uniquement l'envoi de messages locaux aux instances locales d'une file d'attente alors que cette file d'attente est active.

- Le système et les applications IBM MQ ne doivent pas utiliser **CLWLPRTY** pour les files d'attente à surveiller ou les canaux utilisés. Sinon, les actions de l'exemple de programme de surveillance de file d'attente de cluster sur les attributs de file d'attente **CLWLPRTY** peuvent avoir des effets indésirables.
- L'exemple de programme de surveillance de file d'attente de cluster consigne les informations d'exécution dans un ensemble de fichiers de rapport. Un répertoire pour stocker ces rapports est requis et l'exemple de programme de surveillance de file d'attente de cluster doit disposer des droits d'écriture sur ce dernier.

AMQSCLM: Préparation et exécution de l'exemple
L'exemple de surveillance de file d'attente de cluster peut être exécuté en local connecté à un gestionnaire de files d'attente ou en tant que client connecté via un canal. L'exemple doit être exécuté à chaque fois que le gestionnaire de files d'attente est en cours d'exécution. Lorsqu'il est exécuté en local, il peut être configuré en tant que service de gestionnaire de files d'attente pour démarrer et arrêter automatiquement l'exemple avec le gestionnaire de files d'attente.

Avant de commencer

Les étapes suivantes doivent être effectuées avant l'exécution de l'exemple de surveillance de file d'attente de cluster.

1. Créez une file d'attente de travail sur chaque gestionnaire de files d'attente pour l'utilisation interne de l'exemple.

   Chaque instance de l'exemple a besoin d'une file d'attente locale non-cluster pour une utilisation interne exclusive. Vous pouvez choisir le nom de la file d'attente. L'exemple utilise le nom AMQSCLM.CONTROL.QUEUE. Par exemple, sous Windows, vous pouvez créer cette file d'attente à l'aide de la commande **MQSC** suivante:

   ```
   DEFINE QLOCAL(AMQSCLM.CONTROL.QUEUE)
   ```

   Vous pouvez conserver les valeurs par défaut de **MAXDEPTH** et de **MAXMSGL** .

2. Créez un répertoire pour les journaux des erreurs et des messages d'information.

   L'exemple écrit des messages de diagnostic dans des fichiers de rapport. Vous devez choisir un répertoire dans lequel stocker les fichiers. Par exemple, sous Windows, vous pouvez créer un répertoire à l'aide de la commande suivante:

   ```
   mkdir C:\AMQSCLM\rpts
   ```

   Les fichiers de rapport créés par l'exemple ont la convention de dénomination suivante:

   ```
   QmgrName.ClusterName.RPT0n.LOG
   ```

3. (Facultatif) Définissez l'exemple de surveillance de file d'attente de cluster en tant que service IBM MQ .

   Pour surveiller les files d'attente, l'exemple doit toujours être en cours d'exécution. Pour vous assurer que l'exemple de surveillance de file d'attente de cluster est toujours en cours d'exécution, vous

pouvez le définir en tant que service de gestionnaire de files d'attente. La définition de l'exemple en tant que service signifie que AMQSCLM est démarré au démarrage du gestionnaire de files d'attente. Vous pouvez utiliser l'exemple suivant pour définir l'exemple de surveillance de file d'attente de cluster en tant que service IBM MQ .

```
define service(AMQSCLM) +
      descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
      control(qmgr) +
      servtype(server) +
      startcmd('MQ_INSTALLATION_PATH\tools\c\samples\Bin\AMQSCLM.exe') +
      startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l
c:\AMQSCLM\rpts') +
      stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
      stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

Définition	Description
service	Spécifie le nom du service. Vous pouvez choisir le nom du service.
descr	Indique une description textuelle du service.
control	Indique que le service démarre et s'arrête en même temps que le gestionnaire de files d'attente.
servtype	Indique qu'un objet de service serveur, c'est-à-dire une seule instance, peut être exécuté à la fois pour ce gestionnaire de files d'attente.
startcmd	Indique l'emplacement et le nom du programme.
startarg	Indique les arguments de l'exemple. Notez l'utilisation de *+ QMNAME +*. Le nom du gestionnaire de files d'attente est automatiquement remplacé.
stdout	Nom de fichier complet vers lequel la sortie standard est redirigée. L'exemple écrit dans ce fichier uniquement les messages confirmant que l'exemple s'est arrêté. L'exemple effectue cette opération car le fichier d'erreur standard a déjà été fermé à une étape antérieure du processus d'arrêt de l'exemple.
stderr	Nom de fichier qualifié complet vers lequel la sortie d'erreur standard est redirigée. L'exemple écrit dans le fichier d'erreur standard tous les messages d'erreur avant l'arrêt de l'exemple.

Pourquoi et quand exécuter cette tâche

Cette tâche vous permet de démarrer et d'arrêter l'exemple de surveillance de file d'attente de cluster de différentes manières. Il vous permet également d'exécuter l'exemple dans un mode qui génère des fichiers de rapport contenant des informations statistiques sur les files d'attente surveillées.

L'exemple de programme peut être exécuté à l'aide de la commande suivante.

```
AMQSCLM -m QMgrName -c ClusterName (-q QNameMask| -f QListFile) -r MonitorQName
[-l ReportDir] [-t] [-u ActiveVal] [-i Interval] [-d] [-s] [-v]
```

Le tableau répertorie les arguments pouvant être utilisés avec l'exemple de surveillance de file d'attente de cluster, ainsi que des informations supplémentaires sur chacun d'eux.

Argument	Variable	Informations complémentaires
-m	**QMgrName**	Gestionnaire de files d'attente à surveiller.
-c	**ClusterName**	Cluster contenant les files d'attente à surveiller.
-q	**QNameMask**	La ou les files d'attente à surveiller. Un * de fin surveille toutes les files d'attente dont le nom correspond à zéro ou plusieurs caractères de fin.
-f	**QListFile**	Chemin d'accès complet et nom de fichier d'un fichier contenant une liste de noms de file d'attente ou de masques de nom de file d'attente à surveiller. Le fichier doit contenir un nom / masque de file d'attente par ligne. Vous pouvez spécifier **-q** ou **-f**, mais pas les deux.
-r	**MonitorQName**	File d'attente locale utilisée exclusivement par l'échantillon.
-l	**ReportDir**	Chemin de répertoire dans lequel stocker les messages d'information consignés dans un ensemble d'encapsulage[9]Fichiers de rapport.
-t		(Facultatif) Active le transfert des messages en file d'attente des files d'attente locales inactives vers les files d'attente actives. Si cette option n'est pas activée, seuls les nouveaux messages entrant dans le cluster sont routés dynamiquement vers les instances actives d'une file d'attente.
-u	**ActiveVal**	(Facultatif) bascule automatiquement la propriété **CLWLUSEQ** d'une instance de file d'attente surveillée sur ANY lorsqu'elle est inactive, et sur la valeur de **ActiveVal** lorsqu'elle est active. **ActiveVal** peut être LOCAL ou QMGR. Si cet argument n'est pas défini dans un système où les applications d'insertion se connectent au même gestionnaire de files d'attente ou où le transfert de messages est activé, les files d'attente surveillées doivent avoir la valeur **CLWLUSEQ** ANYou QMGR avec le gestionnaire de files d'attente ayant la valeur ANY.
-i	**Interval**	(Facultatif) Intervalle de temps en secondes pendant lequel le moniteur vérifie les files d'attente. La valeur par défaut est 300 secondes (5 minutes).
-d		(Facultatif) Active la sortie de diagnostic supplémentaire. La sortie de débogage peut être utile lors de la configuration initiale du système ou lors de l'utilisation de l'exemple de code.
-s		(Facultatif) Active la sortie statistique minimale par intervalle.
-v		(Facultatif) Consigner les informations de rapport dans `standard out`, en plus des fichiers de rapport.

Exemples de liste d'arguments:

```
-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
```

Exemple de fichier liste de files d'attente:

```
Q1
QUEUE.*
```

[9] Pour chaque combinaison de gestionnaire de files d'attente et de file d'attente, un fichier journal de taille fixe est généré qui, lorsqu'il est saturé, est remplacé. Le consignateur écrit toujours dans le même fichier et conserve également les deux versions précédentes du fichier.

```
ABC
ABD
```

Procédure

1. Démarrez l'exemple de surveillance de file d'attente de cluster. Vous pouvez démarrer l'exemple de l'une des manières suivantes:
   - Utilisez une invite de commande avec les droits utilisateur appropriés.
   - Utilisez la commande MQSC **START SERVICE** si l'exemple est configuré en tant que service IBM MQ .

   La liste des arguments est la même dans les deux cas.

   L'exemple ne démarre pas la surveillance des files d'attente pendant 10 secondes après l'initialisation du programme. Ce délai permet aux applications consommatrices de se connecter d'abord aux files d'attente surveillées, ce qui empêche les modifications inutiles de l'état actif de la file d'attente.
2. Arrêtez l'exemple de surveillance de file d'attente de cluster. L'exemple s'arrête automatiquement lorsque le gestionnaire de files d'attente est arrêté, arrêté, mis au repos ou si la connexion au gestionnaire de files d'attente est interrompue. Vous pouvez arrêter l'exemple sans arrêter le gestionnaire de files d'attente:
   - Configurez la file d'attente locale utilisée exclusivement par l'exemple pour désactiver la fonction Get.
   - Envoyez un message avec le **CorrelId** `"STOP CLUSTER MONITOR\0\0\0\0"`à la file d'attente locale utilisée exclusivement par l'exemple.
   - Mettez fin au processus d'échantillonnage. Cela peut entraîner la perte de messages non persistants transférés vers des files d'attente actives. Il se peut également que la file d'attente locale utilisée par l'exemple soit maintenue ouverte pendant un certain nombre de secondes après l'arrêt. Cette situation empêche une nouvelle instance de l'échantillon de surveillance de file d'attente de cluster de démarrer immédiatement.

   Si l'exemple a été démarré en tant que service IBM MQ , **STOP SERVICE** n'a aucun effet. Il est possible d'utiliser l'une des méthodes d'arrêt décrites comme mécanisme **STOP SERVICE** configuré dans le gestionnaire de files d'attente.

Que faire ensuite

Vérifiez le statut de l'exemple.

Si la génération de rapports est activée, vous pouvez vérifier le statut des fichiers de rapport. Utilisez la commande suivante pour consulter le fichier de rapport le plus récent:

```
QMgrName.ClusterName.RPT01.LOG
```

Pour passer en revue les anciens fichiers de rapport, utilisez les commandes suivantes:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Les fichiers de rapport atteignent une taille maximale d'environ 1 Mo. Lorsque le fichier RPT01 se remplit, un nouveau fichier RPT01 est créé. L'ancien fichier RPT01 est renommé en RPT02. RPT02 est renommé en RPT03. L'ancien RPT03 est supprimé.

L'exemple crée des messages d'information dans les situations suivantes:

- au démarrage
- à la fin
- lorsqu'il marque une file d'attente **ACTIVE** ou **INACTIVE**

- lors de la remise en file d'attente de messages d'une file d'attente inactive vers une ou plusieurs instances actives

L'exemple crée un message d'erreur *CLMnnnnE* pour signaler un incident nécessitant votre attention.

Toutes les 30 minutes, l'exemple indique le temps de traitement moyen par intervalle d'interrogation et le temps de traitement écoulé. Ces informations sont contenues dans le message CLM0045I.

Lorsque les messages statistiques sont activés **-s**, l'exemple fournit les informations statistiques suivantes sur chaque vérification de file d'attente:

- Temps nécessaire au traitement des files d'attente (en millisecondes)
- Nombre de files d'attente vérifiées
- Nombre de modifications actives / inactives effectuées
- Nombre de messages transférés

Ces informations sont signalées dans le message CLM0048I.

Les fichiers de rapport peuvent croître rapidement en mode débogage et être rapidement encapsulés. Dans ce cas, la limite de taille de 1 Mo pour les fichiers individuels peut être dépassée.

AMQSCLM: Traitement des incidents
Les sections suivantes contiennent des informations sur les scénarios qui peuvent être rencontrés lors de l'utilisation de l'exemple. Des informations sur les explications potentielles d'un scénario, ainsi que des options sur la manière de le résoudre, sont fournies.

Scénario: AMQSCLM ne démarre pas

Explication possible: Syntaxe incorrecte.

Action: Vérifiez la syntaxe correcte dans la sortie d'erreur standard

Explication possible: Le gestionnaire de files d'attente n'est pas disponible.

Action: Recherchez l'ID de message dans le fichier de rapport CLM0010E.

Explication potentielle: Impossible d'ouvrir ou de créer un ou plusieurs fichiers de rapport.

Action: Recherchez les messages d'erreur dans la sortie d'erreur standard lors de l'initialisation.

Scénario: AMQSCLM ne modifie pas une file d'attente en ACTIVE ou INACTIVE

Explication potentielle: La file d'attente ne figure pas dans la liste des files d'attente à surveiller

Action: Vérifiez les valeurs des paramètres **-q** et **-f** .

Explication possible: La file d'attente n'est pas une file d'attente locale dans le cluster approprié.

Action: Vérifiez que la file d'attente est locale et qu'elle se trouve dans le cluster approprié.

Explication potentielle: AMQSCLM n'est pas en cours d'exécution pour ce gestionnaire de files d'attente et ce cluster.

Action: Démarrez AMQSCLM pour le gestionnaire de files d'attente et le cluster appropriés.

Explication potentielle: La file d'attente est laissée INACTIVE, **CLWLPRTY** = 0, car elle n'a pas de consommateurs. Sinon, il reste ACTIVE **CLWLPRTY** > =1, car il a au moins 1 consommateur.

Action: Vérifiez si les applications consommatrices sont connectées à la file d'attente.

Explication potentielle: Le serveur de commandes du gestionnaire de files d'attente n'est pas en cours d'exécution.

Action: Recherchez les erreurs dans les fichiers de rapport.

Scénario: Les messages ne sont pas acheminés autour des files d'attente INACTIVE

Explication potentielle: Les messages sont placés directement dans le gestionnaire de files d'attente qui possède la file d'attente inactive et la valeur **CLWLUSEQ** de la file d'attente n'est pas ANYet l'argument **-u** n'est pas utilisé pour AMQSCLM.

Action: Vérifiez la valeur **CLWLUSEQ** du gestionnaire de files d'attente approprié ou assurez-vous que l'argument **-u** est utilisé pour AMQSCLM.

Explication possible: Aucune file d'attente n'est active dans les gestionnaires de files d'attente. La charge de travail des messages est équilibrée entre toutes les files d'attente inactives jusqu'à ce qu'une file d'attente devienne active.

Action: Vérifiez le statut des files d'attente sur tous les gestionnaires de files d'attente.

Explication potentielle: Les messages sont insérés dans un gestionnaire de files d'attente différent dans le cluster de celui qui possède la file d'attente inactive et la valeur mise à jour **CLWLPRTY** 0 n'est pas propagée au gestionnaire de files d'attente de l'application d'insertion.

Action: Vérifiez que les canaux de cluster entre le gestionnaire de files d'attente surveillé et le gestionnaire de files d'attente de référentiel complet sont en cours d'exécution. Vérifiez que les canaux entre le gestionnaire de files d'attente d'insertion et le gestionnaire de files d'attente de référentiel complet sont en cours d'exécution. Consultez les journaux d'erreurs des gestionnaires de files d'attente de référentiel surveillés, en cours d'insertion et complets.

Explication possible: Les instances de file d'attente éloignée sont actives (CLWLPRTY=1), mais les messages ne peuvent pas être acheminés vers ces instances de file d'attente car le canal émetteur de cluster du gestionnaire de files d'attente local n'est pas en cours d'exécution.

Action: Vérifiez le statut des canaux émetteurs de cluster du gestionnaire de files d'attente local vers le ou les gestionnaires de files d'attente éloignées avec une instance active de la file d'attente.

Scénario: AMQSCLM ne transfère pas les messages d'une file d'attente inactive

Explication potentielle: Le transfert de message n'est pas activé (**-t**).

Action: Vérifiez que le transfert de message est activé (**-t**).

Explication potentielle: La file d'attente ne figure pas dans la liste des files d'attente à surveiller.

Action: Vérifiez les valeurs des paramètres **-q** et **-f** .

Explication potentielle: AMQSCLM n'est pas en cours d'exécution pour ce gestionnaire de files d'attente ou d'autres gestionnaires de files d'attente du cluster, qui possèdent des instances de la même file d'attente.

Action: Démarrez AMQSCLM.

Explication potentielle: La file d'attente a **CLWLUSEQ** = LOCAL ou **CLWLUSEQ** = QMGRet l'argument **-u** n'est pas défini.

Action: Définissez le paramètre **-u** ou modifiez la file d'attente ou la configuration du gestionnaire de files d'attente sur ANY.

Explication potentielle: Il n'existe aucune instance active de la file d'attente dans le cluster.

Action: Recherchez les instances de la file d'attente dont la valeur **CLWLPRTY** est supérieure ou égale à 1.

Explication potentielle: Les instances de file d'attente distante ont des consommateurs (**IPPROCS** > = 1) mais sont inactives sur ces gestionnaires de files d'attente (**CLWLPRTY** = 0) car AMQSCLM ne surveille pas ces instances distantes.

Action: Vérifiez que AMQSCLM est en cours d'exécution sur ces gestionnaires de files d'attente et / ou que la file d'attente figure dans la liste des files d'attente à surveiller en vérifiant les valeurs des paramètres **-q** et **-f** .

Explication potentielle: Les instances de file d'attente éloignée sont actives (**CLWLPRTY** = 1), mais sont considérées comme inactives sur le gestionnaire de files d'attente local (**CLWLPRTY** = 0). Cette situation est due au fait que la valeur **CLWLPRTY** mise à jour n'est pas propagée à ce gestionnaire de files d'attente.

Action: Vérifiez que les gestionnaires de files d'attente éloignées sont connectés à au moins un des gestionnaires de files d'attente de référentiel complet du cluster. Vérifiez que les gestionnaires de files d'attente de référentiel complet fonctionnent correctement. Vérifiez que les canaux entre les gestionnaires de files d'attente de référentiel complet et les gestionnaires de files d'attente surveillés sont en cours d'exécution.

Explication potentielle: Les messages ne sont pas validés et ne peuvent donc pas être extraits.

Action: Vérifiez que l'application émettrice fonctionne correctement.

Explication potentielle: AMQSCLM n'a pas accès à la file d'attente locale dans laquelle les messages sont mis en file d'attente.

Action: Vérifiez si AMQSCLM s'exécute en tant qu'utilisateur disposant des droits suffisants pour accéder à la file d'attente.

Explication potentielle: Le serveur de commandes du gestionnaire de files d'attente n'est pas en cours d'exécution.

Action: Démarrez le serveur de commandes du gestionnaire de files d'attente.

Explication potentielle: AMQSCLM a rencontré une erreur.

Action: Recherchez les erreurs dans les fichiers de rapport.

Explication possible: Les instances de file d'attente éloignée sont actives (CLWLPRTY=1), mais les messages ne peuvent pas être transférés vers ces instances de file d'attente car le canal émetteur de cluster du gestionnaire de files d'attente local n'est pas en cours d'exécution. Ceci est souvent accompagné d'un avertissement CLM0030W dans le journal de rapport amqsclm.

Action: Vérifiez le statut des canaux émetteurs de cluster du gestionnaire de files d'attente local vers le ou les gestionnaires de files d'attente éloignées avec une instance active de la file d'attente.

ALW *Exemple de programme de recherche de noeud final de connexion (CEPL)*

L'exemple IBM MQ Connection Endpoint Lookup fournit un module d'exit simple mais puissant qui permet aux utilisateurs IBM MQ d'extraire des définitions de connexion d'un référentiel LDAP tel que Tivoli Directory Server.

Tivoli Directory Server v6.3 Client doit être installé pour utiliser CEPL.

Une connaissance pratique de l'administration de IBM MQ sur les plateformes prises en charge est requise pour utiliser cet exemple.

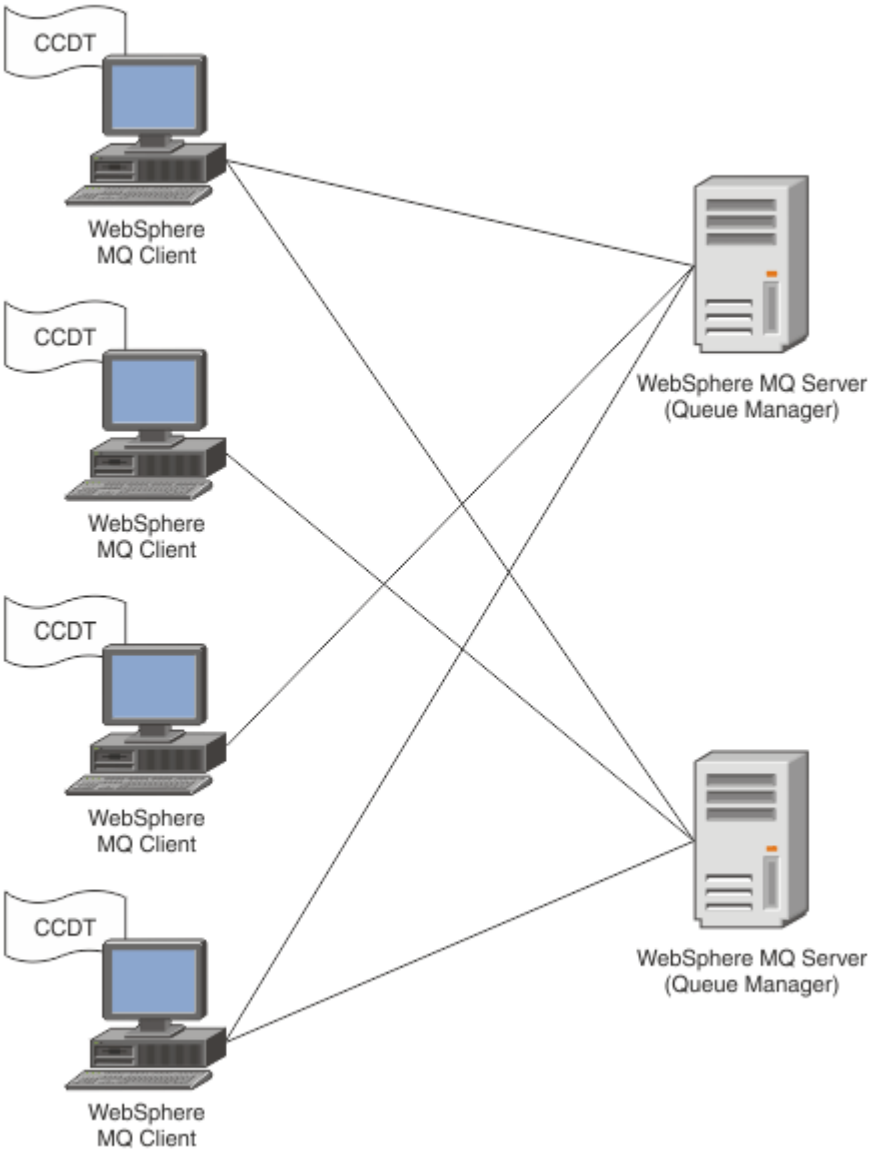
Windows Linux AIX *Introduction*

Configurez un référentiel global, par exemple un annuaire LDAP (Lightweight Directory Access Protocol), pour stocker les définitions de connexion client afin d'aider à la maintenance et à l'administration.

Utilisation d'une application client IBM MQ pour établir une connexion à un gestionnaire de files d'attente via une table de définition de canal du client (CCDT).

La table de définition de canal du client est créée via l'interface d'administration IBM MQ MQSC standard. L'utilisateur doit être connecté à un gestionnaire de files d'attente afin de créer des définitions de connexion client, même si les données contenues dans la définition ne sont pas limitées au gestionnaire

de files d'attente. Le fichier CCDT généré doit être distribué manuellement entre les machines client et les



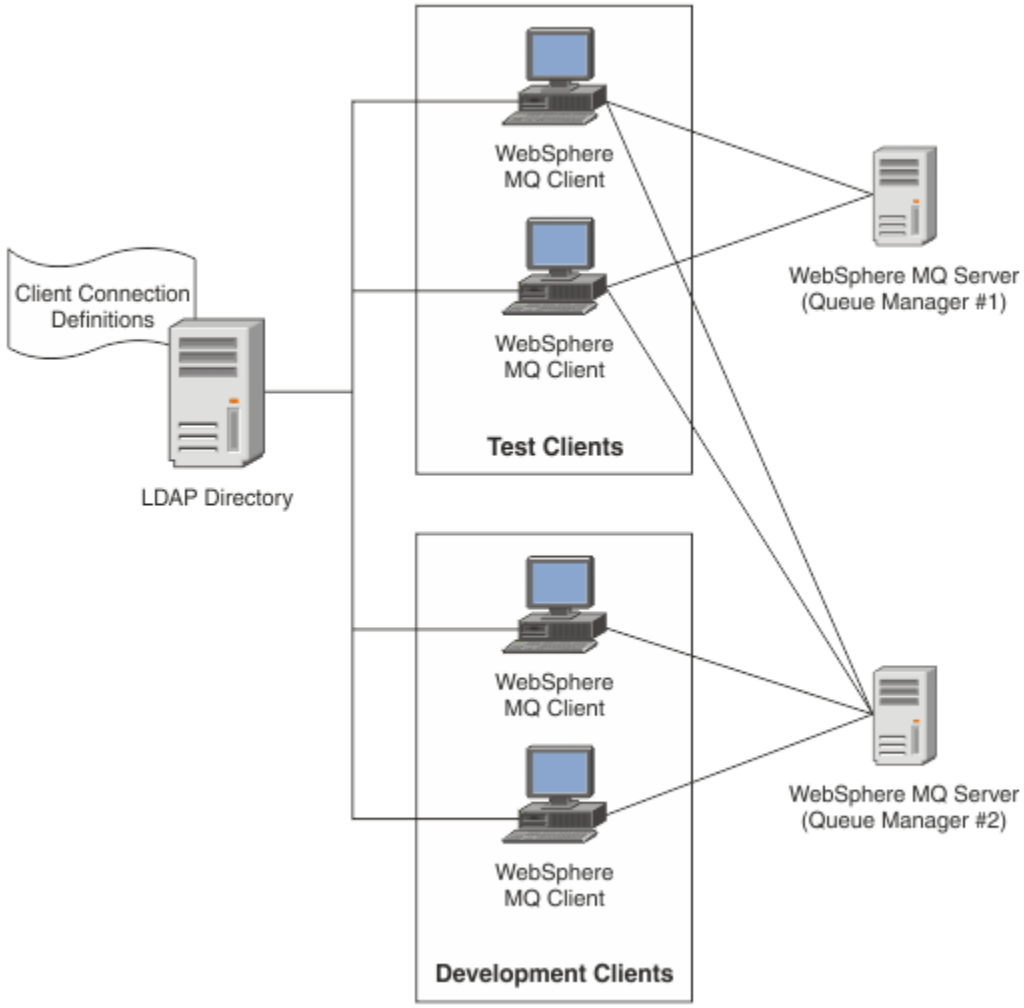
applications.

Le fichier CCDT doit être distribué à chaque client IBM MQ . Là où des milliers de clients peuvent exister localement ou globalement, il serait bientôt difficile de les maintenir et de les administrer. Une approche plus souple est nécessaire pour s'assurer que chaque client dispose des définitions de client appropriées.

L'une de ces approches consiste à stocker les définitions de connexion client dans un référentiel global tel qu'un annuaire LDAP (Lightweight Directory Access Protocol). Un annuaire LDAP peut également fournir des fonctions de sécurité, d'indexation et de recherche supplémentaires, ce qui permet à chaque client d'accéder uniquement aux définitions de connexion qui lui appartiennent.

L'annuaire LDAP peut être configuré de sorte que seules des définitions spécifiques soient disponibles pour certains groupes d'utilisateurs. Par exemple, les clients de test peuvent accéder au gestionnaire de files d'attente #1 et au gestionnaire de files d'attente #2, tandis que

les clients de développement ne peuvent accéder qu'au gestionnaire de files d'attente #2 .



Le module d'exit peut rechercher un référentiel LDAP, par exemple IBM Tivoli Directory Server, pour extraire des définitions de canal. A l'aide de ces définitions de connexion, une application client IBM MQ peut établir une connexion à un gestionnaire de files d'attente.

Le module d'exit est un module d'exit de préconnexion qui permet d'obtenir une définition de canal lors de l'appel MQCONN/MQCONNX à partir d'un référentiel LDAP.

Le module d'exit et le schéma peuvent être implémentés par:

- Les clients qui ont déjà créé une base de compétences à l'aide de la technologie existante basée sur les fichiers CCDT et qui souhaitent réduire les coûts d'administration et de distribution.
- Clients existants qui utilisent déjà leur propre technologie de propriété pour la distribution des définitions de connexion client.
- Clients nouveaux ou existants qui n'utilisent actuellement aucun type de solution de connexion client et qui souhaitent utiliser les fonctions offertes par IBM MQ.
- Clients nouveaux ou existants qui souhaitent utiliser directement ou optimiser leur modèle de messagerie en ligne avec l'architecture métier LDAP en cours.

ALW *Environnements pris en charge*

Vérifiez que vous disposez d'un système d'exploitation pris en charge et des logiciels appropriés avant d'exécuter l'exemple Recherche de noeud final de connexion.

L'exemple de programme de recherche de noeud final IBM MQ Connection requiert les logiciels suivants:

- IBM WebSphere MQ 7.0ou version ultérieure

- Tivoli Directory Server V6.3 Client ou version ultérieure

Systèmes d'exploitation pris en charge:

1. Windows (7/8/2008/2012)
2. AIX
3. Linux
   - RHEL v4 et v5 sous System p
   - SUSE v9 et v10 sur System p
   - RHEL v4 et v5 x86-64 32 bits et 64 bits
   - SUSE v9 et v10 x86-64 32 bits et 64 bits

Remarque : L'exemple n'est pas disponible pour les plateformes suivantes:

- z/OS
- IBM i

ALW *Installation et configuration*
Installation et configuration du module d'exit et du schéma de noeud final de connexion.

Installation du module d'exit

Lors de l'installation de IBM MQ, le module d'exit est installé sous `tools/samples/c/preconnexit/bin`. Pour les plateformes 32 bits, le module d'exit doit être copié dans `exit/`*`installation_name`*`/` pour pouvoir être utilisé. Pour les plateformes 64 bits, le module d'exit doit être copié dans exit64 / *nom_installation*/ avant de pouvoir être utilisé.

Installation du schéma de noeud final de connexion

L'exit utilise le schéma de noeud final de connexion, `ibm-amq.schema`. Le fichier schéma doit être importé sur n'importe quel serveur LDAP pour que l'exit puisse être utilisé. Après l'importation du schéma, les valeurs des attributs doivent être ajoutées.

Voici un exemple d'importation du schéma de noeud final de connexion. L'exemple suppose que IBM Tivoli Directory Server (ITDS) est en cours d'utilisation.

- Vérifiez que IBM Tivoli Directory Server est en cours d'exécution, puis copiez ou envoyez par FTP le fichier `ibm-amq.schema` sur le serveur ITDS.
- Sur le serveur ITDS, entrez la commande suivante pour installer le schéma dans le magasin ITDS, où *ID LDAP* et *mot de passe LDAP* sont le nom distinctif et le mot de passe root du serveur LDAP:

  `ldapadd -D "`*`LDAP ID`*`" -w "`*`LDAP password`*`" -f ibm-amq.schema`
- Dans une fenêtre de commande, entrez la commande suivante ou utilisez un outil tiers pour parcourir le schéma à des fins de vérification:

  ```
  ldapsearch objectclass=ibm-amqClientConnection
  ```

Consultez la documentation de votre serveur LDAP pour plus de détails sur l'importation du fichier de schéma.

Configuration

Une nouvelle section appelée PreConnect doit être ajoutée au fichier de configuration du client, par exemple `mqclient.ini`. La section PreConnect contient les mots clés suivants:

Module
Nom du module contenant le code d'exit d'API. Si cette zone contient le chemin d'accès complet du module, elle est utilisée telle qu'elle est. Sinon, la recherche est effectuée dans le dossier `exit` ou `exit64` de l'installation IBM MQ .

Fonction
Nom du point d'entrée fonctionnel dans la bibliothèque qui contient le code d'exit `LdapPreConnect` . La définition de fonction est conforme au prototype de fonction de votre entreprise.

Avertissement : Vous devez supprimer les guillemets dans l'instruction de fonction lorsque vous indiquez votre point d'entrée d'exit réel.

Data
URI du référentiel LDAP contenant les définitions de canal.

Le fragment suivant est un exemple des modifications requises dans le fichier `mqclient.ini` .

```
PreConnect:
Module=amqlcelp
Function="LdapPreconnectExit"
Data=ldap:dap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ALW *Présentation de l'exit et du schéma*

Syntaxe et paramètres utilisés pour établir une connexion à un gestionnaire de files d'attente.

IBM MQ 9.3 définit la syntaxe suivante pour un point d'entrée dans un module d'exit.

```
void MQENTRY MQ_PRECONNECT_EXIT ( PMQNXP pExitParms
                                , PMQCHAR pQMgrName
                                , PPMQCNO ppConnectOpts
                                , PMQLONG pCompCode
                                , PMQLONG pReason)
```

Lors de l'exécution de l'appel MQCONN/X, le client IBM MQ C charge le module d'exit contenant une implémentation de la syntaxe de la fonction. Il appelle ensuite une fonction d'exit pour extraire les définitions de canal. Les définitions de canal extraites sont ensuite utilisées pour établir une connexion à un gestionnaire de files d'attente.

Paramètres

Paramètres pExit
Type: entrée / sortie PMQNXP

Structure du paramètre d'exit PreConnection . La structure est allouée et gérée par l'appelant de l'exit.

```
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 */
};
```

pQMgrNom
Type: entrée / sortie PMQCHAR

Nom du gestionnaire de files d'attente. En entrée, ce paramètre est la chaîne de filtrage fournie à l'appel de l'API MQCONN via le paramètre **QMgrName** . Cette zone peut être vide, explicite ou contenir certains caractères génériques. La zone est modifiée par l'exit. Le paramètre est NULL lorsque l'exit est appelé avec MQXR_TERM.

ppConnectOpts

Type: ppConnectOpts en entrée / sortie

Options qui contrôlent l'action de MQCONNX. Il s'agit d'un pointeur vers une structure d'options de connexion MQCNO qui contrôle l'action de l'appel de l'API MQCONN. Le paramètre est NULL lorsque l'exit est appelé avec MQXR_TERM. Le client MQI fournit toujours une structure MQCNO à l'exit, même s'il n'a pas été fourni à l'origine par l'application. Si une application fournit une structure MQCNO, le client effectue un doublon pour la transmettre à l'exit où elle est modifiée. Le client conserve la propriété du MQCNO. Un MQCD référencé via le MQCNO est prioritaire sur toute définition de connexion fournie via le tableau. Le client utilise la structure MQCNO pour se connecter au gestionnaire de files d'attente et les autres sont ignorés.

Code pComp

Type: entrée / sortie PMQLONG

Code achèvement. Pointeur vers un MQLONG qui reçoit le code achèvement des exits. Il doit s'agir de l'une des valeurs suivantes:

- MQCC_OK -L'exécution a abouti
- MQCC_WARNING -Avertissement (achèvement partiel)
- MQCC_FAILED -Echec de l'appel

pReason

Type: entrée / sortie PMQLONG

Motif qualifiant le code pComp. Pointeur vers un MQLONG qui reçoit le code anomalie de l'exit. Si le code achèvement est MQCC_OK, la seule valeur valide est: MQRC_NONE -(0, x'000') Aucune raison de signaler.

Si le code achèvement est MQCC_FAILED ou MQCC_WARNING, la fonction d'exit peut définir la zone de code anomalie sur n'importe quelle valeur MQRC_ * valide.

ALW *Informations de contexte LDAP MQ*

L'exit utilise la structure de données suivante pour les informations de contexte.

MQNLDAPCTX

La structure MQNLDAPCTX possède le prototype C suivant.

```
typedef struct tagMQNLDAPCTX MQNLDAPCTX;
typedef MQNLDAPCTX MQPOINTER PMQNLDAPCTX;

struct tagMQNLDAPCTX
{
  MQCHAR4      StrucId;          /* Structure identifier */
  MQLONG       Version;          /* Structure version number */
  LDAP *       objectDirectory   /* LDAP Instance */
  MQLONG       ldapVersion;      /* Which LDAP version to use? */
  MQLONG       port;             /* Port number for LDAP server*/
  MQLONG       sizeLimit;        /* Size limit */
  MQBOOL       ssl;              /* SSL enabled? */
  MQCHAR *     host;             /* Hostname of LDAP server */
  MQCHAR *     password;         /* Password of LDAP server */
  MQCHAR *     searchFilter;     /* LDAP search filter */
  MQCHAR *     baseDN;           /* Base Distinguished Name */
  MQCHAR *     charSet;          /* Character set */
};
```

Windows Linux AIX *Exemple de code pour la génération de l'exit de recherche de noeud final de connexion*

Vous pouvez utiliser les exemples de fragments de code pour compiler la source sous AIX, Linuxou Windows.

Compilation de la source

Vous pouvez compiler la source avec n'importe quelle bibliothèque client LDAP, par exemple IBM Tivoli Directory Server V6.3 . Cette documentation suppose que vous utilisez les bibliothèques client Tivoli Directory Server V6.3 .

Remarque : La bibliothèque d'exit de préconnexion est prise en charge avec les serveurs LDAP suivants:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Les fragments de code suivants décrivent comment compiler les exits:

Windows **Compilation de l'exit sur la plateforme Windows**

Vous pouvez utiliser le fragment suivant pour compiler la source d'exit:

```
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
```

Remarque : Si vous utilisez les bibliothèques client IBM Tivoli Directory Server V6.3 qui sont compilées avec le compilateur Microsoft Visual Studio 2003 , des avertissements peuvent s'afficher lorsque vous compilez les bibliothèques client IBM Tivoli Directory Server V6.3 avec le compilateur Microsoft Visual Studio 2012ou version ultérieure.

Linux AIX **Compilation de l'exit sous AIX, Linux**

Le fragment de code suivant permet de compiler la source d'exit sous Linux. Certaines options de compilation peuvent différer sous AIX.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server est livré avec des bibliothèques de liens statiques et dynamiques, mais vous ne pouvez utiliser qu'un seul type de bibliothèque. Ce script suppose que vous utilisez les bibliothèques statiques.

```
#Use static libraries.
LDLIBS=-L$(TDSL) -libibmldapstatic

CFLAGS=-I. -I$(MQMI) -I$(TDSI)

all:amqlcepl
```

```
amqlcepl: amqlcel0.c
$(CC) -o cepl amqlcel0.c -shared -fPIC $(XFLAG) $(CFLAGS) $(LDLIBS)
```

ALW *Appel du module d'exit PreConnect*

Le module d'exit PreConnect peut être appelé avec trois codes anomalie différents: le code anomalie MQXR_INIT pour l'initialisation et l'établissement d'une connexion à un serveur LDAP, le code anomalie MQXR_PRECONNECT pour l'extraction des définitions de canal à partir d'un serveur LDAP ou le code anomalie MQXR_TERM lorsque l'exit doit être nettoyé.

MQXR_INIT

L'exit est appelé avec le code anomalie MQXR_INIT pour l'initialisation et l'établissement d'une connexion à un serveur LDAP.

Avant l'appel MQXR_INIT, la zone pExitDataPtr de la structure MQNXP est remplie avec l'attribut Data de la section PreConnect dans le fichier `mqclient.ini` (c'est-à-dire LDAP).

Une URL LDAP se compose au moins du protocole, du nom d'hôte, du numéro de port et du nom distinctif de base pour la recherche. L'exit analyse l'URL LDAP contenue dans la zone pExitDataPtr , alloue une structure de contexte de recherche LDAP MQNLDAPCTX et la remplit en conséquence. L'adresse de cette structure est stockée dans la zone Ptr pExitUserArea. Si l'URL LDAP n'est pas correctement analysée, l'erreur MQCC_FAILED se produit.

A ce stade, l'exit se connecte et se lie au serveur LDAP à l'aide des paramètres **MQNLDAPCTX** . Les descripteurs d'API LDAP résultants sont également stockés dans cette structure.

MQXR_PRECONNECT

Le module d'exit est appelé avec le code anomalie MQXR_PRECONNECT pour l'extraction des définitions de canal à partir d'un serveur LDAP.

L'exit recherche sur le serveur LDAP les définitions de canal correspondant au filtre indiqué. Si **QMgrNameparameter** contient un nom de gestionnaire de files d'attente spécifique, la recherche renvoie toutes les définitions de canal pour lesquelles la valeur d'attribut LDAP **ibm-amqQueueManagerName** correspond au nom de gestionnaire de files d'attente donné.

Si le paramètre **QMgrName** est'*'ou' '(vide), la recherche renvoie toutes les définitions de canal pour lesquelles l'attribut de noeud final **ibm-amqIsClientDefault Connection** est défini sur TRUE.

Après une recherche réussie, l'exit prépare une ou plusieurs définitions MQCD et les renvoie à l'appelant.

MQXR_TERM

L'exit est appelé avec ce code anomalie lorsque l'exit doit être nettoyé. Lors de ce nettoyage, l'exit se déconnecte du serveur LDAP et libère toute la mémoire allouée et gérée par l'exit, y compris la structure MQNLDAPCTX, le tableau de pointeurs et toutes les références MQCD. Toutes les autres zones sont définies sur les valeurs par défaut. Les paramètres d'exit **pQMgrName** et **ppConnectOpts** ne sont pas utilisés lors d'un exit avec le code anomalie MQXR_TERM et peuvent être NULL.

Référence associée

Section PreConnect du fichier de configuration du client

ALW *Schémas LDAP*

Les données de connexion client sont stockées dans un référentiel global appelé annuaire LDAP (Lightweight Directory Access Protocol). Un client IBM MQ utilise un annuaire LDAP pour obtenir les définitions de connexion. La structure des définitions de connexion client IBM MQ dans l'annuaire LDAP est appelée schéma LDAP. Un schéma LDAP est une collection de définitions de type d'attribut, de définitions de classe d'objet et d'autres informations qu'un serveur utilise pour déterminer si une vérification de filtre ou de valeur d'attribut correspond aux attributs d'une entrée et pour autoriser, ajouter et modifier des opérations.

Stockage des données dans l'annuaire LDAP

Les définitions de connexion client se trouvent sous une branche spécifique de l'arborescence de répertoires appelée point de connexion. Comme tous les autres noeuds d'un annuaire LDAP, un nom

distinctif (DN) est associé au point de connexion. Vous pouvez utiliser ce noeud comme point de départ pour toutes les requêtes que vous effectuez sur le répertoire. Utilisez le filtrage lors de l'interrogation de l'annuaire LDAP pour renvoyer un sous-ensemble de définitions de connexion client. Vous pouvez restreindre l'accès aux sous-arborescences en fonction des droits accordés dans d'autres parties de l'arborescence de répertoires, par exemple, aux utilisateurs, aux services ou aux groupes.

Définition de vos propres attributs et classes

Stockez la définition de canal du client en modifiant le schéma LDAP. Toutes les définitions de données LDAP requièrent des objets et des attributs. Les objets et les attributs sont identifiés par un numéro d'ID objet (OID) qui identifie de manière unique l'objet ou l'attribut. Toutes les classes d'un schéma LDAP héritent directement ou indirectement de l'objet supérieur. L'objet de définition de canal du client contient les attributs de l'objet supérieur. Toutes les définitions de données LDAP requièrent des objets et des attributs:

- Les définitions d'objet sont des collections d'attributs LDAP.
- Les attributs sont des types de données LDAP.

La description de chaque attribut et la façon dont ils sont mappés aux propriétés IBM MQ normales sont décrites dans la rubrique Attributs LDAP.

ALW *Attributs LDAP*

Les attributs LDAP définis sont spécifiques à IBM MQ et sont mappés directement aux propriétés de connexion client.

Attributs de chaîne de répertoire de canal du client IBM MQ

Les attributs de chaîne de caractères avec leur mappage aux propriétés IBM MQ sont répertoriés dans le tableau suivant. Les attributs peuvent contenir les valeurs de la syntaxe directoryString (Unicode codé enUTF-8 , c'est-à-dire un système de codage d'octets de variable incluant la syntaxe IA5/ASCII en tant que sous-ensemble). La syntaxe est spécifiée par son numéro d'identification d'objet (OID).

Tableau 166. Attributs de chaîne de répertoire de canal du client IBM MQ

Attribut LDAP	Description	IBM MQ Propriété
CN	Nom usuel composé du nom du canal et du nom du gestionnaire de files d'attente de définition.	
ibm-amqChannelNom	Nom de la définition de canal.	Canal
ibm-amqConnectionNom	Identificateur de la connexion de communication.	CONNAME
ibm-amqDescription	Description du canal.	DESCR
ibm-amqLocalAdresse	Adresse de communication locale du canal.	LOCLADDR
ibm-amqModeNom	Nom du mode LU 6.2.	MODENAME
ibm-amqPassword	Mot de passe pouvant être utilisé.	PASSWORD
ibm-amqQueueManagerName	Nom du gestionnaire de files d'attente ou du groupe de gestionnaires de files d'attente auquel une application client IBM MQ peut demander une connexion.	QMNAME
ibm-amqSecurityExitUserDonnées	Données utilisateur transmises à l'exit de sécurité.	SCYDATA
ibm-amqSecurityExitName	Nom du programme d'exit à exécuter par l'exit de sécurité du canal.	SCYEXIT
ibm-amqSslCipherSpec	Un CipherSpec unique pour une connexion TLS.	SSLCIPH

Tableau 166. Attributs de chaîne de répertoire de canal du client IBM MQ (suite)		
Attribut LDAP	**Description**	**IBM MQ Propriété**
ibm-amqSslPeerName	Vérifie le nom distinctif (DN) du certificat du gestionnaire de files d'attente ou du client homologue à l'autre extrémité d'un canal IBM MQ .	SSLPEER
ibm-amqTransactionProgramName	Nom du programme de transaction.	TPNAME
ibm-amqUserID	ID utilisateur à utiliser par l'agent MCA lors de la tentative de lancement d'une session SNA sécurisée avec un agent MCA éloigné.	USERID

Attributs de nombre entier de connexions client IBM MQ

Les attributs avec des valeurs prédéfinies (par exemple, un type énuméré) sont stockés en tant qu'entiers standard. Ces valeurs sont stockées dans l'annuaire LDAP en tant que valeurs entières et non en utilisant le nom de constante associé.

Tableau 167. Attributs d'entier du répertoire de canal du client IBM MQ		
Attribut LDAP	**Description**	**IBM MQ Propriété**
ibm-amqConnectionAffinité	Détermine si les applications client, qui se connectent plusieurs fois via le même nom de gestionnaire de files d'attente, utilisent le même canal client.	AFFINITY
ibm-amqClientChannelWeight	Pondération permettant d'influencer la définition de canal de connexion client utilisée.	CLNTWGHT
ibm-amqHeartBeatInterval	Délai, en secondes, entre les flux de pulsations transmis par un agent MCA lorsque la file d'attente de transmission ne contient pas de messages.	HBINT
ibm-amqKeepAliveInterval	Valeur de délai d'attente pour un canal.	KAINT
ibm-amqMaximumMessageLength	Longueur maximale d'un message qui peut être transmis sur le canal.	MAXMSGL
Conversations ibm-amqSharing	Nombre maximal de conversations qui partagent chaque instance de canal TCP/IP.	SHARECNV
ibm-amqTransportType	Type de transport à utiliser.	TRPTYPE

Attribut booléen de canal client IBM MQ

Cet attribut booléen n'est mappé à aucune propriété IBM MQ . La syntaxe de cet attribut indique une valeur booléenne.

Tableau 168. Attribut booléen de canal client IBM MQ	
Attribut LDAP	**Description**
ibm-amqIsClientDefault	Cet attribut booléen est défini pour résoudre le problème de recherche d'entrées dont l'attribut ibm-amqQueueManagerName n'a pas été défini.

Attributs de liste de canaux client IBM MQ

Les propriétés IBM MQ sont stockées sous forme d'attribut de liste à valeur unique, séparés par des virgules, dans l'annuaire LDAP. Les attributs sont définis de la même manière que les autres attributs

de chaîne de répertoire. Les attributs de liste ainsi que leur mappage aux propriétés IBM MQ sont décrits dans le tableau suivant.

Tableau 169. Attributs de liste de canaux client IBM MQ

Attribut LDAP	Description	IBM MQ Propriété
ibm-amqHeaderCompression	Liste des techniques de compression de données d'en-tête prises en charge par le canal.	COMPHDR
ibm-amqMessageCompression	Liste des techniques de compression de données de message prises en charge par le canal.	COMPMSG
ibm-amqSendExitUserDonnées	Données utilisateur transmises à l'exit d'émission.	SENDDATA
ibm-amqSendExitUserNom	Nom du programme d'exit à exécuter par l'exit d'émission de canal.	SENDEXIT
ibm-amqReceiveExitUserDonnées	Données utilisateur transmises à l'exit de réception.	RCVDATA
ibm-amqReceiveExitName	Nom du programme d'exit utilisateur à exécuter par l'exit utilisateur de réception de canal.	RCVEXIT

ALW *Nom CN*

Le nom usuel (CN) se compose du nom du canal et du nom du gestionnaire de files d'attente de définition.

Il s'agit d'un attribut préexistant.

Le format du CN est le suivant:

```
CN=CHANNEL_NAME(DEFINING_Q_MGR_NAME)
```

Exemple :

```
CN=TC1(QM_T1)
```

Vous ne pouvez spécifier qu'une seule valeur pour cet attribut.

Cet attribut est un attribut de chaîne et les valeurs ne sont pas sensibles à la casse. La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche, à l'aide d'une sous-chaîne (par exemple, CN=jim * où CN est un attribut) et contient un ou plusieurs caractères génériques.

ALW *Nom ibm-amqChannel*

Cet attribut indique le nom de la définition de canal.

Cet attribut possède une valeur de chaîne unique avec un maximum de 20 caractères qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche, à l'aide d'une sous-chaîne et qui contient un ou plusieurs caractères génériques.

ALW *ibm-amqDescription*

Cet attribut LDAP fournit la description du canal.

Cet attribut possède une valeur de chaîne unique d'un maximum de 64 octets, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqConnectionNom*
Cet attribut LDAP correspond à l'identificateur de la connexion de communication. Il indique les liaisons de communication particulières à utiliser par ce canal.

Cet attribut possède une valeur de chaîne unique d'un maximum de 264 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqLocalAdresse*
Cet attribut indique l'adresse de communication locale du canal.

Cet attribut a une valeur de chaîne unique avec un maximum de 48 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *Nom ibm-amqMode*
Cet attribut doit être utilisé pour les connexions LU 6.2. Il apporte une définition supplémentaire aux caractéristiques de session de la connexion lorsqu'une allocation de session de communication est effectuée.

Cet attribut a une valeur de chaîne unique de 8 caractères exactement, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqPassword*
Cet attribut LDAP indique un mot de passe qui peut être utilisé par l'agent MCA lors de la tentative de lancement d'une session LU 6.2 sécurisée avec un agent MCA distant.

Cet attribut a une valeur entière unique avec un maximum de 12 chiffres. Il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqQueueManagerName*
Cet attribut indique le nom du gestionnaire de files d'attente ou du groupe de gestionnaires de files d'attente auquel une application client IBM MQ peut demander une connexion.

Cet attribut a une valeur de chaîne unique avec un maximum de 48 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

Référence associée

«ibm-amqIsClientDefault», à la page 1192
Cet attribut booléen résout le problème de recherche d'entrées dans lesquelles l'attribut ibm-amqQueueManagerName n'a pas été défini.

ALW *ibm-amqSecurityExitUserDonnées*
Cet attribut LDAP spécifie les données utilisateur qui sont transmises à l'exit de sécurité.

Cet attribut possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqSecurityExitName*
Cet attribut LDAP indique le nom du programme d'exit à exécuter par l'exit de sécurité de canal.

Laissez cette zone vide si aucun exit de sécurité de canal n'est actif.

Cet attribut possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Cet attribut n'est pas préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqSslCipherSpec*
Cet attribut LDAP spécifie un CipherSpec unique pour une connexion TLS.

Cet attribut possède une valeur de chaîne unique d'un maximum de 32 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqSslPeerName*
Cet attribut LDAP est utilisé pour vérifier le nom distinctif (DN) du certificat du gestionnaire de files d'attente ou du client homologue à l'autre extrémité d'un canal IBM MQ .

Cet attribut LDAP a une valeur de chaîne unique avec un maximum de 1024 octets, qui ne sont pas sensibles à la casse. Ce n'est pas une préexistante.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-amqTransactionProgramName*
Cet attribut LDAP indique le nom du programme de transaction. Il est destiné à être utilisé avec les connexions LU 6.2 .

Cet attribut a une valeur de chaîne unique d'un maximum de 64 caractères, qui ne sont pas sensibles à la casse. Ce n'est pas une préexistante.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-IDamqUser*
Cet attribut LDAP indique l'ID utilisateur à utiliser par l'agent MCA lors de la tentative de lancement d'une session SNA sécurisée avec un agent MCA distant.

Cet attribut a une valeur de chaîne unique de exactement 12 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

ALW *ibm-AffinitéamqConnection*
Cet attribut LDAP indique si les applications client, qui se connectent plusieurs fois à l'aide du même nom de gestionnaire de files d'attente, utilisent le même canal client.

Cet attribut a une valeur entière unique. Il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqClientChannelWeight*
Cet attribut LDAP spécifie une pondération qui influence la définition de canal de connexion client utilisée.

L'attribut de pondération de canal client est utilisé pour biaiser la sélection des définitions de canal client lorsque plusieurs définitions appropriées sont disponibles.

Cet attribut a une valeur entière unique. Il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqHeartBeatInterval*
Cet attribut LDAP indique la durée approximative entre les flux de pulsations qui doivent être transmis à partir d'un agent MCA émetteur lorsqu'il n'y a pas de messages dans la file d'attente de transmission.

Cet attribut a une valeur entière unique. Il ne s'agit pas d'un attribut préexistant. La valeur par défaut est 1. La valeur par défaut est définie dans l'opération de variable d'environnement MQSERVER en cours.

ALW *ibm-amqKeepAliveInterval*
Cet attribut LDAP permet de spécifier une valeur de délai d'attente pour un canal.

La valeur de cet attribut est transmise à la pile de communications en spécifiant le délai de signal de présence pour le canal. Vous pouvez l'utiliser pour spécifier une valeur de signal de présence différente pour chaque canal.

Cet attribut a une valeur entière unique. Il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqMaximumMessageLength*
Cet attribut LDAP indique la longueur maximale d'un message pouvant être transmis sur le canal.

La valeur par défaut de cet attribut est 104857600, conformément à l'opération de variable d'environnement MQSERVER en cours. Cet attribut a une valeur entière unique et il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqSharingConversations*
Cet attribut LDAP indique le nombre maximal de conversations qui partagent chaque instance de canal TCP/IP.

Cet attribut a une valeur entière unique. Cet attribut n'est pas un attribut préexistant.

ALW *ibm-amqTransportType*
Cet attribut LDAP indique le type de transport à utiliser.

Cet attribut a une valeur entière unique. Il ne s'agit pas d'un attribut préexistant.

ALW *ibm-amqIsClientDefault*
Cet attribut booléen résout le problème de recherche d'entrées dans lesquelles l'attribut ibm-amqQueueManagerName n'a pas été défini.

Les modules d'exit de préconnexion recherchent généralement les serveurs LDAP avec la valeur de l'attribut ibm-amqQueueManagerName comme critère de recherche. Une telle requête renvoie toutes les entrées dans lesquelles la valeur de l'attribut ibm-amqQueueManagerName correspond au nom du gestionnaire de files d'attente spécifié dans l'appel MQCONN/X. Toutefois, lorsque vous utilisez les

tables de définition de canal du client (CCDT), vous pouvez soit définir le nom du gestionnaire de files d'attente sur un appel MQCONN/X comme étant vide, soit le préfixer avec un astérisque (*). Si le nom du gestionnaire de files d'attente est vide, le client se connecte au gestionnaire de files d'attente par défaut. Si le nom est précédé d'un astérisque (*) dans le gestionnaire de files d'attente, le client se connecte à n'importe quel gestionnaire de files d'attente.

De même, l'attribut ibm-amqQueueManagerName d'une entrée peut ne pas être défini. Dans ce cas, il est prévu que le client qui utilise ces informations de noeud final puisse se connecter à n'importe quel gestionnaire de files d'attente. Par exemple, une entrée contient les lignes suivantes:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

Dans cet exemple, le client tente de se connecter au gestionnaire de files d'attente spécifié qui s'exécute sous `myhost`.

Toutefois, dans les serveurs LDAP, aucune recherche n'est effectuée sur une valeur d'attribut qui n'a pas été définie. Par exemple, si une entrée contient les informations de connexion, à l'exception de ibm-amqQueueManagerName, les résultats de la recherche n'incluent pas cette entrée. Pour résoudre ce problème, vous pouvez définir ibm-amqIsClientDefault. Il s'agit d'un attribut booléen qui est supposé avoir la valeur FALSE s'il n'est pas défini.

Pour les entrées dans lesquelles ibm-amqQueueManagerName n'a pas été défini et qui doivent faire partie de la recherche, définissez ibm-amqIsClientDefault sur TRUE. Lorsqu'un blanc ou un astérisque (*) est spécifié comme nom de gestionnaire de files d'attente dans un appel à MQCONN/X, l'exit de préconnexion recherche sur le serveur LDAP toutes les entrées pour lesquelles la valeur de l'attribut ibm-amqIsClientDefault est définie sur TRUE.

Remarque : Ne définissez pas l'attribut ibm-amqQueueManagerName si ibm-amqIsClientDefault est défini sur TRUE.

Référence associée

«ibm-amqQueueManagerName», à la page 1190
Cet attribut indique le nom du gestionnaire de files d'attente ou du groupe de gestionnaires de files d'attente auquel une application client IBM MQ peut demander une connexion.

ALW *Compression ibm-amqHeader*
Cet attribut LDAP est une liste de techniques de compression de données d'en-tête prises en charge par le canal.

La taille maximale de cet attribut est de 48 caractères. Il ne s'agit pas d'un attribut préexistant.

Vous ne pouvez spécifier qu'une seule valeur pour cet attribut.

Cet attribut de liste est spécifié en tant que chaînes de répertoire à l'aide d'un format séparé par des virgules. Par exemple, la valeur spécifiée pour **ibm-amqHeaderCompression** est `0` , qui est mappée à `NONE`. Toutes les valeurs qui dépassent la limite maximale autorisée sont ignorées par le client. Par exemple, la compression ibm-amqHeadercontient un maximum de 2 entiers dans la liste.

ALW *ibm-amqMessageCompression*
Cet attribut LDAP est une liste de techniques de compression de données de message prises en charge par le canal.

La taille maximale de cet attribut est de 48 caractères. Il ne s'agit pas d'un attribut préexistant.

Cet attribut ne prend pas en charge les valeurs multiples.

V 9.4.0 Cet attribut de liste est spécifié en tant que chaînes de répertoire à l'aide d'un format séparé par des virgules. Par exemple, la valeur spécifiée pour cet attribut est 1,2,4,16,32 qui correspond à la séquence de compression sous-jacente RLE, ZLIBFAST, ZLIBHIGH, LZ4FASTet LZ4HIGH.

Toutes les valeurs qui dépassent la limite maximale autorisée sont ignorées par le client. Par exemple, la compression ibm-amqMessagecontient un maximum de 16 entiers dans la liste.

ALW *ibm-amqSendExitUserDonnées*
Cet attribut LDAP spécifie les données utilisateur qui sont transmises à l'exit d'émission.

Cet attribut LDAP possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

Remarque : ibm-amqSendExitName et **ibm-amqSendExitUserData** doivent être synchronisés par paires. Les données utilisateur doivent être synchronisées avec le nom de l'exit. Ainsi, si l'un est spécifié, l'autre doit également être spécifié de manière symétrique, même s'il ne contient pas de données.

ALW *ibm-amqSendExitName*
Cet attribut LDAP indique le nom du programme d'exit à exécuter par l'exit d'émission de canal.

Cet attribut possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

Remarque : ibm-amqSendExitName et **ibm-amqSendExitUserData** doivent être synchronisés par paires. Les données utilisateur doivent être synchronisées avec le nom de l'exit. Ainsi, si l'un est spécifié, l'autre doit également être spécifié de manière symétrique même s'il ne contient pas de données.

ALW *ibm-amqReceiveExitUserDonnées*
Cet attribut LDAP spécifie les données utilisateur qui sont transmises à l'exit de réception.

Vous pouvez exécuter une séquence d'exits de réception. La chaîne de données utilisateur d'une série d'exits est séparée par une virgule, des espaces ou les deux.

Cet attribut possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

Remarque : ibm-amqReceiveExitName et **ibm-amqReceiveExitUserData** doivent être synchronisés par paires. Les données utilisateur doivent être synchronisées avec le nom de l'exit. Ainsi, si l'un est spécifié, l'autre doit également être spécifié de manière symétrique même s'il ne contient pas de données.

ALW *ibm-amqReceiveExitName*
Cet attribut LDAP indique le nom du programme d'exit utilisateur à exécuter par l'exit utilisateur de réception de canal.

Cet attribut est une liste de noms de programmes qui doivent être exécutés successivement. Laissez cette zone à blanc si aucun exit utilisateur de réception de canal n'est actif.

Cet attribut possède une valeur de chaîne unique d'un maximum de 999 caractères, qui ne sont pas sensibles à la casse. Il ne s'agit pas d'un attribut préexistant.

La correspondance de sous-chaîne est ignorée. La correspondance de sous-chaîne est une règle de correspondance utilisée dans le sous-schéma qui spécifie le comportement de l'attribut dans un filtre de recherche.

Remarque : **ibm-amqReceiveExitName** et **ibm-amqReceiveExitUserData** doivent être synchronisés par paires. Les données utilisateur doivent être synchronisées avec le nom de l'exit. Ainsi, si l'un est spécifié, l'autre doit également être spécifié de manière symétrique, même s'il ne contient pas de données.

z/OS Using the sample programs for z/OS

The sample procedural applications that are delivered with IBM MQ for z/OS demonstrate typical uses of the Message Queue Interface (MQI).

About this task

IBM MQ for z/OS also provides sample data-conversion exits, described in “Ecriture des exits de conversion de données” on page 1010.

All the sample applications are supplied in source form; several are also supplied in executable form. The source modules include pseudocode that describes the program logic.

Note: Although some of the sample applications have basic panel-driven interfaces, they do not aim to demonstrate how to design the look and feel of your applications. For more information about how to design panel-driven interfaces for non-programmable terminals, see the *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) and its addendum (GG22-9508). These provide guidelines to help you to design applications that are consistent both within the application and across other applications.

Procedure

- Use the following links to find out more about the sample programs:
  - “Features demonstrated in the sample applications for z/OS” on page 1196
  - “Preparing and running sample applications for the batch environment on z/OS” on page 1202
  - “Preparing sample applications for the TSO environment on z/OS” on page 1205
  - “Preparing the sample applications for the CICS environment on z/OS” on page 1207
  - “Preparing the sample application for the IMS environment on z/OS” on page 1210
  - “The Put samples on z/OS” on page 1211
  - “The Get samples on z/OS” on page 1213
  - “The Browse sample on z/OS” on page 1215
  - “The Print Message sample on z/OS” on page 1217
  - “The Queue Attributes sample on z/OS” on page 1221
  - “The Mail Manager sample on z/OS” on page 1222
  - “The Credit Check sample on z/OS” on page 1229
  - “The Message Handler sample on z/OS” on page 1240
  - “The Asynchronous Put sample on z/OS” on page 1243
  - “The Batch Asynchronous Consumption sample on z/OS” on page 1244
  - “The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” on page 1246
  - “The Publish/Subscribe sample on z/OS” on page 1248
  - “The Set and Inquire message property sample on z/OS” on page 1251

Related tasks

“Utilisation des exemples de programme sur Multiplatforms” on page 1086
Ces exemples de programmes procéduraux sont fournis avec le produit. Les exemples sont écrits en langage C et COBOL et illustrent les utilisations typiques de l'interface MQI (Message Queue Interface).

z/OS *Features demonstrated in the sample applications for z/OS*

This section summarizes the MQI features demonstrated in each of the sample applications, shows the programming languages in which each sample is written, and the environment in which each sample runs.

z/OS *Put samples on z/OS*

The Put samples demonstrate how to put messages on a queue using the MQPUT call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1203 for the batch application and Table 179 on page 1207 for the CICS application.

z/OS *Get samples on z/OS*

The Get samples demonstrate how to get messages from a queue using the MQGET call.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

The program is delivered in COBOL and C, and runs in the batch and CICS environment. See Table 172 on page 1203 for the batch application and Table 179 on page 1207 for the CICS application.

z/OS *Browse sample on z/OS*

The Browse sample demonstrates how to use the Browse option to find a message, print it, then step through the messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for browsing messages
- MQCLOSE
- MQDISC

The program is delivered in the COBOL, assembler, PL/I, and C languages. The application runs in the batch environment. See Table 173 on page 1203 for the batch application.

z/OS *Print Message sample on z/OS*

The Print Message sample demonstrates how to remove a message from a queue and print the data in the message, together with all the fields of its message descriptor. It can, optionally, display all of the message properties associated with each message.

By removing comment characters from two lines in the source module, you can change the program so that it browses, rather than removes, the messages on a queue. This program can usefully be used for diagnosing problems with an application that is putting messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQGET for removing messages from a queue (with an option to browse)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

The program is delivered in the C language. The application runs in the batch environment. See Table 174 on page 1203 for the batch application.

z/OS *Queue Attributes sample on z/OS*

The Queue Attributes sample demonstrates how to inquire about and set the values of IBM MQ for z/OS object attributes.

The application uses these MQI calls:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

The program is delivered in the COBOL, assembler, and C languages. The application runs in the CICS environment. See Table 180 on page 1208 for the CICS application.

z/OS *Mail Manager sample on z/OS*

Considerations to note when using Mail Manager sample.

The Mail Manager sample demonstrates these techniques:

- Using alias queues
- Using a model queue to create a temporary dynamic queue
- Using reply-to queues
- Using syncpoints in the CICS and batch environments
- Sending commands to the system-command input queue
- Testing return codes
- Sending messages to remote queue managers, both by using a local definition of a remote queue and by putting messages directly on a named queue at a remote queue manager

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQPUT1
- MQGET
- MQINQ
- MQCMIT
- MQCLOSE
- MQDISC

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

The TSO applications use the IBM MQ for z/OS batch adapter and include some ISPF panels.

See Table 177 on page 1205 for the TSO application, and Table 181 on page 1208 for the CICS application.

z/OS *Credit Check sample on z/OS*

This information contains points to consider when using Credit Check sample.

The Credit Check sample is a suite of programs that demonstrates these techniques:

- Developing an application that runs in more than one environment
- Using a model queue to create a temporary dynamic queue
- Using a correlation identifier
- Setting and passing context information
- Using message priority and persistence
- Starting programs by using triggering
- Using reply-to queues
- Using alias queues
- Using a dead-letter queue
- Using a namelist
- Testing return codes

The application uses these MQI calls:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET for browsing and getting messages, using the wait and signal options, and for getting a specific message
- MQINQ
- MQSET
- MQCLOSE

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered in C.

See Table 182 on page 1209 for the CICS application, and Table 184 on page 1210 for the IMS application.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample allows you to browse, forward, and delete messages on a queue.

The application uses these MQI calls:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1

- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

The program is delivered in C and COBOL programming languages. The application runs under TSO. SeeTable 178 on page 1206for the TSO application.

z/OS *Distributed queuing exit samples on z/OS*

A table of source programs of Distributed queuing exit samples.

The names of the source programs of the distributed queuing exit samples are listed in the following table:

Table 170. Source for the distributed queuing exit samples

Member name	For language	Description	Supplied in library
CSQ4BAX0	Assembler	Source program	SCSQASMS
CSQ4BCX1	C	Source program	SCSQC37S
CSQ4BCX2	C	Source program	SCSQC37S
CSQ4BCX4	C	Source program	SCSQC37S

Note: The source programs are link-edited with CSQXSTUB.

z/OS *Data-conversion exit samples on z/OS*

A skeleton is provided for a data-conversion exit routine, and a sample is shipped with IBM MQ illustrating the MQXCNVC call.

The names of the source programs of the data-conversion exit samples are listed in the following table:

Table 171. Source for the data conversion exit samples (assembler language only)

Member name	Description	Supplied in library
CSQ4BAX8	Source program	SCSQASMS
CSQ4BAX9	Source program	SCSQASMS
CSQ4CAX9	Source program	SCSQASMS

Note: The source programs are link-edited with CSQASTUB.

See “Ecriture des exits de conversion de données” on page 1010 for more information.

z/OS *Publish/Subscribe samples on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB

- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

The Public/Subscribe sample programs are delivered in the C and COBOL programming languages. The sample applications run in the batch environment. See Publish/Subscribe samples for the batch applications.

z/OS *Configuring a queue manager to accept client connections on z/OS*

Before you can run the sample applications, you must first create a queue manager. You can then configure the queue manager to securely accept incoming connection requests from applications that are running in client mode.

Before you begin

Ensure the queue manager already exists and has been started. Determine whether channel authentication records are already enabled by issuing the MQSC command:

```
DISPLAY QMGR CHLAUTH
```

Important: This task expects that channel authentication records are enabled. If this is a queue manager used by other users and applications, changing this setting will affect all other users and applications. If your queue manager does not make use of channel authentication records then step 4 can be replaced with an alternate authentication method (for example a security exit) which sets the MCAUSER to the *non-privileged-user-id* you will obtain in step "1" on page 1200.

You must know which channel name your application expects to use so that the application can be permitted to use the channel. You must also know which objects, for example queues or topics, your application expects to use so that your application can be permitted to use them.

About this task

This task creates a non-privileged user ID to be used for a client application which connects to the queue manager. Access is granted for the client application only to be able to use the channel it needs and the queue it needs by use of this user ID.

Procedure

1. Obtain a user ID on the system your queue manager is running on.

   For this task this user ID must not be a privileged administrative user. This user ID is the authority under which the client connection will run on the queue manager.
2. Start a listener program.
   a) Ensure that your channel initiator is started. If not, start it by issuing the **START CHINIT** command.
   b) Start the listener program by issuing the following command:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   where *nnnn* is your chosen port number.
3. If your application uses the SYSTEM.DEF.SVRCONN then this channel is already defined. If your application uses another channel, create it by issuing the MQSC command:

```
DEFINE CHANNEL(' channel-name ') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Channel for use by sample programs')
```

channel-name is the name of your channel.

4. Create a channel authentication rule allowing only the IP address of your client system to use the channel by issuing the MQSC command:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

where

channel-name is the name of your channel.

client-machine-IP-address is the IP address of your client system. If your sample client application is running on the same machine as the queue manager then use an IP address of '127.0.0.1' if your application is going to connect using 'localhost'. If several different client machines are going to connect in, you can use a pattern or a range instead of a single IP address. See Generic IP addresses for details.

non-privileged-user-id is the user ID you obtained in step “1” on page 1200

5. If your application uses the SYSTEM.DEFAULT.LOCAL.QUEUE, then this queue is already defined. If your application uses another queue, create it by issuing the MQSC command:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

where *queue-name* is the name of your queue.

6. Grant access to connect to and inquire the queue manager:
   a) Ensure that your channel initiator is started. If not, start the channel initiator by issuing the START CHINIT command.
   b) Start a TCP listener, for example issue the following command:

```
START LISTENER TRPTYPE(TCP) PORT(nnnn)
```

   where *nnnn* is your chosen port number.

7. If your application is a point-to-point application, that is it makes use of queues, grant access to allow inquiring and the putting and getting messages using your queue by the user ID to be used, by issuing the MQSC commands:

   Issue the RACF commands:

```
RDEFINE MQQUEUE qmgr-name.QUEUE. queue-name UACC(NONE)

PERMIT qmgr-name.QUEUE. queue-name CLASS(MQQUEUE) ID(non-privileged-user-id) ACCESS(UPDATE)
```

where

qmgr-name is the name of your queue manager

queue-name is the name of your queue.

non-privileged-user-id is the user ID you obtained in step “1” on page 1200

8. If your application is a publish/subscribe application, that is it makes use of topics, grant access to allow publishing and subscribing using your topic by the user ID to be used, by issuing the following RACF commands:

```
RDEFINE MQTOPIC qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC UACC(NONE)

PERMIT qmgr-name.PUBLISH.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)

RDEFINE MQTOPIC qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC UACC(NONE)
```

```
PERMIT qmgr-name.SUBSCRIBE.SYSTEM.BASE.TOPIC CLASS(MQTOPIC) ID(non-privileged-user-id)
ACCESS(UPDATE)
```

where

qmgr-name is the name of your queue manager

non-privileged-user-id is the user ID you obtained in step “1” on page 1200

This will give *non-privileged-user-id* access to any topic in the topic tree, alternatively, you can define a topic object using **DEFINE TOPIC** and grant accesses only to the part of the topic tree referenced by that topic object. For more information, see Controlling user access to topics.

What to do next

Your client application can now connect to the queue manager and put or get messages using the queue.

Related concepts

z/OS Authority to work with IBM MQ objects on z/OS

Related reference

SET CHLAUTH

DEFINE CHANNEL

DEFINE QLOCAL

SET AUTHREC

z/OS *Preparing and running sample applications for the batch environment on z/OS*

To prepare a sample application that runs in the batch environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in “Building z/OS batch applications” on page 1051.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

Note: The assembler language version of the Browse sample uses data control blocks (DCBs), so you must link-edit it using RMODE(24).

The library members to use are listed in Table 172 on page 1203, Table 173 on page 1203, Table 174 on page 1203, and Table 175 on page 1204.

You must edit the run JCL supplied for the samples that you want to use (see Table 172 on page 1203, Table 173 on page 1203, Table 174 on page 1203, and Table 175 on page 1204).

The PARM statement in the supplied JCL contains a number of parameters that you need to modify. To run the C sample programs, separate the parameters by spaces; to run the assembler, COBOL, and PL/I sample programs, separate them by commas. For example, if the name of your queue manager is CSQ1 and you want to run the application with a queue named LOCALQ1, in the COBOL, PL/I, and assembler-language JCL, your PARM statement should look like this:

```
PARM=(CSQ1,LOCALQ1)
```

In the C language JCL, your PARM statement should look like this:

```
PARM=('CSQ1 LOCALQ1')
```

You are now ready to submit the jobs.

z/OS *Names of the sample batch applications on z/OS*

A summary of the programs that are supplied for sample batch applications.

The batch application programs are summarized in the following tables:

- Table 172 on page 1203 Put and Get samples
- Table 173 on page 1203 Browse sample
- Table 174 on page 1203 Print message sample
- Table 175 on page 1204 Publish/Subscribe samples
- Table 176 on page 1204 Other samples

Table 172. Batch Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCJ1	C	Get source program	SCSQC37S	SCSQLOAD
CSQ4BCK1	C	Put source program	SCSQC37S	SCSQLOAD
CSQ4BCJR	C	Sample run JCL for CSQ4BCJ1 and CSQBCK1	SCSQPROC	None
CSQ4BVJ1	COBOL	Get source program	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Put source program	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	Sample run JCL for CSQBVJ1 and CSQBVK1	SCSQPROC	None

Table 173. Batch Browse sample

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4BVA1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	Sample run JCL for CSQ4BVA1	SCSQPROC	None
CSQ4BAA1	Assembler	Source program	SCSQASMS	SCSQLOAD
CSQ4BAAR	Assembler	Sample run JCL for CSQ4BAA1	SCSQPROC	None
CSQ4BCA1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4BCAR	C	Sample run JCL for CSQ4BCA1	SCSQPROC	None
CSQ4BPA1	PL/I	Source program	SCSQPLIS	SCSQLOAD
CSQ4BPAR	PL/I	Sample run JCL for CSQ4BPA1	SCSQPROC	None

Table 174. Batch Print Message sample (C language only)

Member name	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCG1	Source program	SCSQC37S	SCSQLOAD

Table 174. Batch Print Message sample (C language only) (continued)

Member name	Description	Source file supplied in library	Executable file supplied in library
CSQ4BCGR	Sample run JCL for CSQ4BCG1	SCSQPROC	None
CSQ4BCL1	Browse source program	SCSQC37S	SCSQLOAD
CSQ4BCLR	Sample run JCL for CSQ4BCL1	SCSQPROC	None

Table 175. Publish/Subscribe samples

Member name	For language	Description	Source file supplied in library	JCL in SCSQPROC	Executable file supplied in library
CSQ4BCP1	C	Publish to topic source program	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	Subscribe to topic and get messages source program	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	Subscribe to topic using a user provided destination and get messages source program	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Subscribe to topic using extended options and get messages source program	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	Publish to topic source program	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	Subscribe to topic and get messages source program	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Table 176. Other samples

Member name	For language	Description	Source file supplied in library	JCL in SCSQPROC	Executable file supplied in library
CSQ4BCS1	C	Asynchronous consumption source program	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	Asynchronous Put, and Check status source program	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Inquire message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD

Table 176. Other samples (continued)					
Member name	**For language**	**Description**	**Source file supplied in library**	**JCL in SCSQPROC**	**Executable file supplied in library**
CSQ4BCM2	C	Set message properties source program	SCSQC37S	CSQ4BCMP	SCSQLOAD

z/OS *Preparing sample applications for the TSO environment on z/OS*

To prepare a sample application that runs in the TSO environment, perform the same steps that you would when building any batch IBM MQ for z/OS application.

These steps are listed in "Building z/OS batch applications" on page 1051. The library members to use are listed in Table 177 on page 1205.

Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQLOAD load library.

For the Mail Manager sample application, ensure that the queues that it uses are available on your system. They are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

z/OS *Names of the sample TSO applications on z/OS*

Information about the names of the programs that are supplied for each of the sample TSO applications, and the libraries where the source, JCL, and, for the Message Handler sample only, the executable files reside.

The TSO application programs are summarized in the following tables:

- Table 177 on page 1205 Mail manager sample
- Table 178 on page 1206 Message handler sample

These samples use ISPF panels. You must therefore include the ISPF stub, ISPLINK, when you link-edit the programs.

Table 177. TSO Mail Manager sample			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4CVD	independent	IBM MQ for z/OS object definitions	SCSQPROC
CSQ40	independent	ISPF messages	SCSQMSGE
CSQ4RVD1	COBOL	CLIST to initiate CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Source program for Menu program	SCSQCOBS
CSQ4TVD2	COBOL	Source program for Get Mail program	SCSQCOBS
CSQ4TVD4	COBOL	Source program for Send Mail program	SCSQCOBS
CSQ4TVD5	COBOL	Source program for Nickname program	SCSQCOBS

Table 177. TSO Mail Manager sample (continued)			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4VDP1-6	COBOL	Panel definitions	SCSQPNLA
CSQ4VD0	COBOL	Data definition	SCSQCOBC
CSQ4VD1	COBOL	Data definition	SCSQCOBC
CSQ4VD2	COBOL	Data definition	SCSQCOBC
CSQ4VD4	COBOL	Data definition	SCSQCOBC
CSQ4RCD1	C	CLIST to initiate CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Source program for Menu program	SCSQC37S
CSQ4TCD2	C	Source program for Get Mail program	SCSQC37S
CSQ4TCD4	C	Source program for Send Mail program	SCSQC37S
CSQ4TCD5	C	Source program for Nickname program	SCSQC37S
CSQ4CDP1-6	C	Panel definitions	SCSQPNLA
CSQ4TC0	C	Include file	SCSQC370

Table 178. TSO Message Handler sample				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4TCH0	C	Data definition	SCSQC370	None
CSQ4TCH1	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Source program	SCSQC37S	SCSQLOAD
CSQ4TCH3	C	Source program	SCSQC37S	SCSQLOAD
CSQ4RCH1	C and COBOL	CLIST to initiate CSQ4TCH1 or CSQ4TVH1	SCSQCLST	None
CSQ4CHP1	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP2	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP3	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4CHP9	C and COBOL	Panel definition	SCSQPNLA	None
CSQ4TVH0	COBOL	Data definition	SCSQCOBC	None
CSQ4TVH1	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Source program	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Source program	SCSQCOBS	SCSQLOAD

z/OS *Preparing the sample applications for the CICS environment on z/OS*

Before you run the CICS sample programs, log on to CICS using a LOGMODE of 32702. This is because the sample programs have been written to use a 3270 mode 2 screen.

To prepare a sample application that runs in the CICS environment, perform the following steps:

1. Create the symbolic description map and the physical screen map for the sample by assembling the BMS screen definition source (supplied in library **thlqual**.SCSQMAPS, where **thlqual** is the high-level qualifier used by your installation). When you name the maps, use the name of the BMS screen definition source (not available for Put and Get sample programs), but omit the last character of that name.
2. Perform the same steps that you would when building any CICS IBM MQ for z/OS application. These steps are listed in "Building CICS applications in z/OS" on page 1054. The library members to use are listed in Table 179 on page 1207, Table 180 on page 1208, Table 181 on page 1208, and Table 182 on page 1209.

   Alternatively, where we supply an executable form of a sample, you can run it from the thlqual.SCSQCICS load library.
3. Identify the map set, programs, and transaction to CICS by updating the CICS system definition (CSD) data set. The definitions that you require are in the member **thlqual**.SCSQPROC(CSQ4S100). For guidance on how to do this, see *The CICS-IBM MQ Adapter* section in the CICS Transaction Server for z/OS 4.1 product documentation at: CICS Transaction Server for z/OS 4.1, The CICS-IBM MQ adapter.

   Note: For the Credit Check sample application, you get an error message at this stage if you have not already created the VSAM data set that the sample uses.
4. For the Credit Check and Mail Manager sample applications, ensure that the queues that they use are available on your system. For the Credit Check sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVB) for COBOL, and **thlqual**.SCSQPROC(CSQ4CCB) for C. For the Mail Manager sample, they are defined in the member **thlqual**.SCSQPROC(CSQ4CVD). To ensure that these queues are always available, you could add these members to your CSQINP2 initialization input data set, or use the CSQUTIL program to load these queue definitions.

   For the Queue Attributes sample application, you could use one or more of the queues that are supplied for the other sample applications. Alternatively, you could use your own queues. However, in the form that it is supplied, this sample works only with queues that have the characters CSQ4SAMP in the first eight bytes of their name.

z/OS *Names of the sample CICS applications on z/OS*

This topic provides a summary of the programs supplied for sample CICS applications.

The CICS application programs are summarized in the following tables:

- Table 179 on page 1207 Put and Get samples
- Table 180 on page 1208 Queue Attributes sample
- Table 181 on page 1208 Mail Manager sample (COBOL only)
- Table 182 on page 1209 Credit Check sample
- Table 183 on page 1210 Asynchronous Consumption and Publish/Subscribe samples

Table 179. CICS Put and Get samples

Member name	For language	Description	Source file supplied in library	Executable file supplied in library
CSQ4CCK1	C	Put source program	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Get source program	SCSQC37S	SCSQCICS

Table 179. CICS Put and Get samples (continued)				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4CVJ1	COBOL	Get source program	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Put source program	SCSQCOBS	SCSQCICS
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 180. CICS Queue Attributes sample				
Member name	**For language**	**Description**	**Source file supplied in library**	**Executable file supplied in library**
CSQ4CVC1	COBOL	Source program	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Message definition	SCSQCOBC	None
CSQ4VCMS	COBOL	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CAC1	Assembler	Source program	SCSQASMS	SCSQCICS
CSQ4AMSG	Assembler	Message definition	SCSQMACS	None
CSQ4ACMS	Assembler	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4CCC1	C	Source program	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Message definition	SCSQC370	None
CSQ4CCMS	C	BMS screen definition	SCSQMAPS	SCSQCICS (named CSQ4ACM)
CSQ4S100	independent	CICS system definition data set	SCSQPROC	None

Table 181. CICS Mail Manager sample (COBOL only)		
Member name	**Description**	**Source file supplied in library**
CSQ4CVD	IBM MQ for z/OS object definitions	SCSQPROC
CSQ4CVD1	Source for Menu program	SCSQCOBS
CSQ4CVD2	Source for Get Mail program	SCSQCOBS
CSQ4CVD3	Source for Display Message program	SCSQCOBS
CSQ4CVD4	Source for Send Mail program	SCSQCOBS
CSQ4CVD5	Source for Nickname program	SCSQCOBS
CSQ4VDMS	BMS screen definition source	SCSQMAPS
CSQ4S100	CICS system definition data set	SCSQPROC
CSQ4VD0	Data definition	SCSQCOBC

Table 181. CICS Mail Manager sample (COBOL only) (continued)		
Member name	**Description**	**Source file supplied in library**
CSQ4VD3	Data definition	SCSQCOBC
CSQ4VD4	Data definition	SCSQCOBC

Table 182. CICS Credit Check sample			
Member name	**For language**	**Description**	**Source file supplied in library**
CSQ4CVB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CCB	independent	IBM MQ object definitions	SCSQPROC
CSQ4CVB1	COBOL	Source for user-interface program	SCSQCOBS
CSQ4CVB2	COBOL	Source for credit application manager	SCSQCOBS
CSQ4CVB3	COBOL	Source for checking-account program	SCSQCOBS
CSQ4CVB4	COBOL	Source for distribution program	SCSQCOBS
CSQ4CVB5	COBOL	Source for agency-query program	SCSQCOBS
CSQ4CCB1	C	Source for user-interface program	SCSQC37S
CSQ4CCB2	C	Source for credit application manager	SCSQC37S
CSQ4CCB3	C	Source for checking-account program	SCSQC37S
CSQ4CCB4	C	Source for distribution program	SCSQC37S
CSQ4CCB5	C	Source for agency-query program	SCSQC37S
CSQ4CB0	C	Include file	SCSQC370
CSQ4CBMS	C	BMS screen definition source	SCSQMAPS
CSQ4VBMS	COBOL	BMS screen definition source	SCSQMAPS
CSQ4VB0	COBOL	Data definition	SCSQCOBC
CSQ4VB1	COBOL	Data definition	SCSQCOBC
CSQ4VB2	COBOL	Data definition	SCSQCOBC
CSQ4VB3	COBOL	Data definition	SCSQCOBC
CSQ4VB4	COBOL	Data definition	SCSQCOBC
CSQ4VB5	COBOL	Data definition	SCSQCOBC
CSQ4VB6	COBOL	Data definition	SCSQCOBC
CSQ4VB7	COBOL	Data definition	SCSQCOBC
CSQ4VB8	COBOL	Data definition	SCSQCOBC
CSQ4BAQ	independent	Source for VSAM data set	SCSQPROC
CSQ4FILE	independent	JCL to build VSAM data set used by CSQ4CVB3	SCSQPROC
CSQ4S100	independent	CICS system definition data set	SCSQPROC

Table 183. CICS Asynchronous Consumption and Publish/Subscribe samples		
Member name	**Description**	**Source file supplied in library**
CSQ4CVCN	Source for Simple Message Consumption program	SCSQCOBS
CSQ4CVCT	Source for Control Message Consumption program	SCSQCOBS
CSQ4CVEV	Source for Event Handler program	SCSQCOBS
CSQ4CVPT	Source for Message Put Client program	SCSQCOBS
CSQ4CVRG	Source for Registration Client program	SCSQCOBS
CSQ4S100	CICS System Definition data set	SCSQPROC

z/OS *Preparing the sample application for the IMS environment on z/OS*

Part of the Credit Check sample application can run in the IMS environment.

To prepare this part of the application to run with the CICS sample, first perform the steps described in “Preparing the sample applications for the CICS environment on z/OS” on page 1207.

Then perform the following steps:

1. Perform the same steps that you would when building any IMS IBM MQ for z/OS application. These steps are listed in “Building IMS (BMP or MPP) applications” on page 1055. The library members to use are listed in Table 184 on page 1210.
2. Identify the application program and database to IMS. Samples are provided with PSBGEN, DBDGEN, ACB definition, IMSGEN, and IMSDALOC statements to enable this.
3. Load the database CSQ4CA by tailoring and running the sample JCL provided for this purpose (CSQ4ILDB). This JCL loads the database with data from the file CSQ4BAQ. Update the IMS control region with a DD statement for the database CSQ4CA.
4. Start the checking-account program as a batch message processing (BMP) program by tailoring and running the sample JCL provided for this purpose. This JCL starts a batch-oriented BMP program. To run the program as a message-oriented BMP program, remove the comment characters from the line in the JCL that contains the IN= statement.

z/OS *Names of the sample IMS application on z/OS*

This information provides a table with the list of the sources and JCLs that are supplied for the Credit Check sample IMS application.

Table 184. Source and JCL for the Credit Check IMS sample (C only)		
Member name	**Description**	**Supplied in library**
CSQ4CVB	IBM MQ object definitions	SCSQPROC
CSQ4ICB3	Source for checking-account program	SCSQC37S
CSQ4ICBL	Source for loading the checking-account database	SCSQC37S
CSQ4CBI	Data definition	SCSQC370
CSQ4PSBL	PSBGEN JCL for database-load program	SCSQPROC

Table 184. Source and JCL for the Credit Check IMS sample (C only) (continued)		
Member name	**Description**	**Supplied in library**
CSQ4PSB3	PSBGEN JCL for checking-account program	SCSQPROC
CSQ4DBDS	DBDGEN JCL for database CSQ4CA	SCSQPROC
CSQ4GIMS	IMSGEN macro definitions for CSQ4IVB3 and CSQ4CA	SCSQPROC
CSQ4ACBG	Application control block (ACB) definition for CSQ4IVB3	SCSQPROC
CSQ4BAQ	Source for database	SCSQPROC
CSQ4ILDB	Sample run JCL for database-load job	SCSQPROC
CSQ4ICBR	Sample run JCL for checking-account program	SCSQPROC
CSQ4DYNA	IMSDALOC macro definitions for database	SCSQPROC

z/OS *The Put samples on z/OS*

The Put sample programs put messages on a queue using the MQPUT call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1203 and Table 179 on page 1207).

Design of the Put sample

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.
2. Open the queue using the MQOPEN call with the MQOO_OUTPUT option. On input to this call, the program uses the connection handle that is returned in step “1” on page 1213. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQPUT calls until the required number of messages are put on the queue. If an MQPUT call fails, the loop is abandoned early, no further MQPUT calls are attempted, and the completion and reason codes are returned.
4. Close the queue using the MQCLOSE call with the object handle returned in step “2” on page 1213. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “1” on page 1213. If this call fails, print the completion and reason codes.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

z/OS *The Put samples for the batch environment on z/OS*

Use this topic when considering Put samples for the batch environment.

To run the samples, edit and run the sample JCL, as described in "Preparing and running sample applications for the batch environment on z/OS" on page 1202.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages (up to 4 digits)
4. The padding character to write in the message (1 character)
5. The number of characters to write in the message (up to 4 digits)
6. The persistence of the message (1 character: P for persistent or N for nonpersistent)

If you enter any of the these parameters wrongly, you receive appropriate error messages.

Any messages from the samples are written to the SYSPRINT data set.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR. None of the differences relate to the MQI.
- CSQ4BCK1 allows you to enter more than four digits for the number of messages sent and the length of the messages.
- For the two numeric fields, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, but the C program receives an error.
- For both programs, CSQ4BCK1 and CSQ4BVK1, you must enter P in the persistence parameter, ++PER++, if you want the message to be persistent. If you fail to do so, the message will be nonpersistent.

z/OS *The Put samples for the CICS environment on z/OS*

Use this topic when considering Put samples for the CICS environment.

The transactions take the following parameters separated by commas:

1. The number of messages (up to 4 digits)
2. The padding character to write in the message (1 character)
3. The number of characters to write in the message (up to 4 digits)
4. The persistence of the message (1 character: P for persistent or N for nonpersistent)
5. The name of the target queue (48 characters)

If you enter any of these parameters wrongly, you receive appropriate error messages.

For the COBOL sample, invoke the Put sample in the CICS environment by entering:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

For the C sample, invoke the Put sample in the CICS environment by entering:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Any messages from the samples are displayed on the screen.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- For the two numeric fields, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to put a single message, you can enter the value 1, 01, 001, or 0001. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program sends a 1-byte message, and the C program abends with an error from malloc().
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter P in the persistence parameter if you want the message to be persistent. For non-persistent messages, enter N in the persistence parameter. If you enter any other value you receive an error message.
- The messages are put in syncpoint because default values are used for all parameters except those set during program invocation.

z/OS *The Get samples on z/OS*

The Get sample programs get messages from a queue using the MQGET call.

The source programs are supplied in C and COBOL in the batch and CICS environments (see Table 172 on page 1203 and Table 179 on page 1207).

z/OS *Design of the Get sample on z/OS*

Learn about the design of the Get sample, and some usage notes to consider.

The flow through the program logic is:

1. Connect to the queue manager using the MQCONN call. If this call fails, print the completion and reason codes and stop processing.

   Note: If you are running the sample in a CICS environment, you do not need to issue an MQCONN call; if you do, it returns DEF_HCONN. You can use the connection handle MQHC_DEF_HCONN for the MQI calls that follow.
2. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED and MQOO_BROWSE options. On input to this call, the program uses the connection handle that is returned in step "1" on page 1213. For the object descriptor structure (MQOD), it uses the default values for all fields except the queue name field, which is passed as a parameter to the program. If the MQOPEN call fails, print the completion and reason codes and stop processing.
3. Create a loop within the program issuing MQGET calls until the required number of messages are retrieved from the queue. If an MQGET call fails, the loop is abandoned early, no further MQGET calls are attempted, and the completion and reason codes are returned. The following options are specified on the MQGET call:

   - MQGMO_NO_WAIT
   - MQGMO_ACCEPT_TRUNCATED_MESSAGE
   - MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT
   - MQGMO_BROWSE_FIRST and MQGMO_BROWSE_NEXT

   For a description of these options, see MQGET. For each message, the message number is printed followed by the length of the message and the message data.
4. Close the queue using the MQCLOSE call with the object handle returned in step "2" on page 1213. If this call fails, print the completion and reason codes.
5. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step "1" on page 1213. If this call fails, print the completion and reason codes.

Note: If you are running the sample in a CICS environment, you do not need to issue an MQDISC call.

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. However, these differences are minimized if you use the layout of the parameters shown in the sample run JCL, CSQ4BCJR, and CSQ4BVJR,. None of the differences relate to the MQI.
- CSQ4BCJ1 allows you to enter more than four digits for the number of messages retrieved.
- Messages longer than 64 KB are truncated.
- CSQ4BCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.
- For the numeric number-of-messages field, enter any digit in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter 1, 01, 001, or 0001 as the value. If you enter nonnumeric or negative values, you might receive an error. For example, if you enter -1, the COBOL program retrieves one message, but the C program does not retrieve any messages.
- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter B in the get parameter, ++GET++, if you want to browse the messages.
- For both programs, CSQ4BCJ1 and CSQ4BVJ1, enter S in the syncpoint parameter, ++SYNC++, for messages to be retrieved in syncpoint.

z/OS *The Get samples for the batch environment on z/OS*

To run the samples, edit and run the sample JCL, as described in "Preparing and running sample applications for the batch environment on z/OS" on page 1202.

The programs take the following parameters in an EXEC PARM, separated by spaces in C and commas in COBOL:

1. The name of the queue manager (4 characters)
2. The name of the target queue (48 characters)
3. The number of messages to get (up to 4 digits)
4. The browse/get message option (1 character: B to browse or D to destructively get the messages)
5. The syncpoint control (1 character: S for syncpoint or N for no syncpoint)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

Output from the samples is written to the SYSPRINT data set:

```
====================================
PARAMETERS PASSED :
QMGR    - VC9
QNAME    - A.Q
NUMMSGS   - 000000002
GET     - D
SYNCPOINT  - N
====================================
MQCONN SUCCESSFUL
MQOPEN SUCCESSFUL
000000000 : 000000010 : **********
000000001 : 000000010 : **********
000000002 MESSAGES GOT FROM QUEUE
MQCLOSE SUCCESSFUL
MQDISC SUCCESSFUL
```

z/OS *The Get samples for the CICS environment on z/OS*

Special considerations for the Get samples for the CICS environment.

The transactions take the following parameters in an EXEC PARM, separated by commas:

1. The number of messages to get (up to four digits)
2. The browse/get message option (one character: B to browse or D to destructively get the messages)
3. The syncpoint control (one character: S for syncpoint or N for no syncpoint)
4. The name of the target queue (48 characters)

If you enter any of these parameters incorrectly, you receive appropriate error messages.

For the COBOL sample, invoke the Get sample in the CICS environment by entering:

```
MVGT,9999,B,S,QUEUE.NAME
```

For the C sample, invoke the Get sample in the CICS environment by entering:

```
MCGT,9999,B,S,QUEUE.NAME
```

When the messages are retrieved from the queue, they are put on a CICS temporary storage queue with the same name as the CICS transaction (for example, MCGT for the C sample).

Here is example output of the Get samples:

```
*************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
*************************** BOTTOM OF QUEUE **********************
```

Usage notes

- To keep the samples simple, there are some minor functional differences between language versions. None of the differences relate to the MQI.
- If you enter a queue name that is longer than 48 characters, its length is truncated to the maximum of 48 characters but no error message is returned.
- Before entering the transaction, press the CLEAR key.
- CSQ4CCJ1 can only correctly display character messages because it only displays until the first NULL (\0) character is displayed.
- For the numeric field, enter any number in the range 1 through 9999. The value that you enter should be a positive number. For example, to get a single message, you can enter the value 1, 01, 001, or 0001. If you enter a nonnumeric or negative value, you might receive an error.
- Messages longer than 24 526 bytes in C and 9 950 bytes in COBOL are truncated. This is due to the way that the CICS temporary storage queues are used.
- For both programs, CSQ4CCK1 and CSQ4CVK1, enter B in the get parameter if you want to browse the messages, otherwise enter D. This performs destructive MQGET calls. If you enter any other value you receive an error message.
- For both programs, CSQ4CCJ1 and CSQ4CVJ1, enter S in the syncpoint parameter to retrieve messages in syncpoint. If you enter N in the syncpoint parameter, the MQGET calls are issued out of syncpoint. If you enter any other value you receive an error message.

z/OS *The Browse sample on z/OS*

The Browse sample is a batch application that demonstrates how to browse messages on a queue using the MQGET call.

The application steps through all the messages in a queue, printing the first 80 bytes of each one. You could use this application to look at the messages on a queue without changing them.

Source programs and sample run JCL are supplied in the COBOL, assembler, PL/I, and C languages (see Table 173 on page 1203).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1202. You can look at messages on one of your own queues by specifying the name of the queue in the run JCL.

When you run the application (and there are some messages on the queue), the output data set looks this:

```
07/12/1998            SAMPLE QUEUE REPORT     PAGE  1
QUEUE MANAGER NAME : VC4
QUEUE NAME : CSQ4SAMP.DEAD.QUEUE
RELATIVE
MESSAGE  MESSAGE
NUMBER  LENGTH ------------------- MESSAGE DATA -------------

1    740 HELLO. PLEASE CALL ME WHEN YOU GET BACK.
2    429 CSQ4BQRM
3    429 CSQ4BQRM
4    429 CSQ4BQRM
5    22 THIS IS A TEST MESSAGE
6     8 CSQ4TEST
7    36 CSQ4MSG - ANOTHER TEST MESSAGE.....
!8     9 CSQ4STOP
********** END OF REPORT **********
```

If there are no messages on the queue, the data set contains the headings and the `End of report` message only. If an error occurs with any of the MQI calls, the completion and reason codes are added to the output data set.

z/OS *Design of the Browse sample on z/OS*

The Browse sample application uses a single program module; one is provided in each of the supported programming languages.

The flow through the program logic is:

1. Open a print data set and print the title line of the report. Check that the names of the queue manager and queue have been passed from the run JCL. If both names have been passed, print the lines of the report that contain the names. If they have not, print an error message, close the print data set, and stop processing.

   The way that the program tests the parameters it is passed from the JCL depends on the language in which the program is written; for more information, see “Language-dependent design considerations on z/OS” on page 1217.
2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
3. Open the queue using the MQOPEN call with the MQOO_BROWSE option. On input to this call, the program uses the connection handle returned in step “2” on page 1216. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step “1” on page 1216). If this call is not successful, print the completion and reason codes, close the print data set, and stop processing.
4. Browse the first message on the queue, using the MQGET call. On input to this call, the program specifies:
   - The connection and queue handles from steps “2” on page 1216 and “3” on page 1216
   - An MQMD structure with all fields set to their initial values
   - Two options:
     - MQGMO_BROWSE_FIRST
     - MQGMO_ACCEPT_TRUNCATED_MSG
   - A buffer of size 80 bytes to hold the data copied from the message

   The MQGMO_ACCEPT_TRUNCATED_MSG option allows the call to complete even if the message is longer than the 80-byte buffer specified in the call. If the message is longer than the buffer, the message is truncated to fit the buffer, and the completion and reason codes are set to show this. The

sample was designed so that messages are truncated to 80 characters to make the report easy to read. The buffer size is set by a `DEFINE` statement, so you can easily change it if you want to.

5. Perform the following loop until the MQGET call fails:
   a. Print a line of the report showing:
      - The sequence number of the message (this is a count of the browse operations).
      - The true length of the message (not the truncated length). This value is returned in the `DataLength` field of the MQGET call.
      - The first 80 bytes of the message data.
   b. Reset the `MsqId` and `CorrelId` fields of the MQMD structure to nulls
   c. Browse the next message, using the MQGET call with these two options:
      - MQGMO_BROWSE_NEXT
      - MQGMO_ACCEPT_TRUNCATED_MSG
6. If the MQGET call fails, test the reason code to see if the call has failed because the browse cursor has got to the end of the queue. In this case, print the `End of report` message and go to step "7" on page 1217 ; otherwise, print the completion and reason codes, close the print data set, and stop processing.
7. Close the queue using the MQCLOSE call with the object handle returned in step "3" on page 1216.
8. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step "2" on page 1216.
9. Close the print data set and stop processing.

z/OS *Language-dependent design considerations on z/OS*

Source modules are provided for the Browse sample in four programming languages.

There are two main differences between the source modules:

- When testing the parameters passed from the run JCL, the COBOL, PL/I, and assembler-language modules search for the comma character (,). If the JCL passes `PARM=(,LOCALQ1)`, the application attempts to open queue LOCALQ1 on the default queue manager. If there is no name after the comma (or no comma), the application returns an error. The C module does not search for the comma character. If the JCL passes a single parameter (for example, `PARM=('LOCALQ1')`), the C module uses this as a queue name on the default queue manager.
- To keep the assembler-language module simple, it uses the date format *yy/ddd* (for example, 05/116) when it creates the print report. The other modules use the calendar date in *mm/dd/yy* format.

z/OS *The Print Message sample on z/OS*

The Print Message sample is a batch application that demonstrates how to remove all the messages from a queue using the MQGET call.

The Print Message sample uses three parameters:

1. The name of the queue manager
2. The name of the source queue
3. An optional parameter for properties

It also prints, for each message, the fields of the message descriptor, followed by the message data. The program prints the data both in hexadecimal and as characters (if they are printable). If a character is not printable, the program replaces it with a period character (.). You can use the program when diagnosing problems with an application that is putting messages on a queue.

Permissible values for the property parameter are:

Table 185. Valeurs admises pour le paramètre de propriété	
Valeur	**Comportement**
0	Comportement par défaut. Les propriétés qui sont distribuées à l'application dépendent de l'attribut de file d'attente **PropertyControl** à partir duquel le message est extrait.

1	Un descripteur de message est créé et utilisé avec MQGET. Les propriétés du message, à l'exception de celles contenues dans le descripteur de message (ou l'extension), sont affichées de la même manière que le descripteur de message. Exemple :
	`****Message properties****`
	`  property name: property value`
	Ou si aucune propriété n'est disponible:
	`****Message properties****`
	`  None`
	Les valeurs numériques sont affichées à l'aide de `printf`, les valeurs de chaîne sont entourées de guillemets simples et les chaînes d'octets sont entourées de X et de guillemets simples, comme pour le descripteur de message.

2	MQGMO_NO_PROPERTIES est indiqué, de sorte que seules les propriétés de descripteur de message soient renvoyées.
3	MQGMO_PROPERTIES_FORCE_MQRFH2 est spécifié afin que toutes les propriétés soient renvoyées dans les données de message.
4	MQGMO_PROPERTIES_COMPATIBILITY est spécifié, de sorte que toutes les propriétés puissent être renvoyées selon qu'une propriété IBM MQ est incluse ou non, sinon les propriétés sont supprimées.

You can change the application so that it browses the messages, rather than removing them from the queue. To do this, compile with the option of -DBROWSE, to define the BROWSE macro, as indicated in “Design of the Print Message sample on z/OS” on page 1219. Executable code is provided for you in the SCSQLOAD library. Module CSQ4BCG0 is built with -DBROWSE; module CSQ4BCG1 destructively reads the queue.

The application has a single source program, which is written in the C language. Sample run JCL code is also supplied (see Table 174 on page 1203).

To start the application, edit and run the sample run JCL, as described in “Preparing and running sample applications for the batch environment on z/OS” on page 1202. When you run the application (and there are some messages on the queue), the output data set looks like that in Figure 139 on page 1219.

```
CSQ4BCG1 - starts here
**********************

MQCONN to MQ1E
MQOPEN - 'TEST.QUEUE'
MQCRTMH


MQGET of message number 1
****Message descriptor****
StrucId : 'MD ' Version : 2
Report  : 0 MsgType : 8
Expiry  : -1 Feedback : 0
Encoding : 785 CodedCharSetId : 500
Format : 'MQSTR  '
Priority : 0 Persistence : 0
MsgId : X'C3E2D840D4D8F1C540404040404040C1EA537F03167D88'
CorrelId : X'C3E2D840D4D8F1C540404040404040C1EA537F0317A928'
BackoutCount : 0
ReplyToQ    : '                          '
ReplyToQMgr  : ''
** Identity Context
UserIdentifier : 'FRED    '
AccountingToken :
X'0000000000000000000000000000000000000000000000000000000000000000'
ApplIdentityData : '                '
** Origin Context
PutApplType  : '2'
PutApplName  : 'FRED6            '
PutDate : '20080207'  PutTime : '17373745'
ApplOriginData : '  '
GroupId : X'000000000000000000000000000000000000000000000000'
MsgSeqNumber  : '1'
Offset     : '0'
MsgFlags    : '0'
OriginalLength : '-1'

****Message properties****
None

****  Message   ****

length - 30 bytes

00000000: E388 89A2 4089 A240 8140 A289 9497 9385 'This is a simple'
00000010: 40A3 85A2 A340 9485 A2A2 8187 855A   ' test message! '



No more messages
MQDLTMH
MQCLOSE
MQDISC
```

Figure 139. Example of a report from the Print Message sample application

z/OS *Design of the Print Message sample on z/OS*

The Print Message sample application uses a single program written in the C language.

The flow through the program logic is:

1. Check that the names of the queue manager and queue have been passed from the run JCL. If they have not, print an error message and stop processing.
2. Connect to the queue manager using the MQCONN call. If this call is not successful, print the completion and reason codes and stop processing; otherwise print the name of the queue manager.
3. Open the queue using the MQOPEN call with the MQOO_INPUT_SHARED option.

   Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When you do this, the macro preprocessor adds the line in the program that selects the MQOO_BROWSE option in the compilation.

On input to this call, the program uses the connection handle returned in step “2” on page 1219. For the object descriptor structure (MQOD), it uses the default values for all the fields except the queue name (which was passed in step “1” on page 1219). If this call is not successful, print the completion and reason codes and stop processing; otherwise, print the name of the queue.

4. If you use a message handle to obtain the message properties use MQCRTMH to create such a handle for use with subsequent MQGET calls. If this call is not successful, print the completion and reason codes and stop processing.
5. Set the get message options to reflect the request action for any message properties.
6. Perform the following loop until the MQGET call fails:
   a. Initialize the buffer to blanks so that the message data does not get corrupted by any data already in the buffer.
   b. Set the `MsgId` and `CorrelId` fields of the MQMD structure to nulls so that the MQGET call selects the first message from the queue.
   c. Get a message from the queue, using the MQGET call. On input to this call, the program specifies:
      - The connection and object handles from steps “2” on page 1219 and “3” on page 1219.
      - An MQMD structure with all fields set to their initial values. (`MsgId` and `CorrelId` are reset to nulls for each MQGET call.)
      - The option MQGMO_NO_WAIT.

        Note: If you want the application to browse the messages rather than remove them from the queue, compile the sample with -DBROWSE, or, add `#define BROWSE` at the beginning of the source. When you do this, the macro preprocessor adds the line in the program that selects the MQGMO_BROWSE_NEXT option to the compilation. When this option is used on a call against a queue for which no browse cursor has previously been used with the current object handle, the browse cursor is positioned logically before the first message.
      - A buffer of size 64KB to hold the data copied from the message.
   d. Call the printMD subroutine. This prints the name of each field in the message descriptor, followed by its contents.
   e. If you created a message handle in step “4” on page 1220 call the printProperties subroutine to display any message properties.
   f. Print the length of the message, followed by the message data. Each line of message data is in this format:
      - Relative position (in hexadecimal) of this part of the data
      - 16 bytes of hexadecimal data
      - The same 16 bytes of data in character format, if it is printable (nonprintable characters are replaced by periods)
7. If the MQGET call fails, test the reason code to see if the call failed because there are no more messages on the queue. In this case, print the message: No more messages; otherwise, print the completion and reason codes. In both cases, go to step “9” on page 1220.

   Note: The MQGET call fails if it finds a message that has more than 64KB of data. To change the program to handle larger messages, you could do one of the following:
   - Add the MQGMO_ACCEPT_TRUNCATED_MSG option to the MQGET call, so that the call gets the first 64KB of data and discards the remainder
   - Make the program leave the message on the queue when it finds one with this amount of data
   - Increase the size of the buffer
8. If you created a message handle in step “4” on page 1220 call MQDLTMH to delete it.
9. Close the queue using the MQCLOSE call with the object handle returned in step “3” on page 1219.
10. Disconnect from the queue manager using the MQDISC call with the connection handle returned in step “2” on page 1219.

z/OS *The Queue Attributes sample on z/OS*

The Queue Attributes sample is a conversational-mode CICS application that demonstrates the use of the MQINQ and MQSET calls.

It shows how to inquire about the values of the **InhibitPut** and **InhibitGet** attributes of queues, and how to change them so that programs cannot put messages on, or get messages from, a queue. You might want to *lock* a queue in this way when you are testing a program.

To prevent accidental interference with your own queues, this sample works only on a queue object that has the characters CSQ4SAMP in the first eight bytes of its name. However, the source code includes comments to show you how to remove this restriction.

Source programs are supplied in the COBOL, assembler, and C languages (see Table 180 on page 1208).

The assembler-language version of the sample uses reenterable code. To do this, you will notice that the code for each MQI call in that version of the sample includes the MF keyword; for example:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

(The VL keyword means that you can use the CICS Execution Diagnostic Facility (CEDF) supplied transaction for debugging the program.) For more information about writing reenterable programs, see Coding in System/390 assembler language.

To start the application, start your CICS system and use the following CICS transactions:

- For COBOL, MVC1
- For assembler language, MAC1
- For C, MCC1

You can change the name of any of these transactions by changing the CSD data set mentioned in step 3.

Design of the sample

When you start the sample, it displays a screen map that has fields for:

- Name of the queue
- User request (valid actions are: inquire, allow, or inhibit)
- Current status of put operations for the queue
- Current status of get operations for the queue

The first two fields are for user input. The last two fields are filled by the application: they show the word `INHIBITED` or the word `ALLOWED`.

The application validates the values that you enter in the first two fields. It checks that the queue name starts with the characters CSQ4SAMP and that you entered one of the three valid requests in the Action field. The application converts all your input to uppercase, so you cannot use any queues with names that contain lowercase characters.

If you enter `inquire` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_INQUIRE option
2. Call MQINQ using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT
3. Close the queue using the MQCLOSE call
4. Analyze the attributes that are returned in the **IntAttrs** parameter of the MQINQ call and move the words INHIBITED or ALLOWED, as appropriate, to the relevant screen fields

If you enter `inhibit` in the **Action** field, the flow through the program logic is:

1. Open the queue using the MQOPEN call with the MQOO_SET option
2. Call MQSET using the selectors MQIA_INHIBIT_GET and MQIA_INHIBIT_PUT, and with the values MQQA_GET_INHIBITED and MQQA_PUT_INHIBITED in the **IntAttrs** parameter

3. Close the queue using the MQCLOSE call
4. Move the word INHIBITED to the relevant screen fields

If you enter `allow` in the **Action** field, the application performs similar processing to that for an inhibit request. The only differences are the settings of the attributes and the words displayed on the screen.

When the application opens the queue, it uses the default connection handle to the queue manager. (CICS establishes a connection to the queue manager when you start your CICS system.) The application can trap the following errors at this stage:

- The application is not connected to the queue manager
- The queue does not exist
- The user is not authorized to access the queue
- The application is not authorized to open the queue

For other MQI errors, the application displays the completion and reason codes.

z/OS *The Mail Manager sample on z/OS*

The Mail Manager sample application is a suite of programs that demonstrates sending and receiving messages, both within a single environment and across different environments. The application is a simple electronic mailing system that allows users to exchange messages, even if they use different queue managers.

The application demonstrates how to create queues using the MQOPEN call and by putting IBM MQ for z/OS commands on the system-command input queue.

Three versions of the application are provided:

- A CICS application written in COBOL
- A TSO application written in COBOL
- A TSO application written in C

z/OS *Preparing the Mail Manager sample on z/OS*

The Mail Manager is provided in versions that run in two environments. The preparation that you must carry out before you run the application depends on the environment that you want to use.

Users can access mail queues and nickname queues from both TSO and CICS so long as their sign-on user IDs are the same on each system.

Before you can send messages to another queue manager, you must set up a message channel to that queue manager. To do this, use the channel control function of IBM MQ, described in Channel control function.

Preparing the sample for the TSO environment

Follow these steps:

1. Prepare the sample as described in “Preparing sample applications for the TSO environment on z/OS” on page 1205.
2. Tailor the CLIST provided for the sample to define:
   - The location of the panels
   - The location of the message file
   - The location of the load modules
   - The name of the queue manager that you want to use with the application

   A separate CLIST is provided for each language version of the sample:
   - For the COBOL version: CSQ4RVD1
   - For the C version: CSQ4RCD1

3. Ensure that the queues used by the application are available on the queue manager. (The queues are defined in CSQ4CVD.)

Note: VS COBOL II does not support multitasking with ISPF. This means that you cannot use the Mail Manager sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Running the Mail Manager sample on z/OS*

To start the sample in the CICS Transaction Server for z/OS environment, run transaction MAIL. If you have not already signed on to CICS, the application prompts you to enter a user ID to which it can send your mail.

When you start the application, it opens your mail queue. If this queue does not exist, the application creates one for you. Mail queues have names of the form CSQ4SAMP.MAILMGR. *userid*, where *userid* depends on the environment:

In TSO
The user's TSO ID

In CICS
The user's CICS sign-on or the user ID entered by the user when prompted when the Mail Manager started

All parts of the queue names that the Mail Manager uses must be uppercase.

The application then presents a menu panel that has options for:

- Read incoming mail
- Send mail
- Create nickname

The menu panel also shows you how many messages are waiting on your mail queue. Each of the menu options displays a further panel:

Read incoming mail
The Mail Manager displays a list of the messages that are on your mail queue. (Only the first 99 messages on the queue are displayed.) For an example of this panel, see Figure 142 on page 1227. When you select a message from this list, the contents of the message are displayed (see Figure 143 on page 1228).

Send mail
A panel prompts you to enter:

- The name of the user to whom you want to send a message
- The name of the queue manager that owns their mail queue
- The text of your message

In the user name field, you can enter either a user ID or a nickname that you created using the Mail Manager. You can leave the queue manager name field blank if the user's mail queue is owned by the same queue manager that you are using, and you must leave it blank if you entered a nickname in the user name field:

- If you specify only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If you specify both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

For example, if you want to send a message to user JONESM on remote queue manager QM12, you could send them a message in either of two ways:

- Use both fields to specify user JONESM at queue manager QM12.
- Define a nickname (for example, MARY) for that user and send them a message by putting MARY in the user name field and nothing in the queue manager name field.

Create nickname

You can define an easy-to-remember name that you can use when you send a message to another user who you contact frequently. You are prompted to enter the user ID of the other user and the name of the queue manager that owns their mail queue.

Nicknames are queues that have names of the form CSQ4SAMP.MAILMGR. *userid.nickname*, where *userid* is your own user ID and *nickname* is the nickname that you want to use. With names structured in this way, users can each have their own set of nicknames.

The type of queue that the program creates depends on how you complete the fields of the Create Nickname panel:

- If you specify only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.
- If you specify both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

For example, if your own user ID is SMITHK and you create a nickname called MARY for user JONESM (who uses the remote queue manager QM12), the nickname program creates a local definition of a remote queue named CSQ4SAMP.MAILMGR.SMITHK.MARY. This definition resolves to Mary's mail queue, which is CSQ4SAMP.MAILMGR.JONESM at queue manager QM12. If you are using queue manager QM12 yourself, the program instead creates an alias queue of the same name (CSQ4SAMP.MAILMGR.SMITHK.MARY).

The C version of the TSO application makes greater use of ISPF's message-handling capabilities than does the COBOL version. You might notice that different error messages are displayed by the C and COBOL versions.

z/OS *Design of the Mail Manager sample on z/OS*

The following sections describe each of the programs that make up the Mail Manager sample application.

The relationships between the programs and the panels that the application uses is shown in Figure 140 on page 1225 for the TSO version, and Figure 141 on page 1226 for the CICS Transaction Server for z/OS version.

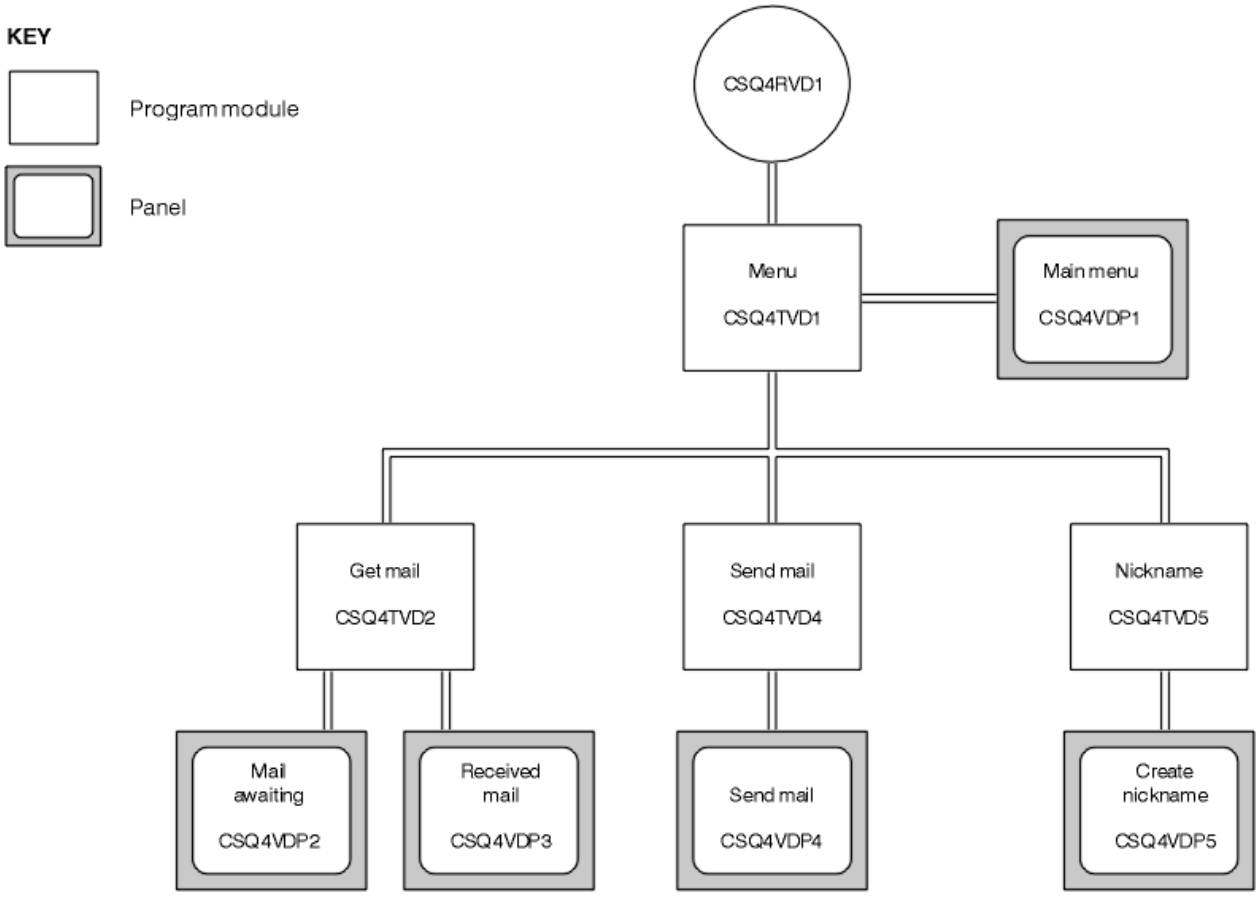


Figure 140. Programs and panels for the TSO versions of the Mail Manager

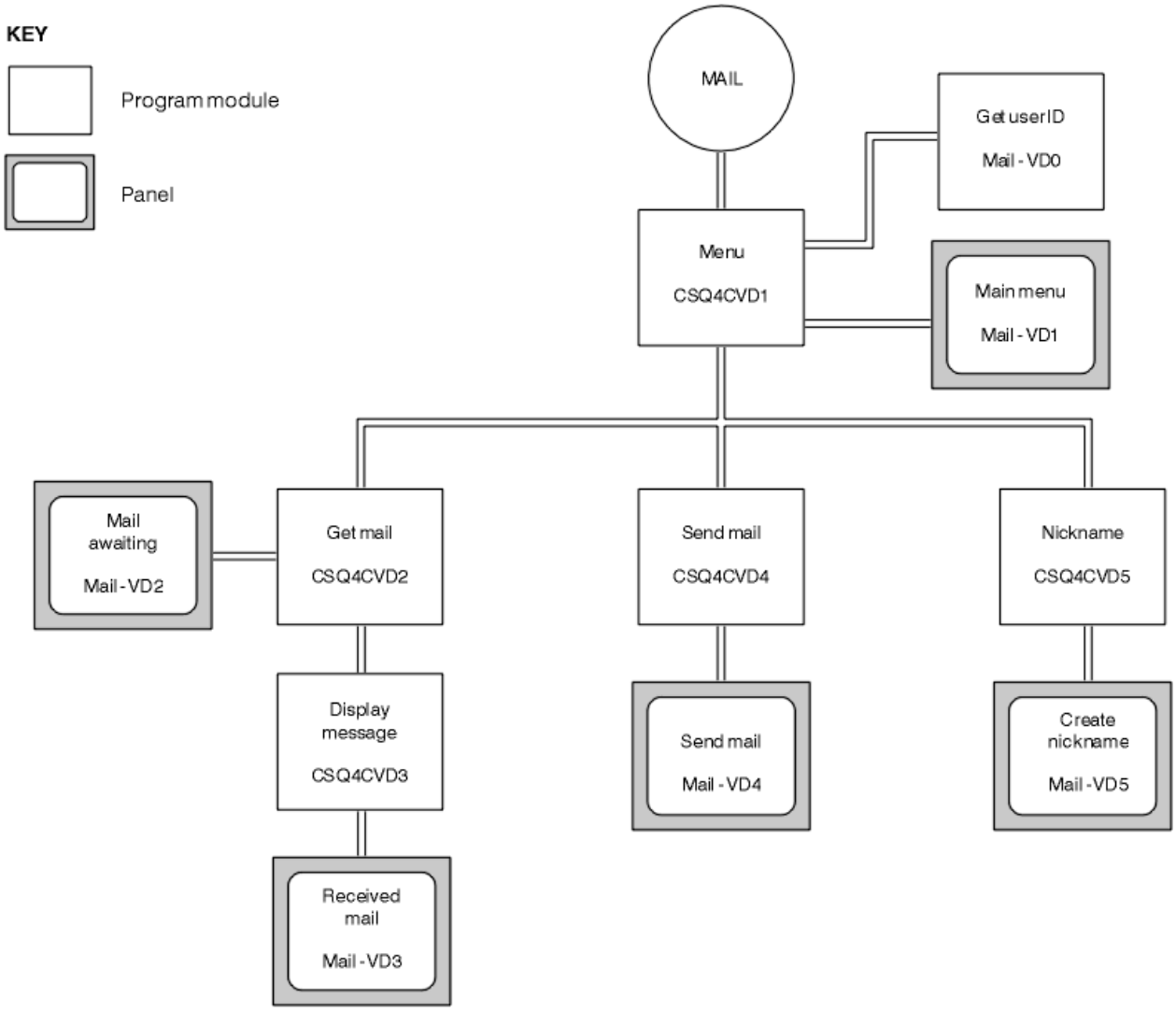


Figure 141. Programs and panels for the CICS version of the Mail Manager

z/OS *Menu program on z/OS*

In the TSO environment, the menu program is invoked by the CLIST. In the CICS environment, the program is invoked by transaction MAIL.

The menu program (CSQ4TVD1 for TSO, CSQ4CVD1 for CICS) is the initial program in the suite. It displays the menu (CSQ4VDP1 for TSO, VD1 for CICS) and invokes the other programs when they are selected from the menu.

The program first obtains the user's ID:

- In the CICS version of the program, if the user has signed on to CICS, the user ID is obtained by using the CICS command ASSIGN USERID. If the user has not signed on, the program displays the sign on panel (CSQ4VD0) to prompt the user to enter a user ID. There is no security processing within this program; the user can give any user ID.
- In the TSO version, the user's ID is obtained from TSO in the CLIST. It is passed to the menu program as a variable in the ISPF shared pool.

After the program has obtained the user ID, it checks to ensure that the user has a mail queue (CSQ4SAMP.MAILMGR. *userid*). If a mail queue does not exist, the program creates one by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QLOCAL. The object definition that this command uses sets the maximum depth of the queue to 9999 messages.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue. To do this, the program uses the MQOPEN call, specifying the SYSTEM.DEFAULT.MODEL.QUEUE as the template for the dynamic queue. The queue manager creates the temporary dynamic queue with a name that has the prefix CSQ4SAMP; the remainder of the name is generated by the queue manager.

The program then opens the user's mail queue and finds the number of messages on the queue by inquiring about the current depth of the queue. To do this, the program uses the MQINQ call, specifying the MQIA_CURRENT_Q_DEPTH selector.

The program then performs a loop that displays the menu and processes the selection that the user makes. The loop is stopped when the user presses the PF3 key. When a valid selection is made, the appropriate program is started; otherwise an error message is displayed.

z/OS *Get-mail and display-message programs on z/OS*

In the TSO versions of the application, the get-mail and display-message functions are performed by the same program (CSQ4TVD2). In the CICS version of the application, these functions are performed by separate programs (CSQ4CVD2 and CSQ4CVD3).

The Mail Awaiting panel (CSQ4VDP2 for TSO, VD2 for CICS ; see Figure 142 on page 1227 for an example) shows all the messages that are on the user's mail queue. To create this list, the program uses the MQGET call to browse all the messages on the queue, saving information about each one. In addition to the information displayed, the program records the `MsgId` and `CorrelId` of each message.

```
---------------------- IBM MQ for z/OS Sample Programs ------- ROW 16 OF 29
COMMAND ==>                          Scroll  ===> PAGE
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Mail Awaiting

Msg   Mail    Date    Time
No    From    Sent    Sent
16
16   Deleted
17   JOHNJ   01/06/1993 12:52:02
18   JOHNJ   01/06/1993 12:52:02
19   JOHNJ   01/06/1993 12:52:03
20   JOHNJ   01/06/1993 12:52:03
21   JOHNJ   01/06/1993 12:52:03
22   JOHNJ   01/06/1993 12:52:04
23   JOHNJ   01/06/1993 12:52:04
24   JOHNJ   01/06/1993 12:52:04
25   JOHNJ   01/06/1993 12:52:05
26   JOHNJ   01/06/1993 12:52:05
27   JOHNJ   01/06/1993 12:52:05
28   JOHNJ   01/06/1993 12:52:06
29   JOHNJ   01/06/1993 12:52:06
```

Figure 142. Example of a panel showing a list of waiting messages

From the Mail Awaiting panel the user can select one message and display the contents of the message (see Figure 143 on page 1228 for an example). The program uses the MQGET call to remove this message from the queue, using the `MsgId` and `CorrelId` that the program noted when it browsed all the messages. This MQGET call is performed using the MQGMO_SYNCPOINT option. The program displays the contents of the message, then declares a syncpoint: this commits the MQGET call, so the message now no longer exists.

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                                |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
|                                    |
'------------------------------------------------------------------------'
```

Figure 143. Example of a panel showing the contents of a message

An obvious extension to the function provided by the Mail Manager is to give the user the option to leave the message on the queue after viewing its contents. To do this, you would have to back out the MQGET call that removes the message from the queue, after displaying the message.

z/OS *Send-mail program on z/OS*

When the user has completed the Send Mail panel (CSQ4VDP4 for TSO, VD4 for CICS), the send-mail program (CSQ4TVD4 for TSO, CSQ4CVD4 for CICS) puts the message on the receiver's mail queue.

To do this, the program uses the MQPUT1 call. The destination of the message depends on how the user has filled the fields in the Send Mail panel:

- If the user has specified only a user name, the program first assumes that the name is a nickname, and sends the message to the object defined by that name. If there is no such nickname, the program attempts to send the message to a local queue of that name.
- If the user has specified both a user name and a queue manager name, the program sends the message to the mail queue that is defined by those two names.

The program does not accept blank messages, and it removes leading blanks from each line of the message text.

If the MQPUT1 call is successful, the program displays a message that shows that the user name and queue manager name to which the message was put. If the call is unsuccessful, the program checks specifically for the reason codes that indicate the queue or the queue manager do not exist; these are MQRC_UNKNOWN_OBJECT_NAME and MQRC_UNKNOWN_OBJECT_Q_MGR. The program displays its own error message for each of these errors; for other errors, the program displays the completion and reason codes returned by the call.

z/OS *Nickname program on z/OS*

When the user defines a nickname, the nickname program (CSQ4TVD5 for TSO, CSQ4CVD5 for CICS) creates a queue that has the nickname as part of its name.

The program does this by putting a message on the system-command input queue. The message contains the IBM MQ for z/OS command DEFINE QALIAS or DEFINE QREMOTE. The type of queue that the program creates depends on how the user has filled the fields of the Create Nickname panel (CSQ4VDP5 for TSO, VD5 for CICS):

- If the user has specified only a user name, or the queue manager name is the same as that of the queue manager to which the Mail Manager is connected, the program creates an alias queue.

- If the user has specified both a user name and a queue manager name (and the queue manager is not the one to which the Mail Manager is connected), the program creates a local definition of a remote queue. The program does not check the existence of the queue to which this definition resolves, or even that the remote queue manager exists.

The program also creates a temporary dynamic queue to handle replies from the system-command input queue.

If the queue manager cannot create the nickname queue for a reason that the program expects (for example, the queue already exists), the program displays its own error message. If the queue manager cannot create the queue for a reason that the program does not expect, the program displays up to two of the error messages that are returned to the program by the command server.

Note: For each nickname, the nickname program creates only an alias queue or a local definition of a remote queue. The local queues to which these queue names resolve are created only when the user ID that is contained in the nickname is used to start the Mail Manager application.

z/OS *The Credit Check sample on z/OS*

The Credit Check sample application is a suite of programs that demonstrates how to use many of the features provided by IBM MQ for z/OS. It shows how the many component programs of an application can pass messages to each other using message queuing techniques.

The sample can run as a stand-alone CICS application. However, to demonstrate how to design a message queuing application that uses the facilities provided by both the CICS and IMS environments, one module is also supplied as an IMS batch message processing program. This extension to the sample is described in “The IMS extension to the Credit Check sample on z/OS” on page 1239.

You can also run the sample on more than one queue manager, and send messages between each instance of the application. To do so, see “The Credit Check sample with multiple queue managers on z/OS” on page 1238.

The CICS programs are delivered in C and COBOL. The single IMS program is delivered only in C. The supplied data sets are shown in Table 182 on page 1209 and Table 184 on page 1210.

The application demonstrates a method of assessing the risk when bank customers ask for loans. The application shows how a bank could work in two ways to process loan requests:

- When dealing directly with a customer, bank staff want immediate access to account and credit-risk information.
- When dealing with written applications, bank staff can submit a series of requests for account and credit-risk information, and deal with the replies at a later time.

The financial and security details in the application have been kept simple so that the message queuing techniques are clear.

z/OS *Preparing and running the Credit Check sample on z/OS*

To prepare and run the Credit Check sample, perform the following steps:

1. Create the VSAM data set that holds information about some example accounts. Do this by editing and running the JCL supplied in data set CSQ4FILE.
2. Perform the steps in “Preparing the sample applications for the CICS environment on z/OS” on page 1207. (The additional steps that you must perform if you want to use the IMS extension to the sample are described in “The IMS extension to the Credit Check sample on z/OS” on page 1239.)
3. Start the CKTI trigger monitor (supplied with IBM MQ for z/OS) against queue CSQ4SAMP.INITIATION.QUEUE, using the CICS transaction CKQC.
4. To start the application, start your CICS system and use the transaction MVB1.
5. Select **Immediate** or **Batch** inquiry from the first panel.

   The immediate and batch inquiry panels are similar; Figure 144 on page 1230 shows the Immediate Inquiry panel.

```
CSQ4VB2       IBM MQ for z/OS Sample Programs

Credit Check - Immediate Inquiry

Specify details of the request, then press Enter.
Name . . . . . _____________________
Social security number ___ __ ____
Bank account name . . ______________________________
Account number . . . . __________
Amount requested . . . 012345
Response from CHECKING ACCOUNT for name : ____________________
Account information not found
Credit worthiness index - NOT KNOWN
..
..
..
..
..
..
..
..
..
MESSAGE LINE
F1=Help F3=Exit F5=Make another inquiry
```

Figure 144. Immediate Inquiry panel for the Credit Check sample application

6. Enter an account number and loan amount in the appropriate fields. See "Entering information in the inquiry panels" on page 1230 for guidance on what information to enter in these fields.

Entering information in the inquiry panels

The Credit Check sample application checks that the data you enter in the **Amount requested** field of the inquiry panels is in the form of integers.

If you enter one of the following account numbers, the application finds the appropriate account name, average account balance, and credit worthiness index in the VSAM data set CSQ4BAQ:

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

You can enter any, or no, information in the other fields. The application retains any information that you enter and returns the same information in the reports that it generates.

z/OS *Design of the Credit Check sample on z/OS*

This section describes the design of each of the programs that make up the Credit Check sample application.

For more information about of some of the techniques that were considered during the design of the application, see "Design considerations for the Credit check sample on z/OS" on page 1236.

Figure 145 on page 1231 shows the programs that make up the application, and also the queues that these programs serve. In this figure, the prefix CSQ4SAMP has been omitted from all the queue names to make the figure easier to understand.

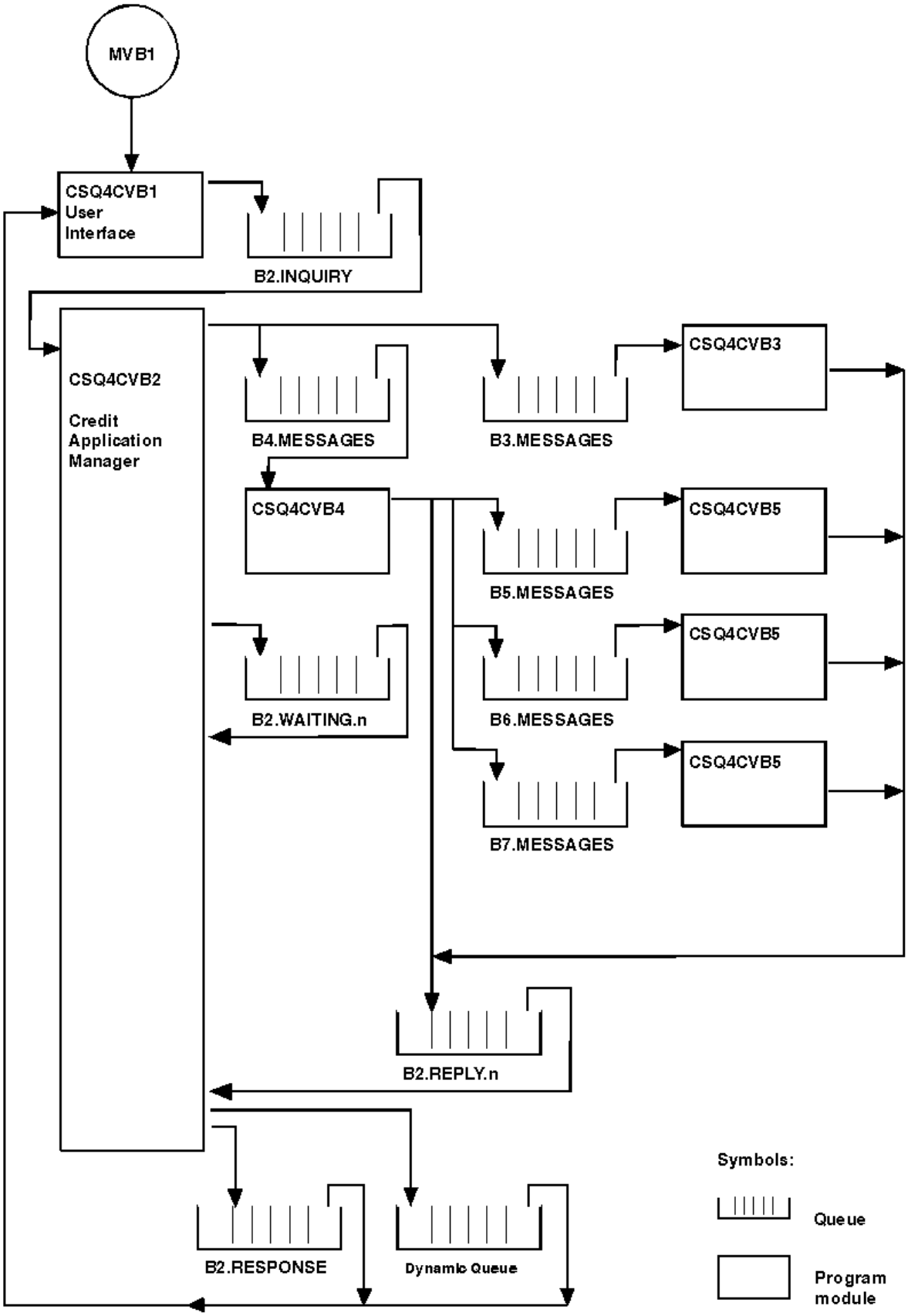


Figure 145. Programs and queues for the Credit Check sample application (COBOL programs only)

z/OS *User interface program (CSQ4CVB1) on z/OS*

When you start the conversational-mode CICS transaction MVB1, this starts the user interface program for the application.

This program puts inquiry messages on queue CSQ4SAMP.B2.INQUIRY and gets replies to those inquiries from a reply-to queue that it specifies when it makes the inquiry. From the user interface you can submit either immediate or batch inquiries:

- For immediate inquiries, the program creates a temporary dynamic queue that it uses as a reply-to queue. This means that each inquiry has its own reply-to queue.
- For batch inquiries, the user-interface program gets replies from the queue CSQ4SAMP.B2.RESPONSE. For simplicity, the program gets replies for all its inquiries from this one reply-to queue. It is easy to see that a bank might want to use a separate reply-to queue for each user of MVB1, so that they could each see replies to only those inquiries that they had initiated.

Important differences between the properties of messages used in the application when in batch and immediate mode are:

- For batch working, the messages have a low priority, so they are processed after any loan requests that are entered in immediate mode. Also, the messages are persistent, so they are recovered if the application or the queue manager has to restart.
- For immediate working, the messages have a high priority, so they are processed before any loan requests that are entered in batch mode. Also, messages are not persistent so they are discarded if the application or the queue manager has to restart.

However, in all cases, the properties of loan request messages are propagated throughout the application. So, for example, all messages that result from a high-priority request will also have a high priority.

z/OS *Credit application manager (CSQ4CVB2) on z/OS*

The Credit Application Manager (CAM) program performs most of the processing for the Credit Check application.

The CAM is started by the CKTI trigger monitor (supplied with IBM MQ for z/OS) when a trigger event occurs on either queue CSQ4SAMP.B2.INQUIRY or queue CSQ4SAMP.B2.REPLY. *n*, where *n* is an integer that identifies one of a set of reply queues. The trigger message contains data that includes the name of the queue on which the trigger event occurred.

The CAM uses queues with names of the form CSQ4SAMP.B2.WAITING.n to store information about inquiries that it is processing. The queues are named so that they are each paired with a reply-to queue; for example, queue CSQ4SAMP.B2.WAITING.3 contains the input data for a particular inquiry, and queue CSQ4SAMP.B2.REPLY.3 contains a set of reply messages (from programs that query databases) all relating to that same inquiry. To understand the reasons behind this design, see “Separate inquiry and reply queues in the CAM” on page 1236.

Startup logic

If the trigger event occurs on queue CSQ4SAMP.B2.INQUIRY, the CAM opens the queue for shared access. It then tries to open each reply queue until a free one is found. If it cannot find a free reply queue, the CAM logs the fact and terminates normally.

If the trigger event occurs on queue CSQ4SAMP.B2.REPLY.n, the CAM opens the queue for exclusive access. If the return code reports that the object is already in use, the CAM terminates normally. If any other error occurs, the CAM logs the error and terminates. The CAM opens the corresponding waiting queue and the inquiry queue, then starts getting and processing messages. From the waiting queue, the CAM recovers details of partially-completed inquiries.

For the sake of simplicity in this sample, the names of the queues used are held in the program. In a business environment, the queue names would probably be held in a file accessed by the program.

Getting a message from the enquiry queue

The CAM first attempts to get a message from the inquiry queue using the MQGET call with the MQGMO_SET_SIGNAL option. If a message is available immediately, the message is processed; if no message is available, a signal is set.

The CAM then attempts to get a message from the reply queue, again using the MQGET call with the same option. If a message is available immediately, the message is processed; otherwise a signal is set.

When both signals are set, the program waits until one of the signals is posted. If a signal is posted to indicate that a message is available, the message is retrieved and processed. If the signal expires or the queue manager is terminating, the program terminates.

Processing the message retrieved by the CAM

A message retrieved by the CAM can be one of four types:

- An inquiry message
- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as described in "Processing the message retrieved by the CAM on z/OS" on page 1233.

Sending an answer

When the CAM has received all the replies it is expecting for an inquiry, it processes the replies and creates a single response message. It consolidates into one message all the data from all reply messages that have the same `CorrelId`. This response is put on the reply-to queue specified in the original loan request. The response message is put within the same unit of work that contains the retrieval of the final reply message. This is to simplify recovery by ensuring that there is never a completed message on queue CSQ4SAMP.B2.WAITING.n.

Recovery of partially-completed inquiries

The CAM copies onto queue CSQ4SAMP.B2.WAITING.n all the messages that it receives. It sets the fields of the message descriptor like this:

- *`Priority`* is determined by the type of message:
  - For request messages, priority = 3
  - For datagrams, priority = 2
  - For reply messages, priority = 1
- *`CorrelId`* is set to the *`MsgId`* of the loan request message
- Other MQMD fields are copied from those of the received message

When an inquiry has been completed, the messages for a specific inquiry are removed from the waiting queue during answer processing. Therefore, at any time, the waiting queue contains all messages relevant to in-progress inquiries. These messages are used to recover details of in-progress inquiries if the program has to restart. The different priorities are set so that inquiry messages are recovered before propagations or reply messages.

z/OS *Processing the message retrieved by the CAM on z/OS*

A message retrieved by the Credit Application Manager (CAM) can be one of four types. The way in which the CAM processes a message depends on its type.

A message retrieved by the CAM can be one of four types:

- An inquiry message

- A reply message
- A propagation message
- An unexpected or unwanted message

The CAM processes these messages as follows:

Inquiry message
Inquiry messages come from the user interface program. It creates an inquiry message for each loan request.

For all loan requests, the CAM requests the average balance of the customer's checking account. It does this by putting a request message on alias queue CSQ4SAMP.B2.OUTPUT.ALIAS. This queue name resolves to queue CSQ4SAMP.B3.MESSAGES, which is processed by the checking-account program, CSQ4CVB3. When the CAM puts a message on this alias queue, it specifies the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. An alias queue is used here so that program CSQ4CVB3 can easily be replaced by another program that processes a base queue of a different name. To do this, you redefine the alias queue so that its name resolves to the new queue. Also, you could assign differing access authorities to the alias queue and to the base queue.

If a user requests a loan that is larger than 10000 units, the CAM initiates checks on other databases as well. It does this by putting a request message on queue CSQ4SAMP.B4.MESSAGES, which is processed by the distribution program, CSQ4CVB4. The process serving this queue propagates the message to queues served by programs that have access to other records such as credit card history, savings accounts, and mortgage payments. The data from these programs is returned to the reply-to queue specified in the put operation. Additionally, a propagation message is sent to the reply-to queue by this program to specify how many propagation messages have been sent.

In a business environment, the distribution program would probably reformat the data provided to match the format required by each of the other types of bank account.

Any of the queues referred to can be on a remote system.

For each inquiry message, the CAM initiates an entry in the memory-resident Inquiry Record Table (IRT). This record contains:

- The `MsgId` of the inquiry message
- In the ReplyExp field, the number of responses expected (equal to the number of messages sent)
- In the ReplyRec field, the number of replies received (zero at this stage)
- In the PropsOut field, an indication of whether a propagation message is expected

The CAM copies the inquiry message onto the waiting queue with:

- `Priority` set to 3
- `CorrelId` set to the `MsgId` of the inquiry message
- The other message-descriptor fields set to those of the inquiry message

Propagation message
A propagation message contains the number of queues to which the distribution program has forwarded the inquiry. The message is processed as follows:

1. Add to the ReplyExp field of the appropriate record in the IRT the number of messages sent. This information is in the message.
2. Increment by 1 the ReplyRec field of the record in the IRT.
3. Decrement by 1 the PropsOut field of the record in the IRT.
4. Copy the message onto the waiting queue. The CAM sets the `Priority` to 2 and the other fields of the message descriptor to those of the propagation message.

Reply message
A reply message contains the response to one of the requests to the checking-account program or to one of the agency-query programs. Reply messages are processed as follows:

1. Increment by 1 the ReplyRec field of the record in the IRT.
2. Copy the message onto the waiting queue with `Priority` set to 1 and the other fields of the message descriptor set to those of the reply message.
3. If ReplyRec = ReplyExp, and PropsOut = 0, set the MsgComplete flag.

Other messages

The application does not expect other messages. However, the application might receive messages broadcast by the system, or reply messages with a unknown `CorrelIds`.

The CAM puts these messages on queue CSQ4SAMP.DEAD.QUEUE, where they can be examined. If this put operation fails, the message is lost and the program continues. For more information about the design of this part of the program, see "How the sample handles unexpected messages" on page 1237.

z/OS *Checking-account program (CSQ4CVB3) on z/OS*

The checking-account program is started by a trigger event on queue CSQ4SAMP.B3.MESSAGES. After it has opened the queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program searches VSAM data set CSQ4BAQ for the account number in the loan request message. It retrieves the corresponding account name, average balance, and credit worthiness index, or notes that the account number is not in the data set.

The program then puts a reply message (using the MQPUT1 call) on the reply-to queue named in the loan request message. For this reply message, the program:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program continues to get messages from the queue until the wait interval expires.

z/OS *Distribution program (CSQ4CVB4) on z/OS*

The distribution program is started by a trigger event on queue CSQ4SAMP.B4.MESSAGES.

To simulate the distribution of the loan request to other agencies that have access to records such as credit card history, savings accounts, and mortgage payments, the program puts a copy of the same message on all the queues in the namelist CSQ4SAMP.B4.NAMELIST. There are three of these queues, with names of the form CSQ4SAMP.B *n*.MESSAGES, where *n* is 5, 6, or 7. In a business application, the agencies could be at separate locations, so these queues could be remote queues. If you want to modify the sample application to show this, see "The Credit Check sample with multiple queue managers on z/OS" on page 1238.

The distribution program performs the following steps:

1. From the namelist, gets the names of the queues that the program is to use. The program does this by using the MQINQ call to inquire about the attributes of the namelist object.
2. Opens these queues and also CSQ4SAMP.B4.MESSAGES.
3. Performs the following loop until there are no more messages on queue CSQ4SAMP.B4.MESSAGES:
   a. Get a message using the MQGET call with the wait option, and with the wait interval set to 30 seconds.
   b. Put a message on each queue listed in the namelist, specifying the name of the appropriate CSQ4SAMP.B2.REPLY.n queue for the reply-to queue. The program copies the *`CorrelId`* of the loan request message to these copy messages, and it uses the MQPMO_PASS_IDENTITY_CONTEXT option on the MQPUT call.
   c. Send a datagram message to queue CSQ4SAMP.B2.REPLY.n to show how many messages it has successfully put.
   d. Declare a syncpoint.

z/OS *Agency-query program (CSQ4CVB5/CSQ4CCB5) on z/OS*

The agency-query program is supplied as both a COBOL program and a C program. Both programs have the same design. This shows that programs of different types can easily coexist within an IBM MQ application, and that the program modules that make up such an application can easily be replaced.

An instance of the program is started by a trigger event on any of these queues:

- For the COBOL program (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- For the C program (CSQ4CCB5), queue CSQ4SAMP.B8.MESSAGES

Note: If you want to use the C program, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace the queue CSQ4SAMP.B7.MESSAGES with CSQ4SAMP.B8.MESSAGES. To do this, you can use any one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command
- The CSQUTIL utility

After it has opened the appropriate queue, this program gets a message from the queue using the MQGET call with the wait option, and with the wait interval set to 30 seconds.

The program simulates the search of an agency's database by searching the VSAM data set CSQ4BAQ for the account number that was passed in the loan request message. It then builds a reply that includes the name of the queue that it is serving and a creditworthiness index. To simplify the processing, the creditworthiness index is selected at random.

When putting the reply message, the program uses the MQPUT1 call and:

- Copies the `CorrelId` of the loan request message
- Uses the MQPMO_PASS_IDENTITY_CONTEXT option

The program sends the reply message to the reply-to queue named in the loan request message. (The name of the queue manager that owns the reply-to queue is also specified in the loan request message.)

z/OS *Design considerations for the Credit check sample on z/OS*

Design considerations for the Credit Check sample.

This topic contains information about:

Separate inquiry and reply queues in the CAM

The application could use a single queue for both inquiries and replies, but it was designed to use separate queues for the following reasons:

- When the program is handling the maximum number of inquiries, further inquiries can be left on the queue. If a single queue is being used, this would have to be taken off the queue and stored elsewhere.
- Other instances of the CAM could be started automatically to service the same inquiry queue if message traffic was high enough to warrant it. But the program must track in-progress inquiries, and to do this,

it must get back all replies to inquiries it has initiated. If only one queue is used, the program would have to browse the messages to see if they were for this program or for another. This would make the operation much less efficient.

The application can support multiple CAMs and can recover in-progress inquiries effectively by using paired reply-to and waiting queues.

- The program can wait on multiple queues effectively by using signaling.

How the sample handles errors

The user interface program handles errors by reporting them directly to the user.

The other programs do not have user interfaces, so they have to handle errors in other ways. Also, in many situations (for example, if an MQGET call fails) these other programs do not know the identity of the user of the application.

The other programs put error messages on a CICS temporary storage queue called CSQ4SAMP. You can browse this queue using the CICS-supplied transaction CEBR. The programs also write error messages to the CICS CSML log.

How the sample handles unexpected messages

When you design a message-queuing application, you must decide how to handle messages that arrive on a queue unexpectedly.

The two basic choices are:

- The application does no more work until it has processed the unexpected message. This probably means that the application notifies an operator, terminates itself, and ensures that it is not restarted automatically (it can do this by setting triggering off). This choice means that all processing for the application can be halted by a single unexpected message, and the intervention of an operator is required to restart the application.
- The application removes the message from the queue it is serving, puts the message in another location, and continues processing. The best place to put this message is on the system dead-letter queue.

If you choose the second option:

- An operator, or another program, should examine the messages that are put on the dead-letter queue to find out where the messages are coming from.
- An unexpected message is lost if it cannot be put on the dead-letter queue.
- A long unexpected message is truncated if it is longer than the limit for messages on the dead-letter queue, or longer than the buffer size in the program.

To ensure that the application smoothly handles all inquiries with minimal effect from outside activities, the Credit Check sample application uses the second option. To allow you to keep the sample separate from other applications that use the same queue manager, the Credit Check sample does not use the system dead-letter queue; instead, it uses its own dead-letter queue. This queue is named CSQ4SAMP.DEAD.QUEUE. The sample truncates any messages that are longer than the buffer area provided for the sample programs. You can use the Browse sample application to browse messages on this queue, or use the Print Message sample application to print the messages together with their message descriptors.

However, if you extend the sample to run across more than one queue manager, unexpected messages, or messages that cannot be delivered, could be put on the system dead-letter queue by the queue manager.

How the sample uses syncpoints

The programs in the Credit Check sample application declare syncpoints to ensure that:

- Only one reply message is sent in response to each expected message

- Multiple copies of unexpected messages are never put on the sample's dead-letter queue
- The CAM can recover the state of all partially completed inquiries by getting persistent messages from its waiting queue

To achieve this, a single unit of work is used to cover the getting of a message, the processing of that message, and any subsequent put operations.

How the sample uses message context information

When the user interface program (CSQ4CVB1) sends messages, it uses the MQPMO_DEFAULT_CONTEXT option. This means that the queue manager generates both identity and origin context information. The queue manager gets this information from the transaction that started the program (MVB1) and from the user ID that started the transaction.

When the CAM sends inquiry messages, it uses the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

When the CAM sends reply messages, it uses the MQPMO_ALTERNATE_USER_AUTHORITY option. This causes the queue manager to use an alternate user ID for its security check when the CAM opens a reply-to queue. The CAM uses the user ID of the submitter of the original inquiry message. This means that users are allowed to see replies to only those inquiries that they have originated. The alternate user ID is obtained from the identity context information in the message descriptor of the original inquiry message.

When the query programs (CSQ4CVB3/4/5) send reply messages, they use the MQPMO_PASS_IDENTITY_CONTEXT option. This means that the identity context information of the message being put is copied from the identity context of the original inquiry message. With this option, origin context information is generated by the queue manager.

Note: The user ID associated with the MVB3/4/5 transactions requires access to the B2.REPLY.n queues. These user IDs might not be the same as those associated with the request being processed. To get around this possible security exposure, the query programs could use the MQPMO_ALTERNATE_USER_AUTHORITY option when putting their replies. This would mean that each individual user of MVB1 needs authority to open the B2.REPLY.n queues.

Use of message and correlation identifiers in the CAM

The application has to monitor the progress of all the live inquiries it is processing at any one time. To do this it uses the unique message identifier of each loan request message to associate all the information that it has about each inquiry.

The CAM copies the `MsgId` of the inquiry message into the `CorrelId` of all the request messages it sends for that inquiry. The other programs in the sample (CSQ4CVB3 - 5) copy the `CorrelId` of each message that they receive into the `CorrelId` of their reply message.

z/OS *The Credit Check sample with multiple queue managers on z/OS*

You can use the Credit Check sample application to demonstrate distributed queuing by installing the sample on two queue managers and CICS systems (with each queue manager connected to a different CICS system).

When the sample program is installed, and the trigger monitor (CKTI) is running on each system, you need to:

1. Set up the communication link between the two queue managers. For information on how to do this, see Configuring distributed queuing.
2. On one queue manager, create a local definition for each of the remote queues (on the other queue manager) that you want to use. These queues can be any of CSQ4SAMP.B *n*.MESSAGES, where *n* is 3, 5, 6, or 7. (These are the queues that are served by the checking-account program and the agency-query program.) For information on how to do this, see DEFINE QREMOTE and DEFINE queues.

3. Change the definition of the namelist (CSQ4SAMP.B4.NAMELIST) so that it contains the names of the remote queues that you want to use. For information on how to do this, see DEFINE NAMELIST.

z/OS *The IMS extension to the Credit Check sample on z/OS*

A version of the checking-account program is supplied as an IMS batch message processing (BMP) program. It is written in the C language.

The program performs the same function as the CICS version, except that to obtain the account information, the program reads an IMS database instead of a VSAM file. If you replace the CICS version of the checking-account program with the IMS version, you see no difference in the method of using the application.

To prepare and run the IMS version you must:

1. Follow the steps in “Preparing and running the Credit Check sample on z/OS” on page 1229.
2. Follow the steps in “Preparing the sample application for the IMS environment on z/OS” on page 1210.
3. Alter the definition of the alias queue CSQ4SAMP.B2.OUTPUT.ALIAS to resolve to queue CSQ4SAMP.B3.IMS.MESSAGES (instead of CSQ4SAMP.B3.MESSAGES). To do this, you can use one of:
   - The IBM MQ for z/OS operations and control panels
   - The ALTER QALIAS command .

Another way of using the IMS checking-account program is to make it serve one of the queues that receives messages from the distribution program. In the delivered form of the Credit Check sample application, there are three of these queues (B5/6/7.MESSAGES), all served by the agency-query program. This program searches a VSAM data set. To compare the use of the VSAM data set and the IMS database, you could make the IMS checking-account program serve one of these queues instead. To do this, you must alter the definition of the namelist CSQ4SAMP.B4.NAMELIST to replace one of the CSQ4SAMP.B *n*.MESSAGES queues with the CSQ4SAMP.B3.IMS.MESSAGES queue. You can use one of:

- The IBM MQ for z/OS operations and control panels
- The ALTER NAMELIST command.

You can then run the sample from CICS transaction MVB1. The user sees no difference in operation or response. The IMS BMP stops either after receiving a stop message or after being inactive for five minutes.

Design of the IMS checking-account program (CSQ4ICB3)

This program runs as a BMP. Start the program using its JCL before any IBM MQ messages are sent to it.

The program searches an IMS database for the account number in the loan request messages. It retrieves the corresponding account name, average balance, and credit worthiness index.

The program sends the results of the database search to the reply-to queue named in the IBM MQ message being processed. The message returned appends the account type and the results of the search to the message received so that the transaction building the response can confirm that the correct query is being processed. The message is in the form of three 79-character groups, as follows:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

When running as a message-oriented BMP, the program drains the IMS message queue, then reads messages from the IBM MQ for z/OS queue and processes them. No information is received from the IMS message queue. The program reconnects to the queue manager after each checkpoint because the handles have been closed.

When running in a batch-oriented BMP, the program continues to be connected to the queue manager after each checkpoint because the handles are not closed.

z/OS *The Message Handler sample on z/OS*

The Message Handler sample TSO application allows you to browse, forward, and delete messages on a queue. The sample is available in C and COBOL.

Preparing and running the sample

Follow these steps:

1. Prepare the sample as described in "Preparing sample applications for the TSO environment on z/OS" on page 1205.
2. Tailor the CLIST (CSQ4RCH1) provided for the sample to define the location of the panels, the location of the message file, and the location of the load modules.

You can use CLIST CSQ4RCH1 to run both the C and the COBOL version of the sample. The supplied version of CSQ4RCH1 runs the C version, and contains instructions on the tailoring necessary for the COBOL version.

Note:

1. There are no sample queue definitions provided with the sample.
2. VS COBOL II does not support multitasking with ISPF, so do not use the Message Handler sample application on both sides of a split screen. If you do, the results are unpredictable.

z/OS *Using the Message Handler sample on z/OS*

Having installed the sample and invoked it from the tailored CLIST CSQ4RCH1, the screen shown in Figure 146 on page 1240 is displayed.

```
----------------------- IBM MQ for z/OS -- Samples ------------------------
COMMAND ===>
User Id : JOHNJ



Enter information. Press ENTER :

Queue Manager Name  : ________________________________________________ :

Queue Name      : ________________________________________________ :









F1=HELP   F2=SPLIT   F3=END    F4=RETURN  F5=RFIND   F6=RCHANGE
F7=UP    F8=DOWN   F9=SWAP   F10=LEFT   F11=RIGHT  F12=RETRIEVE
```

Figure 146. Initial screen for Message Handler sample

Enter the queue manager and queue name to be viewed (case sensitive) and the message list screen is displayed (see Figure 147 on page 1241).

```
---------------------- IBM MQ for z/OS -- Samples ------- Row 1 to 4 of 4
COMMAND ==>

Queue Manager  : VM03                       :
Queue      : MQEI.IMS.BRIDGE.QUEUE              :

Message number  01 of 04

Msg  Put Date  Put Time Format   User   Put Application
No  MM/DD/YYYY HH:MM:SS Name  Identifier  Type   Name
01  10/16/1998 13:51:19 MQIMS  NTSFV02   00000002 NTSFV02A
02  10/16/1998 13:55:45 MQIMS  JOHNJ    00000011 EDIT\CLASSES\BIN\PROGTS
03  10/16/1998 13:54:01 MQIMS  NTSFV02   00000002 NTSFV02B
04  10/16/1998 13:57:22 MQIMS  johnj    00000011 EDIT\CLASSES\BIN\PROGTS
****************************** Bottom of data ********************************
```

Figure 147. Message list screen for Message Handler sample

This screen shows the first 99 messages on the queue and, for each, shows the following fields:

Msg No
Message number

Put Date MM/DD/YYYY
Date that the message was put on the queue (GMT)

Put Time HH:MM:SS
Time that the message was put on the queue (GMT)

Format Name
MQMD.Format field

User Identifier
MQMD.UserIdentifier field

Put Application Type
MQMD.PutApplType field

Put Application Name
MQMD.PutApplName field

The total number of messages on the queue is also displayed.

From this screen a message can be chosen, by number not by cursor position, and then displayed. For an example, see Figure 148 on page 1242.

```
---------------------- IBM MQ for z/OS -- Samples ----- Row 1 to 35 of 35
COMMAND ==>

Queue Manager  : VM03                        :
Queue      : MQEI.IMS.BRIDGE.QUEUE               :
Forward to Q Mgr : VM03                       :
Forward to Queue : QL.TEST.ISCRES1                 :

Action : _ :  (D)elete (F)orward

Message Content :
------------------------------------------------------------------------------
Message Descriptor
StrucId     : `MD `
Version     : 000000001
Report      : 000000000
MsgType     : 000000001
Expiry      : -00000001
Feedback     : 000000000
Encoding     : 000000785
CodedCharSetId  : 000000500
Format      : `MQIMS  `
Priority     : 000000000
Persistence    : 000000001
MsgId      : `C3E2D840E5D4F0F34040404040404040AF6B30F0A89B7605`X
CorrelId     : `00000000000000000000000000000000000000000000000000000000`X
BackoutCount    : 000000000
ReplyToQ     : `QL.TEST.ISCRES1                `
ReplyToQMgr    : `VM03                      `
UserIdentifier  : `NTSFV02   `
AccountingToken :
`06F2F5F5F3F0F1000000000000000000000000000000000000000000000000000000`X
ApplIdentityData : `                        `
PutApplType    : 000000002
PutApplName    : `NTSFV02A          `
PutDate     : `19971016`
PutTime     : `13511903`
ApplOriginData  : `   `

Message Buffer :  108 byte(s)
00000000 : C9C9 C840 0000 0001 0000 0054 0000 0311 `IIH .............`
00000010 : 0000 0000 4040 4040 4040 4040 0000 0000 `....    ....`
00000020 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000030 : 4040 4040 4040 4040 4040 4040 4040 4040 `                `
00000040 : 0000 0000 0000 0000 0000 0000 0000 0000 `................`
00000050 : 40F1 C300 0018 0000 C9C1 D7D4 C4C9 F2F8 ` 1C.....IAPMDI28`
00000060 : 40C8 C5D3 D3D6 40E6 D6D9 D3C4      ` HELLO WORLD  `
****************************** Bottom of data ********************************
```

Figure 148. Chosen message is displayed

Once the message has been displayed it can be deleted, left on the queue, or forwarded to another queue. The `Forward to Q Mgr` and `Forward to Queue` fields are initialized with values from the MQMD, these can be changed before forwarding the message.

The sample design allows only messages with unique `MsgId / CorrelId` combinations to be selected and displayed, because the message is retrieved using the `MsgId` and `CorrelId` as the key. If the key is not unique the sample cannot retrieve the chosen message with certainty.

Note: When you use the SCSQCLST(CSQ4RCH1) sample to browse messages, each invocation causes the backout count of the message to increase. If you want to change the behavior of this sample, copy the sample and modify the contents as necessary. You should be aware that other applications that rely on this backout count can be influenced by this increasing count.

z/OS *Design of the sample Message Handler sample on z/OS*

This topic describes the design of each of the programs that make up the Message Handler sample application.

Object validation program

This requests a valid queue and queue manager name.

If you do not specify a queue manager name, the default queue manager is used, if available. Only local queues can be used; an MQINQ is issued to check that the queue type and an error is reported if the queue is not local. If the queue is not opened successfully, or the MQGET call is inhibited on the queue, error messages are returned indicating the CompCode and Reason return code.

Message list program

This displays a list of messages on a queue with information about them such as the putdate, puttime, and the message format.

The maximum number of messages stored in the list is 99. If there are more messages on the queue than this, the current queue depth is also displayed. To choose a message for display, type the message number into the entry field (the default is 01). If your entry is not valid, you receive an appropriate error message.

Message content program

This displays message content.

The content is formatted and split into two parts:

1. Message descriptor
2. Message buffer

The message descriptor shows the contents of each field on a separate line.

The message buffer is formatted depending on its contents. If the buffer holds a dead letter header (MQDLH) or a transmission queue header (MQXQH), these are formatted and displayed before the buffer itself.

Before the buffer data is formatted, a title line shows the buffer length of the message in bytes. The maximum buffer size is 32768 bytes, and any message longer than this is truncated. The full size of the buffer is displayed along with a message indicating that only the first 32768 bytes of the message are displayed.

The buffer data is formatted in two ways:

1. After the offset into the buffer is printed, the buffer data is displayed in hexadecimal.
2. The buffer data is then displayed again as EBCDIC values. If any EBCDIC value cannot be printed, it prints a period (.) instead.

You can enter D for delete, or F for forward into the action field. If you choose to forward the message, the `forward-to queue` and `queue manager name` must be set correctly. The defaults for these fields are read from the message descriptor ReplyToQ and ReplyToQMgr fields.

If you forward a message, any header block stored in the buffer is stripped. If the message is forwarded successfully, it is removed from the original queue. If you enter invalid actions, error messages are displayed.

An example help panel called CSQ4CHP9 is also available.

z/OS *The Asynchronous Put sample on z/OS*

The Asynchronous Put sample program puts messages on a queue using the asynchronous MQPUT call. The sample also retrieves status information using the MQSTAT call.

The Asynchronous Put applications use these MQI calls:

- MQCONN
- MQOPEN

- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

The sample programs are delivered in the C programming language.

The Asynchronous Put applications run in the batch environment. See Other samples for the batch applications.

This topic also provides information about the design of the Asynchronous Consumption program, and running the CSQ4BCS2 sample.

- "Running the CSQ4BCS2 sample" on page 1244
- "Design of the Asynchronous Put sample program" on page 1244

Running the CSQ4BCS2 sample

This sample program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

If a queue manager is not specified, CSQ4BCS2 connects to the default queue manager. Message content is provided through standard input (**SYSIN DD**).

There is a sample JCL to run the program, it resides in CSQ4BCSP.

Design of the Asynchronous Put sample program

The program uses the MQOPEN call with either the output options supplied, or with the MQOO_OUTPUT and MQOO_FAIL_IF_QUIESCING options, to open the target queue for putting messages.

If the program cannot open the queue, the program outputs an error message containing the reason code returned by the MQOPEN call. To keep the program simple on this and subsequent MQI calls, default values are used for many of the options.

For each line of input, the program reads the text into a buffer and uses the MQPUT call with MQPMO_ASYNC_RESPONSE to create a datagram message containing the text of that line and asynchronously puts the message on the target queue. The program continues until it reaches the end of the input, or until the MQPUT call fails. If the program reaches the end of the input, it closes the queue using the MQCLOSE call.

The program then issues the MQSTAT call which returns an MQSTS structure, and displays messages containing the number of messages put successfully, the number of messages put with a warning, and the number of failures.

Note: To observe what happens when an MQPUT error is detected by the MQSTAT call, set MAXDEPTH on the target queue to a low value.

z/OS *The Batch Asynchronous Consumption sample on z/OS*

The CSQ4BCS1 sample program is delivered in C, it demonstrates the use of MQCB and MQCTL to consume messages from multiple queues asynchronously.

The Asynchronous Consumption samples run in the batch environment. See Other samples for the batch applications.

There is also a COBOL sample which runs in the CICS environment, see “The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS” on page 1246.

The applications use these MQI calls:

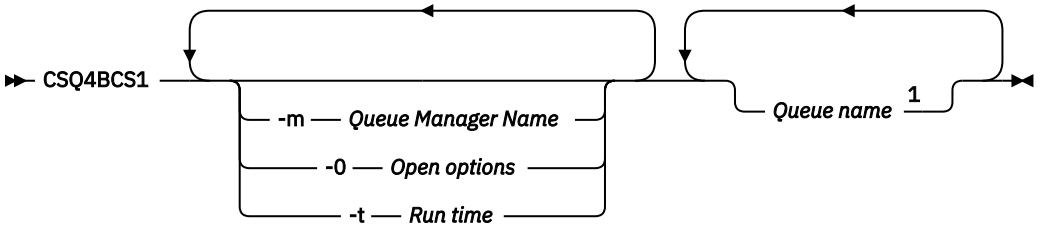
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- MQCTL

This topic also provided information about the following headings:

- “Running the CSQ4BCS1 sample” on page 1245
- “Design of the Batch Asynchronous Consumption sample program” on page 1245

Running the CSQ4BCS1 sample

This sample program follows the following syntax:



Notes:

[1] A maximum of 10 queues are supported by this sample. Provide more than one queue name in order to read messages from multiple queues.

There is a sample JCL to run this program, it resides in CSQ4BCSC.

Design of the Batch Asynchronous Consumption sample program

The sample shows how to read messages from multiple queues in the order of their arrival. This would require more code using synchronous MQGET. With asynchronous consumption, no polling is required, and thread and storage management is performed by IBM MQ. In the sample program, errors are written to the console.

The sample code has the following steps:

1. Define the single message consumption callback function.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Connect to the queue manager.

```
MQCONN(QMName,&Hcon,&CompCode,&CReason);
```

3. Open the input queues, and associate each queue with the `MessageConsumer` callback function.

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` does not need to be set for each queue; it is an input-only field. You can associate a different callback function with each queue.
4. Start consumption of the messages.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Wait for the user to press Enter, then stop consumption of messages.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Finally, disconnect from the queue manager.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS*

The Asynchronous Consumption and Publish/Subscribe sample programs demonstrate the use of asynchronous consumption, and publish and subscribe features within CICS.

A *Registration client* program registers three Callback handlers (an event handler, and two message consumers), and starts Asynchronous Consumption. A *Messaging client* program puts messages to a queue, or publishes suitable messages from a CICS console for consumption by the two Message Consumers (CSQ4CVCN and CSQ4CVCT).

To provide runtime control over the behavior of the sample, one of the message consumers can be instructed using the messages it receives, to SUSPEND, RESUME, or DEREGISTER any of the Callback handlers. It can also be used to issue an MQCTL STOP to end Asynchronous Consumption under control. The other message consumer is registered to subscribe to a topic.

Each program issues COBOL DISPLAY statements at appropriate points to display the behavior of the sample.

The applications use these MQI calls:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- MQCTL

The programs are delivered in the COBOL language. See CICS Asynchronous Consumption and Publish/Subscribe samples for the CICS applications.

This topic also provides the following information:

- “Setup” on page 1247
- “Registration Client CSQ4CVRG” on page 1247
- “Event handler CSQ4CVEV” on page 1247
- “Simple Message Consumer CSQ4CVCN” on page 1247

Setup

The names of the Queue and Topic used by the Message Consumers are hardcoded in the Registration and Messaging Client programs.

The Queue, **SAMPLE.CONTROL.QUEUE**, should be defined to the Queue Manager associated with the CICS region before running the sample. The Topic, **News/Media/Movies**, can be defined if required, or it is created at runtime under the default Administrative Object if it does not exist.

CICS programs and transaction definitions can be installed by installing a group: CSQ4SAMP.

Registration Client CSQ4CVRG

The Registration Client program must be started under the CICS transaction MVRG. It takes no input.

When started, the Registration Client registers the following Callback handlers using MQCB:

- CSQ4CVEV as an Event Handler.
- CSQ4CVCN as a Message Consumer on a topic, **News/Media/Movies**.
- CSQ4CVCT as a Message Consumer on a Queue, **SAMPLE.CONTROL.QUEUE**.

The Registration Client passes a data structure containing the names of all three registered Callback handlers to CSQ4CVCT, together with the object handles associated with the two message consumers.

Having registered the Callback handlers, the Registration Client issues an MQCTL START_WAIT to start Asynchronous Consumption, and suspend until control is returned to it (for example, by one of the Callback handlers issuing an MQCTL STOP).

Event handler CSQ4CVEV

When driven, the Event Handler displays a message indicating the call type (for example, START). When driven for IBM MQ reason code CONNECTION_QUIESCING, the Event Handler issues an MQCTL STOP to end Asynchronous Consumption and return control to the Registration Client.

Simple Message Consumer CSQ4CVCN

When driven, this Message Consumer displays a message indicating the call type (for example, REGISTER). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and outputs it to the CICS job log.

Control Message Consumer CSQ4CVCT

When driven, this Message Consumer displays a message indicating the call type (for example, START). When driven for the MSG_REMOVED call type, the Message Consumer retrieves the inbound message and the data structure passed by the Registration Client. Based on the message content, it issues appropriate MQCB or MQCTL commands to one of the following:

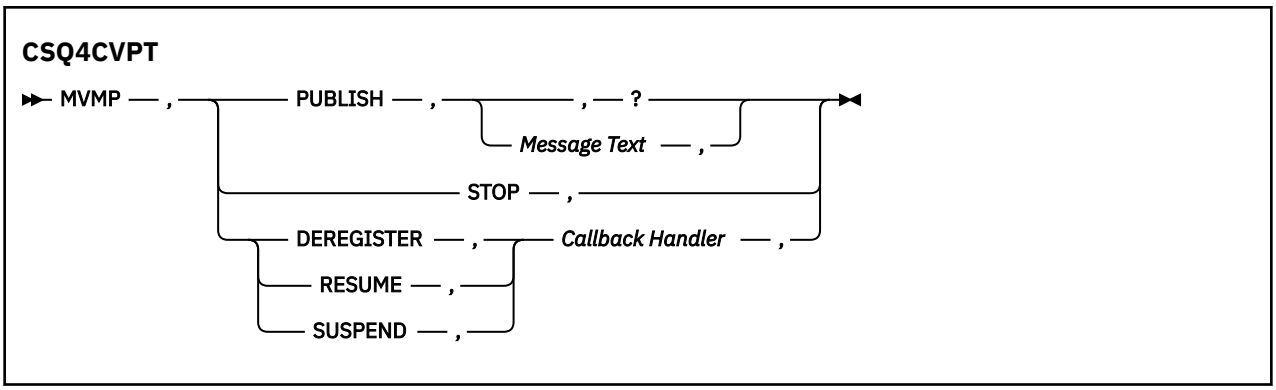
- STOP Asynchronous Consumption (returning control to the Registration Client).
- SUSPEND, RESUME, or DEREGISTER a named Callback handler (including itself).

Messaging Client CSQ4CVPT

The Messaging Client has two functions:

- It publishes a message to a topic for consumption by the Message Consumer CSQ4CVCN.
- It puts a control message to a queue for consumption by the Control Message Consumer CSQ4CVCT, resulting in a potential change in behavior of the sample.

The Messaging Client program must be started from a CICS console under a CICS transaction, and it takes command line input with the following syntax:



PUBLISH
Publish the Message Text (or a default message) as a Retained Message for consumption by the Simple Message Consumer.

STOP
Stop Asynchronous Consumption.

DEREGISTER
Deregister the named Callback handler.

RESUME
Resume the named Callback handler.

SUSPEND
Suspend the named Callback handler.

Input fields are positional, and comma-separated. Keywords and Callback Handler names are not case-sensitive.

Examples:

Table 186. Input examples

Example	Description
MVMP,PUBLISH,,	Publish a default message
MVMP,publish, `A short message`,	Publish the given text
MVMP,STOP,	Stop Asynchronous Consumption
MVMP,DEREGISTER,CSQ4CVEV,	Deregister the Event Handler
MVMP,resume,csq4cvcn,	Resume the Simple Message Consumer
MVMP,SUSPEND,CSQ4CVEV,	Suspend the Event Handler

Where MVMP is the CICS transaction associated with the Messaging Client program CSQ4CVPT.

Note:

- Suspending or deregistering all Callback handlers terminates the START_WAIT issued by the Registration Client, returning control to it, and ending the task.
- Suspending or deregistering the Control Callback Handler has deliberately not been prevented, but it removes the ability to further control the behavior of the sample.

z/OS *The Publish/Subscribe sample on z/OS*

The Publish/Subscribe sample programs demonstrate the use of the publish and subscribe features in IBM MQ.

There are four C and two COBOL programming language sample programs demonstrating how to program to the IBM MQ Publish/Subscribe interface. The programs are delivered in the C and COBOL language. The applications run in the batch environment; see Publish/Subscribe samples for the batch applications.

There are also COBOL samples that run in the CICS environment; see "The CICS Asynchronous Consumption and Publish/Subscribe sample on z/OS" on page 1246.

This topic also provides information about how to run Publish/Subscribe sample programs. These sample programs include:

Running the CSQ4BCP1 sample

This program is written in C; it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

If a queue manager is not specified, CSQ4BCP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPP.

Message content is provided through standard input (**SYSIN DD**).

Running the CSQ4BCP2 sample

This program is written in C; it subscribes to a topic and prints the messages received.

This program takes up to three parameters:

1. The name of the target topic string (required).
2. The name of the queue manager (optional).
3. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP2 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPS.

Running the CSQ4BCP3 sample

This program is written in C; it subscribes to a topic using a user-specified destination queue and prints the messages received.

This program takes up to four parameters:

1. The name of the target topic string (required).
2. The name of the destination (required).
3. The name of the queue manager (optional).
4. MQSD subscription options (optional).

If a queue manager is not specified, CSQ4BCP3 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPD.

Running the CSQ4BCP4 sample

This program is written in C; it subscribes and gets messages from a topic allowing the use of extended options on the MQSUB call, extending those available on the simpler MQSUB sample: CSQ4BCP2. In addition to the message payload, message properties for each message are received and displayed.

This program takes a variable set of parameters:

- **-t** *Topic string*.
- **-o** *Topic object name*.

  Important: One of **-t** or **-o**, or both, is required
- **-m** *Queue manager name* (optional).
- **-b** *Connection binding type* (optional), where *type* can have any of the following values:
  - *standard* : MQCNO_STANDARD_BINDING , which is the default value
  - *shared*: MQCNO_SHARED_BINDING
  - *fastpath*: MQCNO_FASTPATH_BINDING
  - *isolated*: MQCNO_ISOLATED_BINDING
- **-q** *Destination queue name* (optional).
- **-w** *Wait interval on MQGET in seconds* (optional), where *seconds* can have any of the following values:
  - unlimited: MQWI_UNLIMITED
  - none: No wait
  - *n*: Wait interval in seconds
  - No value specified: When no value is specified, the default is 30 seconds
- **-d** *Subscription name* (optional). Creates or resumes named durable subscription.
- **-k** (optional). Keeps durable subscription on MQCLOSE.

If a queue manager is not specified, CSQ4BCP4 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BCPE.

Running the CSQ4BVP1 sample

This program is written in COBOL, it publishes messages to a topic. Start one of the subscriber samples before running this program.

This program takes no parameters. **SYSIN DD** provides the input topic name, queue manager name, and message content.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

Running the CSQ4BVP2 sample

This program is written in COBOL, it subscribes to a topic and prints the messages received.

This program takes no parameters. **SYSIN DD** provides the input for topic name and queue manager name.

If a queue manager is not specified, CSQ4BVP1 connects to the default queue manager. There is a sample JCL to run the program, it resides in CSQ4BVPP.

z/OS *The Set and Inquire message property sample on z/OS*

The message property sample programs demonstrate the addition of user-defined properties to a message handle, and the inquisition of the properties associated with that message.

The applications use these MQI calls:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

The programs are delivered in the C language. The applications run in the batch environment. See Other samples for the batch applications.

The CSQ4BCM1 program is used to inquire the properties of a message handle from a message queue, and it is an example of the use of the MQINQMP API call. The sample gets one message from a queue and then prints all the message handle properties.

The CSQ4BCM2 program is used to set the properties of a message handle on a message queue, and it is an example of the use of the MQSETMP API call. The sample creates a message handle and puts it into the `MsgHandle` field of the MQGMO structure. It then puts the message to a queue.

Other examples of inquiring and printing message properties are included in the CSQ4BCG1 and CSQ4BCP4 sample programs.

This topic also provides information on running the Set and Inquire message property samples under the following headings:

- "Running the CSQ4BCM1 sample" on page 1251
- "Running the CSQ4BCM2 sample" on page 1251

Running the CSQ4BCM1 sample

This program takes up to four parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).

Running the CSQ4BCM2 sample

This program takes up to six parameters:

1. The name of the target queue (required).
2. The name of the queue manager (optional).
3. Open options (optional).
4. Close options (optional).
5. The name of the target queue manager (optional).
6. The name of the dynamic queue (optional).

The property names, values, and message content are provided through the standard input (**SYSIN DD**). There is a sample JCL to run the program, it resides in CSQ4BCMP.

Développement d'applications pour Managed File Transfer

Spécifiez les programmes à exécuter avec Managed File Transfer, utilisez Apache Ant avec Managed File Transfer, personnalisez Managed File Transfer avec des exits utilisateur et contrôlez Managed File Transfer en plaçant les messages dans la file d'attente de commandes de l'agent.

Spécification des programmes à exécuter avec MFT

Vous pouvez exécuter des programmes sur un système sur lequel un Managed File Transfer Agent est en cours d'exécution. Dans le cadre d'une demande de transfert de fichiers, vous pouvez spécifier un programme à exécuter avant ou après le début d'un transfert. En outre, vous pouvez démarrer un programme qui ne fait pas partie d'une demande de transfert de fichier en soumettant une demande d'appel géré.

Pourquoi et quand exécuter cette tâche

Il existe cinq scénarios dans lesquels vous pouvez spécifier un programme à exécuter:

- Dans le cadre d'une demande de transfert, au niveau de l'agent source, avant le début du transfert.
- Dans le cadre d'une demande de transfert, au niveau de l'agent de destination, avant le début du transfert.
- Dans le cadre d'une demande de transfert, au niveau de l'agent source, une fois le transfert terminé.
- Dans le cadre d'une demande de transfert, au niveau de l'agent de destination, une fois le transfert terminé.
- Pas dans le cadre d'une demande de transfert. Vous pouvez soumettre une demande à un agent pour exécuter un programme. Ce scénario est parfois appelé appel géré.

Les exits utilisateur et les appels de programme sont appelés dans l'ordre suivant:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Remarques :

1. Le **DestinationTransferEndExits** est exécuté uniquement lorsque le transfert est terminé, soit avec succès, soit partiellement.
2. Le **postDestinationCall** est exécuté uniquement lorsque le transfert est terminé, soit avec succès, soit partiellement.
3. Le **SourceTransferEndExits** est exécuté pour les transferts réussis, partiellement réussis ou ayant échoué.
4. **postSourceCall** est appelé uniquement si:
   - Le transfert n'a pas été annulé.
   - Le résultat est positif ou partiellement positif.
   - Tous les programmes de transfert post-destination ont été exécutés avec succès.

Procédure

- Indiquez le programme à exécuter à l'aide de l'une des options suivantes:

Utiliser une tâche Apache Ant

Utilisez l'une des tâches `fte:filecopy`, `fte:filemoveet` `fte:call` Ant pour démarrer un programme. A l'aide d'une tâche Ant , vous pouvez spécifier un programme dans l'un des cinq scénarios, à l'aide des éléments imbriqués `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrcet` `fte:command` . Pour plus d'informations, voir Eléments imbriqués d'appel de programme.

Editer le message de demande de transfert de fichier

Vous pouvez éditer le code XML généré par une demande de transfert. Avec cette méthode, vous pouvez exécuter un programme dans l'un des cinq scénarios, en ajoutant des éléments **preSourceCall**, **postSourceCall**, **preDestinationCall**, **postDestinationCall**et **managedCall** au fichier XML. Utilisez ensuite ce fichier XML modifié comme définition de transfert pour une nouvelle demande de transfert de fichier, par exemple avec le paramètre **fteCreateTransfer -td** . Pour plus d'informations, voir Exemples de message de demande d'appel d'agent MFT.

Utilisez la commande `fteCreateTransfer`

Vous pouvez utiliser la commande **fteCreateTransfer** pour spécifier les programmes à démarrer. Vous pouvez utiliser la commande pour spécifier des programmes à exécuter dans les quatre premiers scénarios, dans le cadre d'une demande de transfert, mais vous ne pouvez pas démarrer un appel géré. Pour plus d'informations sur les paramètres à utiliser, voir **fteCreateTransfer**: Démarrer un nouveau transfert de fichier. Pour obtenir des exemples d'utilisation de cette commande, voir Exemples d'utilisation de fteCreateTransfer to start programs.

Référence associée

Propriété commandPath MFT

Appels gérés

Les agents Managed File Transfer (MFT) sont généralement utilisés pour transférer des fichiers ou des messages. Ces transferts sont appelés *transferts gérés*. Les agents peuvent également être utilisés pour exécuter des commandes, des scripts ou des JCL sans qu'il soit nécessaire de transférer des fichiers ou des messages. Cette fonction est appelée *Appels gérés*.

Les demandes d'appels gérés peuvent être soumises à un agent de plusieurs manières:

- A l'aide de la tâche fte: call Ant.
- Configuration d'un moniteur de ressources avec une tâche XML qui exécute une commande ou un script. Pour plus d'informations, voir Configuration des tâches de surveillance pour lancer des commandes et des scripts .
- Insertion directe d'un message XML dans la file d'attente de commandes de l'agent. Voir Format de message de demande de transfert de fichier pour plus de détails sur le schéma XML d'appel géré.

Pour les appels gérés, le répertoire contenant la commande ou le script en cours d'exécution doit être spécifié dans la propriété d'agent **commandPath**.

Les appels gérés ne peuvent pas exécuter des commandes ou des scripts qui se trouvent dans des répertoires qui ne sont pas spécifiés dans le fichier **commandPath**de l'agent. Cela permet de s'assurer que l'agent n'exécute pas de code malveillant.

Important : Pour vous assurer que c'est le cas, par défaut, lorsque vous spécifiez **commandPath**:

- Tout bac à sable d'agent existant est configuré par l'agent au démarrage afin que tous les répertoires **commandPath** soient automatiquement ajoutés à la liste des répertoires auxquels l'accès est refusé pour un transfert.
- Tous les bacs à sable d'utilisateur existants sont mis à jour au démarrage de l'agent de sorte que tous les répertoires **commandPath** (et leurs sous-répertoires) soient ajoutés en tant qu'éléments `<exclude>` aux éléments `<read>` et `<write>` .

- Si l'agent n'est pas configuré pour utiliser un bac à sable d'agent ou des bacs à sable d'utilisateur, un nouveau bac à sable d'agent est créé au démarrage de l'agent avec les répertoires **commandPath** spécifiés comme étant des répertoires refusés.

En outre, vous pouvez également activer la vérification des droits d'accès sur un agent pour vous assurer que seuls les utilisateurs autorisés sont autorisés à soumettre des demandes d'appels gérés. Pour plus d'informations à ce sujet, voir Restriction des droits utilisateur sur les actions de l'agent MFT.

La commande, le script ou le JCL appelé dans le cadre d'un appel géré s'exécute en tant que processus externe, surveillé par l'agent. Lorsque le processus se termine, l'appel géré se termine et le code retour du processus est mis à la disposition de l'agent ou du script Ant qui a appelé la tâche **fte:call** Ant .

Si l'appel géré a été démarré par la tâche **fte:call** Ant , votre script Ant peut vérifier la valeur du code retour pour déterminer si l'appel géré a abouti ou non.

Pour tous les autres types d'appels gérés, vous pouvez spécifier les valeurs de code retour à utiliser pour indiquer que l'appel géré a abouti. L'agent compare le code retour du processus à ces codes retour lorsque le processus externe se termine.

Remarque : Etant donné que les appels gérés s'exécutent en tant que processus externes, ils ne peuvent pas être annulés une fois qu'ils ont démarré.

Appels gérés et emplacements de transfert source

Un agent contient un certain nombre d'emplacements de transfert source, comme indiqué par la propriété d'agent **maxSourceTransfers**, décrite dans Propriétés d'agent avancées: Limite de transfert.

Chaque fois qu'un appel géré ou un transfert géré est exécuté, ils occupent un emplacement de transfert source. L'emplacement est libéré à la fin de l'appel géré ou du transfert géré.

Si tous les emplacements de transfert source sont utilisés lorsqu'un agent reçoit un nouvel appel géré ou une demande de transfert géré, la demande est mise en file d'attente par l'agent jusqu'à ce qu'un emplacement soit disponible.

Si un appel géré démarre un transfert géré (par exemple, si un appel géré exécute un script Ant et que le script Ant utilise la tâche fte: filecopy ou fte: filemove pour transférer un fichier), deux emplacements de transfert source sont requis:

- Un pour le transfert géré
- Un pour l'appel géré

Dans cette situation, il est important de noter que si le transfert géré prend beaucoup de temps à se terminer ou est en cours de récupération, les deux emplacements de transfert source sont occupés jusqu'à ce que le transfert géré soit terminé, soit annulé ou qu'il arrive à expiration en raison d'une **transferRecoveryTimeout**. Voir Transfer recovery timeout concepts pour plus de détails sur **transferRecoveryTimeout**. Cela peut potentiellement limiter le nombre d'autres transferts gérés ou d'appels gérés que l'agent peut traiter.

Par conséquent, vous devez envisager la conception d'un appel géré pour vous assurer qu'il n'occupe pas les créneaux de transfert source pendant une longue période.

Utilisation de REST API avec des appels gérés

Les instructions HTTP GET et HTTP POST sont prises en charge pour l'activation des appels gérés et ne fonctionnent que sur la version 3 de REST API.

D'autres instructions, par exemple HTTP DELETE et HTTP UPDATE, ne sont pas prises en charge et renvoient le code d'erreur HTTP 405 si vous tentez de les utiliser.

Avertissement : Une fois soumis, un appel géré ne peut pas être annulé à l'aide de REST API.

Utilisation de Apache Ant avec MFT

Managed File Transfer fournit des tâches que vous pouvez utiliser pour intégrer la fonction de transfert de fichiers dans l'outil Apache Ant .

Vous pouvez utiliser la commande **fteAnt** pour exécuter des tâches Ant dans un environnement Managed File Transfer que vous avez déjà configuré. Vous pouvez utiliser les tâches Ant de transfert de fichier à partir de vos scripts Ant pour coordonner des opérations de transfert de fichier complexes à partir d'un langage de script interprété.

Pour plus d'informations sur Apache Ant, voir la page Web du projet Apache Ant : https://ant.apache.org/

Concepts associés

«Initiation à l'utilisation des scripts Ant avec MFT», à la page 1255
L'utilisation de scripts Ant avec Managed File Transfer vous permet de coordonner des opérations de transfert de fichiers complexes à partir d'un langage de script interprété.

fteAnt: exécuter les tâches Ant dans MFT

Référence associée

«Exemples de tâches Ant pour MFT», à la page 1256
Un certain nombre d'exemples de script Ant sont fournis avec votre installation de Managed File Transfer. Ces exemples se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Chaque exemple de script contient une cible `init` , éditez les propriétés définies dans la cible `init` pour exécuter ces scripts avec votre configuration.

Initiation à l'utilisation des scripts Ant avec MFT

L'utilisation de scripts Ant avec Managed File Transfer vous permet de coordonner des opérations de transfert de fichiers complexes à partir d'un langage de script interprété.

Scripts Ant

Les scripts Ant (ou fichiers de génération) sont des documents XML qui définissent une ou plusieurs cibles. Ces cibles contiennent des éléments de tâche à exécuter. Managed File Transfer fournit des tâches que vous pouvez utiliser pour intégrer la fonction de transfert de fichiers dans Apache Ant. Pour en savoir plus sur les scripts Ant , consultez la page Web du projet Apache Ant : https://ant.apache.org/

Des exemples de scripts Ant qui utilisent des tâches Managed File Transfer sont fournis avec l'installation de votre produit dans le répertoire *MQ_INSTALLATION_PATH*/mqft/samples/fteant

Sur les agents de pont de protocole, les scripts Ant sont exécutés sur le système d'agent de pont de protocole. Ces scripts Ant n'ont pas d'accès direct aux fichiers sur le serveur FTP ou SFTP.

Espace de nom

Un espace de nom est utilisé pour différencier les tâches Ant de transfert de fichier des autres tâches Ant qui peuvent partager le même nom. Vous définissez l'espace de nom dans la balise de projet de votre script Ant .

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs" default="do_ping">

  <target name="do_ping">
     <fte:ping cmdqm="qm@localhost@1414@SYSTEM.DEF.SVRCONN" agent="agent1@qm1"
      rcproperty="ping.rc" timeout="15"/>
  </target>

</project>
```

L'attribut `xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs"` indique à Ant de rechercher les définitions des tâches préfixées par `fte` dans la bibliothèque `com.ibm.wmqfte.ant.taskdefs`.

Vous n'avez pas besoin d'utiliser `fte` comme préfixe d'espace de nom ; vous pouvez utiliser n'importe quelle valeur. Le préfixe d'espace de nom `fte` est utilisé dans tous les exemples et exemples de script Ant .

Exécution de scripts Ant

Pour exécuter les scripts Ant qui contiennent les tâches Ant de transfert de fichier, utilisez la commande **fteAnt** . Exemple :

```
fteAnt -file ant_script_location/ant_script_name
```

Pour plus d'informations, voir **fteAnt**: exécutez les tâches Ant dans MFT.

Codes retour

Les tâches Ant de transfert de fichier renvoient les mêmes codes retour que les commandes Managed File Transfer . Pour plus d'informations, voir Codes retour pour MFT.

Référence associée

fteAnt: exécuter les tâches Ant dans MFT

«Exemples de tâches Ant pour MFT», à la page 1256

Un certain nombre d'exemples de script Ant sont fournis avec votre installation de Managed File Transfer. Ces exemples se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Chaque exemple de script contient une cible `init` , éditez les propriétés définies dans la cible `init` pour exécuter ces scripts avec votre configuration.

Exemples de tâches Ant pour MFT

Un certain nombre d'exemples de script Ant sont fournis avec votre installation de Managed File Transfer. Ces exemples se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Chaque exemple de script contient une cible `init` , éditez les propriétés définies dans la cible `init` pour exécuter ces scripts avec votre configuration.

e-mail

L'exemple email montre comment utiliser les tâches Ant pour transférer un fichier et envoyer un courrier électronique à une adresse électronique spécifiée en cas d'échec du transfert. Le script vérifie que les agents source et de destination sont actifs et qu'ils peuvent traiter les transferts à l'aide de la tâche Managed File Transfer ping . Si les deux agents sont actifs, le script utilise la tâche Managed File Transfer fte: filecopy pour transférer un fichier entre les agents source et cible, sans supprimer le fichier d'origine. Si le transfert échoue, le script envoie un courrier électronique contenant des informations sur l'échec à l'aide de la tâche standard Ant email .

de sécurité des données

L'exemple hub est constitué de deux scripts: `hubcopy.xml` et `hubprocess.xml.` Le script `hubcopy.xml` montre comment utiliser le scriptage Ant pour générer des topologies de style'hub and spoke'. Dans cet exemple, deux fichiers sont transférés des agents s'exécutant sur des machines satellites vers un agent s'exécutant sur la machine concentrateur. Les deux fichiers sont transférés en même temps et lorsque les transferts sont terminés, le script `hubprocess.xml` Ant est exécuté sur la machine du concentrateur pour traiter les fichiers. Si les deux fichiers sont transférés correctement, le script Ant concatène le contenu des fichiers. Si les fichiers ne sont pas transférés correctement, le script Ant nettoie en supprimant les données de fichier qui ont été transférées. Pour que cet exemple fonctionne correctement, vous devez placer le script `hubprocess.xml` sur le chemin de commande de l'agent concentrateur. Pour plus d'informations sur la définition du chemin de commande d'un agent, voir commandPath MFT property.

librarytransfer (plateformeIBM i uniquement)

IBM i

IBM i L'exemple librarytransfer montre comment utiliser les tâches Ant pour transférer une bibliothèque IBM i sur un système IBM i vers un deuxième système IBM i .

IBM i L'exemple librarytransfer utilise la prise en charge des fichiers de sauvegarde natifs sous IBM i avec des tâches Ant prédéfinies disponibles dans Managed File Transfer pour transférer des objets de bibliothèque natifs entre deux systèmes IBM i . L'exemple utilise un élément imbriqué < presrc> dans une tâche Managed File Transfer filecopy pour appeler un script exécutable `librarysave.sh` qui sauvegarde la bibliothèque demandée sur le système de l'agent source dans un fichier de sauvegarde temporaire. Le fichier sauvegarde est déplacé par la tâche Ant filecopy vers le système d'agent cible où un élément imbriqué < postdst> est utilisé pour appeler le script exécutable `libraryrestore.sh` afin de restaurer la bibliothèque sauvegardée dans le fichier sauvegarde sur le système cible.

IBM i Avant d'exécuter cet exemple, vous devez effectuer une configuration comme décrit dans le fichier `librarytransfer.xml` . Vous devez également disposer d'un environnement Managed File Transfer opérationnel sur deux machines IBM i . La configuration doit se composer d'un agent source s'exécutant sur la première machine IBM i et d'un agent cible s'exécutant sur la deuxième machine IBM i . Les deux agents doivent pouvoir communiquer entre eux.

IBM i L'exemple librarytransfer comprend les trois fichiers suivants:

- `librarytransfer.xml`
- `librarysave.sh` (script exécutable < presrc>)
- `libraryrestore.sh` (script exécutable < postdst>)

Les exemples de fichiers se trouvent dans le répertoire suivant: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Pour exécuter cet exemple, l'utilisateur doit effectuer les étapes suivantes:

1. Démarrez une session Qshell. Dans une fenêtre de commande IBM i , entrez: STRQSH
2. Accédez au répertoire `bin` comme suit:

   ```
   cd  /QIBM/ProdData/WMQFTE/V7/bin
   ```

3. Une fois la configuration requise terminée, exécutez l'exemple à l'aide de la commande suivante:

   ```
   fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
   ```

physicalfiletransfer (plateformeIBM i uniquement)

IBM i L'exemple physicalfiletransfer montre comment utiliser les tâches Ant pour transférer un fichier source physique ou de base de données d'une bibliothèque sur un système IBM i vers une bibliothèque sur un deuxième système IBM i .

IBM i L'exemple physicalfiletransfer utilise la prise en charge du fichier de sauvegarde natif sous IBM i avec des tâches Ant prédéfinies disponibles dans Managed File Transfer pour transférer des fichiers source physiques et de base de données complets entre deux systèmes IBM i . L'exemple utilise un élément imbriqué < presrc> dans une tâche Managed File Transfer filecopy pour appeler un script exécutable `physicalfilesave.sh` afin de sauvegarder le fichier source physique ou de base de données demandé à partir d'une bibliothèque du système de l'agent source dans un fichier de sauvegarde temporaire. Le fichier sauvegarde est déplacé par la tâche Ant filecopy vers le système de l'agent cible où un élément imbriqué < postdst> est utilisé pour appeler le script exécutable `physicalfilerestore.sh` , puis restaure l'objet fichier dans le fichier sauvegarde dans une bibliothèque spécifiée sur le système cible.

IBM i Avant d'exécuter cet exemple, vous devez effectuer une configuration comme décrit dans le fichier `physicalfiletransfer.xml` . Vous devez également disposer d'un environnement Managed File Transfer opérationnel sur deux systèmes IBM i . La configuration doit se composer d'un agent source s'exécutant sur le premier système IBM i et d'un agent cible s'exécutant sur le second système IBM i . Les deux agents doivent pouvoir communiquer entre eux.

IBM i L'exemple physicalfiletransfer comprend les trois fichiers suivants:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (script exécutable < presrc>)
- `physicalfilerestore.sh` (script exécutable < postdst>)

Les exemples de fichiers se trouvent dans le répertoire suivant: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Pour exécuter cet exemple, l'utilisateur doit effectuer les étapes suivantes:

1. Démarrez une session Qshell. Dans une fenêtre de commande IBM i , entrez: STRQSH
2. Accédez au répertoire `bin` comme suit:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Une fois la configuration requise terminée, exécutez l'exemple à l'aide de la commande suivante:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

délai d'attente

L'exemple timeout montre comment utiliser des tâches Ant pour tenter un transfert de fichier et annuler le transfert s'il dure plus longtemps qu'une valeur de délai d'attente spécifiée. Le script lance un transfert de fichier à l'aide de la tâche Managed File Transfer fte: filecopy . Le résultat de ce transfert est différé. Le script utilise la Managed File Transfer tâche fte: awaitissue Ant pour attendre un nombre de secondes donné avant que le transfert ne se termine. Si le transfert n'est pas terminé dans le délai imparti, la Managed File Transfer tâche fte: cancel Ant est utilisée pour annuler le transfert de fichier.

vsamtransfer

z/OS

z/OS L'exemple vsamtransfer montre comment utiliser les tâches Ant pour effectuer un transfert d'un fichier VSAM vers un autre fichier VSAM à l'aide de Managed File Transfer. Actuellement, Managed File Transfer ne prend pas en charge le transfert de fichiers VSAM. L'exemple de script décharge les enregistrements de données VSAM dans un fichier séquentiel à l'aide des `presrc` éléments imbriqués d'appel de programme ../refdev/nested_params.dita pour appeler le fichier exécutable `datasetcopy.sh`. Le script utilise la tâche Managed File Transfer fte: filemove pour transférer le jeu de données séquentielles de l'agent source vers l'agent cible. Le script utilise ensuite les `postdst` éléments imbriqués d'appel de programme ../refdev/nested_params.dita pour appeler le script `loadvsam.jcl` . Ce script JCL charge les enregistrements de fichier transférés dans un fichier VSAM de destination. Cet exemple utilise JCL pour l'appel de destination afin de présenter cette option de langage. Le même résultat peut également être obtenu en utilisant un second script shell à la place.

z/OS Cet exemple ne nécessite pas que les fichiers source et cible soient VSAM. L'exemple fonctionne pour tous les jeux de données si les jeux de données source et cible sont du même type.

z/OS Pour que cet exemple fonctionne correctement, vous devez placer le script `datasetcopy.sh` sur le chemin de commande de l'agent source et le script `loadvsam.jcl` sur le

chemin de commande de l'agent cible. Pour plus d'informations sur la définition du chemin de commande d'un agent, voir commandPath MFT property.

zip

L'exemple zip est constitué de deux scripts: `zip.xml` et `zipfiles.xml`. L'exemple montre comment utiliser l'élément imbriqué `presrc` ../refdev/nested_params.dita dans la tâche Managed File Transfer fte: filemove pour exécuter un script Ant avant d'effectuer une opération de déplacement de transfert de fichier. Le script `zipfiles.xml` appelé par l'élément imbriqué `presrc` dans le script `zip.xml` compresse le contenu d'un répertoire. Le script `zip.xml` transfère le fichier compressé. Cet exemple nécessite que le script `zipfiles.xml` Ant soit présent dans le chemin de commande de l'agent source. En effet, le script `zipfiles.xml` Ant contient la cible utilisée pour compresser le contenu du répertoire sur l'agent source. Pour plus d'informations sur la définition du chemin de commande d'un agent, voir commandPath MFT property.

Concepts associés

«Initiation à l'utilisation des scripts Ant avec MFT», à la page 1255
L'utilisation de scripts Ant avec Managed File Transfer vous permet de coordonner des opérations de transfert de fichiers complexes à partir d'un langage de script interprété.

Référence associée

`fteAnt`: exécuter les tâches Ant dans MFT

Personnalisation de MFT avec des exits utilisateur

Vous pouvez personnaliser les fonctions d' Managed File Transfer à l'aide de vos propres programmes appelés routines d'exit utilisateur.

Important : Tout code contenu dans un exit utilisateur n'est pas pris en charge par IBMet tout problème lié à ce code doit être initialement examiné par votre entreprise ou par le fournisseur qui a fourni l'exit.

Managed File Transfer fournit des points dans le code où Managed File Transfer peut transmettre le contrôle à un programme que vous avez écrit (une routine d'exit utilisateur). Ces points sont appelés points d'exit utilisateur. Managed File Transfer peut alors reprendre le contrôle lorsque votre programme a terminé son travail. Vous n'avez pas besoin d'utiliser les exits utilisateur, mais ils sont utiles si vous souhaitez étendre et personnaliser la fonction de votre système Managed File Transfer pour répondre à vos besoins spécifiques.

Il existe deux points au cours du traitement du transfert de fichiers où vous pouvez appeler un exit utilisateur sur le système source et deux points au cours du traitement du transfert de fichiers où vous pouvez appeler un exit utilisateur sur le système cible. Le tableau suivant récapitule chacun de ces points d'exit utilisateur et l'interface Java que vous devez implémenter pour utiliser les points d'exit.

Tableau 187. Récapitulatif des points d'exit côté source et côté destination et des interfaces Java

Point de sortie	**Interface Java à implémenter**
Points d'exit côté source:	
Avant le début du transfert de fichiers complet	SourceTransferStartExit.java
Une fois le transfert de fichiers complet terminé	SourceTransferEndExitinterface .java
Points d'exit côté destination:	
Avant le début du transfert de fichiers complet	DestinationTransferStartExit.java
Une fois le transfert de fichiers complet terminé	DestinationTransferEndExitinterface .java

Les exits utilisateur sont appelés dans l'ordre suivant:

1. SourceTransferStartExit
2. DestinationTransferStartExit

3. DestinationTransferEndExit
4. SourceTransferEndExit

Les modifications apportées par les exits SourceTransferStartExit et DestinationTransferStartExit sont propagées en tant qu'entrées aux exits suivants. Par exemple, si l'exit SourceTransferStartExit modifie les métadonnées de transfert, les modifications sont reflétées dans les métadonnées de transfert d'entrée vers les autres exits.

Les exits utilisateur et les appels de programme sont appelés dans l'ordre suivant:

```
- SourceTransferStartExit(onSourceTransferStart).
- PRE_SOURCE Command.
- DestinationTransferStartExits(onDestinationTransferStart).
- PRE_DESTINATION Command.
- The Transfer request is performed.
- DestinationTransferEndExits(onDestinationTransferENd).
- POST_DESTINATION Command.
- SourceTransferEndExits(onSourceTransferEnd.
- POST_SOURCE Command.
```

Remarques :

1. Le **DestinationTransferEndExits** est exécuté uniquement lorsque le transfert est terminé, soit avec succès, soit partiellement.
2. Le **postDestinationCall** est exécuté uniquement lorsque le transfert est terminé, soit avec succès, soit partiellement.
3. Le **SourceTransferEndExits** est exécuté pour les transferts réussis, partiellement réussis ou ayant échoué.
4. **postSourceCall** est appelé uniquement si:
   - Le transfert n'a pas été annulé.
   - Le résultat est positif ou partiellement positif.
   - Tous les programmes de transfert post-destination ont été exécutés avec succès.

Génération de votre exit utilisateur

Les interfaces permettant de générer un exit utilisateur sont contenues dans *MQ_INSTALL_DIRECTORY*/`mqft/lib/com.ibm.wmqfte.exitroutines.api.jar`. Vous devez inclure ce fichier .jar dans le chemin d'accès aux classes lorsque vous générez votre exit. Pour exécuter l'exit, extrayez l'exit en tant que fichier .jar et placez ce fichier .jar dans un répertoire, comme décrit dans la section suivante.

Emplacements d'exit utilisateur

Vous pouvez stocker vos routines d'exit utilisateur dans deux emplacements possibles:

- Le répertoire `exits`. Il existe un répertoire d'exits sous chaque répertoire d'agent. Par exemple : `var\mqm\mqft\config\QM_JUPITER\agents\AGENT1\exits`
- Vous pouvez définir la propriété de chemin exitClasspour spécifier un autre emplacement. S'il existe des classes d'exit dans le répertoire `exits` et dans le chemin d'accès aux classes défini par le chemin d'accès exitClass, les classes du répertoire `exits` sont prioritaires, ce qui signifie que s'il existe des classes dans les deux emplacements portant le même nom, les classes du répertoire `exits` sont prioritaires.

Configuration d'un agent pour l'utilisation des exits utilisateur

Quatre propriétés d'agent peuvent être définies pour spécifier
les exits utilisateur qu'un agent appelle. Ces propriétés
d'agent sont `sourceTransferStartExitClasses`, `sourceTransferEndExitClasses`, `destinationTransferStartExitClasseset destinationTransferEndExitClasses`. Pour plus d'informations sur l'utilisation de ces propriétés, voir MFT Propriétés de l'agent pour les exits utilisateur.

Exécution des exits utilisateur sur les agents de pont de protocole

Lorsque l'agent source appelle l'exit, il transmet à l'exit une liste des éléments source pour le transfert. Pour les agents normaux, il s'agit d'une liste de noms de fichiers complets. Comme les fichiers doivent être locaux (ou accessibles via un montage), l'exit peut y accéder et les chiffrer.

Toutefois, pour un agent de pont de protocole, les entrées de la liste sont au format suivant:

```
"<file server identifier>:<fully-qualified file name of the file on the remote file server>"
```

Pour chaque entrée de la liste, l'exit doit d'abord se connecter au serveur de fichiers (via FTP). protocoles FTPS ou SFTP), téléchargez le fichier, chiffrez-le localement, puis téléchargez le fichier chiffré sur le serveur de fichiers.

Exécution des exits utilisateur sur les agents de pont Connect:Direct

Vous ne pouvez pas exécuter d'exits utilisateur sur des agents de pont Connect:Direct .

Concepts associés

«Exits utilisateur source et cible MFT», à la page 1261

Métadonnées pour les exits utilisateur MFT

Interfaces Java pour les exits utilisateur MFT

Référence associée

«Activation du débogage à distance pour les exits utilisateur MFT», à la page 1266
Lorsque vous développez vos exits utilisateur, vous pouvez utiliser un débogueur pour localiser les problèmes dans votre code.

«Exemple d'exit utilisateur de transfert de source MFT», à la page 1267

«Exemple d'exit utilisateur de données d'identification de pont de protocole», à la page 1268

Exits utilisateur du moniteur de ressources MFT

Propriétés de l'agent MFT pour les exits utilisateur

Exits utilisateur source et cible MFT

Séparateurs de répertoire

Les séparateurs de répertoire dans les spécifications de fichier source sont toujours représentés à l'aide de barres obliques (/), quelle que soit la façon dont vous avez spécifié les séparateurs de répertoire dans la commande **fteCreateTransfer** ou dans IBM MQ Explorer. Vous devez en tenir compte lorsque vous écrivez un exit. Par exemple, si vous souhaitez vérifier que le fichier source suivant existe: `c:\a\b.txt` et que vous avez spécifié ce fichier source à l'aide de la commande **fteCreateTransfer** ou de IBM MQ Explorer, notez que le nom de fichier est stocké sous la forme suivante: `c:/a/b.txt` Donc, si vous recherchez la chaîne d'origine de `c:\a\b.txt`, vous ne trouverez pas de correspondance.

Points d'exit côté source

Avant le début du transfert de fichiers complet

Cet exit est appelé par l'agent source lorsqu'une demande de transfert est suivante dans la liste des transferts en attente et que le transfert est sur le point de démarrer.

Par exemple, vous pouvez utiliser ce point d'exit pour envoyer des fichiers par étapes dans un répertoire auquel l'agent a accès en lecture / écriture à l'aide d'une commande externe ou pour renommer les fichiers sur le système cible.

Transmettez les arguments suivants à cet exit:

- Nom de l'agent source
- Nom de l'agent de destination
- Métadonnées d'environnement

- Métadonnées de transfert
- Spécifications de fichier (y compris les métadonnées de fichier)

Les données renvoyées par cet exit sont les suivantes:

- Métadonnées de transfert mises à jour. Les entrées peuvent être ajoutées, modifiées et supprimées.
- Liste mise à jour des spécifications de fichier, qui se compose de paires de nom de fichier source et de nom de fichier cible. Les entrées peuvent être ajoutées, modifiées et supprimées
- Indicateur qui indique si le transfert doit être poursuivi
- Chaîne à insérer dans le journal de transfert.

Implémentez l'interface SourceTransferStartExit.java pour appeler le code d'exit utilisateur à ce point d'exit.

Une fois le transfert de fichiers complet terminé

Cet exit est appelé par l'agent source une fois le transfert de fichier complet terminé.

Un exemple d'utilisation de ce point d'exit consiste à effectuer des tâches d'achèvement, telles que l'envoi d'un courrier électronique ou d'un message IBM MQ pour indiquer que le transfert est terminé.

Transmettez les arguments suivants à cet exit:

- Résultat de l'exit de transfert
- Nom de l'agent source
- Nom de l'agent de destination
- Métadonnées d'environnement
- Métadonnées de transfert
- Résultats du fichier

Les données renvoyées par cet exit sont les suivantes:

- Chaîne mise à jour à insérer dans le journal de transfert.

Implémentez l' interfaceSourceTransferEndExit.java pour appeler le code d'exit utilisateur à ce point d'exit.

Points d'exit côté destination

Avant le début du transfert de fichiers complet

Un exemple d'utilisation de ce point d'exit consiste à valider les droits d'accès à la destination.

Transmettez les arguments suivants à cet exit:

- Nom de l'agent source
- Nom de l'agent de destination
- Métadonnées d'environnement
- Métadonnées de transfert
- Spécifications de fichier

Les données renvoyées par cet exit sont les suivantes:

- Ensemble de noms de fichiers de destination mis à jour. Les entrées peuvent être modifiées mais ne peuvent pas être ajoutées ou supprimées.
- Indicateur qui indique si le transfert doit être poursuivi
- Chaîne à insérer dans le journal de transfert.

Implémentez l'interface DestinationTransferStartExit.java pour appeler le code d'exit utilisateur à ce point d'exit.

Une fois le transfert de fichiers complet terminé

Un exemple d'utilisation de cet exit utilisateur consiste à démarrer un traitement par lots qui utilise les fichiers transférés ou à envoyer un courrier électronique si le transfert a échoué.

Transmettez les arguments suivants à cet exit:

- Résultat de l'exit de transfert
- Nom de l'agent source
- Nom de l'agent de destination
- Métadonnées d'environnement
- Métadonnées de transfert
- Résultats du fichier

Les données renvoyées par cet exit sont les suivantes:

- Chaîne mise à jour à insérer dans le journal de transfert.

Implémentez l' interfaceDestinationTransferEndExit.java pour appeler le code d'exit utilisateur à ce point d'exit.

Concepts associés

Interfaces Java pour les exits utilisateur MFT

Référence associée

«Activation du débogage à distance pour les exits utilisateur MFT», à la page 1266
Lorsque vous développez vos exits utilisateur, vous pouvez utiliser un débogueur pour localiser les problèmes dans votre code.

«Exemple d'exit utilisateur de transfert de source MFT», à la page 1267

Exits utilisateur du moniteur de ressources MFT

Utilisation des exits utilisateur d'E-S de transfert MFT

Vous pouvez utiliser les exits utilisateur d'E-S de transfert Managed File Transfer pour configurer le code personnalisé afin d'effectuer le travail d'E-S de système de fichiers sous-jacent pour les transferts Managed File Transfer .

Généralement, pour les transferts MFT , un agent effectue une sélection parmi l'un des fournisseurs d'E-S intégrés pour interagir avec les systèmes de fichiers appropriés pour le transfert. Les fournisseurs d'E-S intégrés prennent en charge les types de système de fichiers suivants:

- Systèmes de fichiers UNIX-type et Windows-type standard
- z/OS Fichiers séquentiels et partitionnés z/OS (sous z/OS uniquement)
- IBM i Fichiers de sauvegarde natifs IBM i (sous IBM i uniquement)
- Files d'attente IBM MQ
- Serveurs de protocole FTP et SFTP distants (pour les agents de pont de protocole uniquement)
- Noeuds Connect:Direct distants (pour les agents de pont Connect:Direct uniquement)

Pour les systèmes de fichiers qui ne sont pas pris en charge ou pour lesquels vous avez besoin d'un comportement d'E-S personnalisé, vous pouvez écrire un exit utilisateur d'E-S de transfert.

Les exits utilisateur d'E-S de transfert utilisent l'infrastructure existante pour les exits utilisateur. Toutefois, ces exits utilisateur d'E-S de transfert diffèrent des autres exits utilisateur car leur fonction est accessible plusieurs fois tout au long du transfert pour chaque fichier.

Utilisez la propriété d'agent IOExitClasses (dans le fichier `agent.properties`) pour spécifier les classes d'exit d'E-S à charger. Séparez chaque classe d'exit par une virgule, par exemple:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Les interfaces Java pour les exits utilisateur d'E-S de transfert sont les suivantes:

Exit d'E-S
Point d'entrée principal utilisé pour déterminer si l'exit d'E-S est utilisé. Cette instance est chargée de créer des instances IOExitPath .

Vous devez spécifier uniquement l'interface d'exit d'E-S IOExit pour la propriété d'agent IOExitClasses.

IOExitPath
Représente une interface abstraite ; par exemple, un conteneur de données ou un caractère générique représentant un ensemble de conteneurs de données. Vous ne pouvez pas créer une instance de classe qui implémente cette interface. L'interface permet d'examiner le chemin et de répertorier les chemins dérivés. Les interfaces IOExitResourcePath et IOExitWildcardPath étendent IOExitPath.

IOExitChannel
Permet de lire ou d'écrire des données dans une ressource IOExitPath .

Canal IOExitRecord
Etend l'interface IOExitChannel pour les ressources IOExitPath orientées enregistrement, ce qui permet de lire ou d'écrire des données dans une ressource IOExitPath en multiples d'enregistrements.

IOExitLock
Représente un verrou sur une ressource IOExitPath pour un accès partagé ou exclusif.

z/OS **IOExitRecordResourcePath**
Etend l'interface de chemin IOExitResourcepour représenter un conteneur de données pour un fichier orienté enregistrement ; par exemple, un fichier z/OS . Vous pouvez utiliser l'interface pour localiser des données et créer des instances de canal IOExitRecordpour des opérations de lecture ou d'écriture.

Chemin d'accès à IOExitResource
Etend l'interface IOExitPath pour représenter un conteneur de données ; par exemple, un fichier ou un répertoire. Vous pouvez utiliser l'interface pour localiser des données. Si l'interface représente un répertoire, vous pouvez utiliser la méthode listPaths pour renvoyer une liste de chemins.

Chemin IOExitWildcard
Etend l'interface IOExitPath pour représenter un chemin indiquant un caractère générique. Vous pouvez utiliser cette interface pour faire correspondre plusieurs chemins IOExitResource.

IOExitProperties
Indique les propriétés qui déterminent comment Managed File Transfer gère IOExitPath pour certains aspects des E-S. Par exemple, s'il faut utiliser des fichiers intermédiaires ou relire une ressource depuis le début si un transfert est redémarré.

Concepts associés

«Personnalisation de MFT avec des exits utilisateur», à la page 1259
Vous pouvez personnaliser les fonctions d' Managed File Transfer à l'aide de vos propres programmes appelés routines d'exit utilisateur.

Référence associée

Interface IOExit.java

Interface IOExitChannel.java
Interface IOExitLock.java
Interface IOExitPath.java
Interface IOExitProperties.java
Interface IOExitRecordChannel.java
z/OS Interface IOExitRecordResourcePath.java
Interface IOExitResourcePath.java
Interface IOExitWildcardPath.java
Le fichier MFT`agent.properties`

IBM i Exemples d'exit utilisateur MFT on IBM i

Managed File Transfer fournit des exemples d'exit utilisateur spécifiques à IBM i avec votre installation. Les exemples se trouvent dans les répertoires *MQMFT_install_dir*`/samples/ioexit-IBMi` et *MQMFT_install_dir*`/samples/userexit-IBMi`.

com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit
L'exemple d'exit utilisateur com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit transfère des fichiers dans le système de fichiers QDLS sous IBM i. Une fois l'exit installé, tous les transferts vers des fichiers qui commencent par `/QDLS` utilisent automatiquement l'exit.

Pour installer cet exit, procédez comme suit:

1. Copiez le fichier `com.ibm.wmqfte.samples.ibmi.ioexits.jar` du répertoire *WMQFTE_install_dir*`/samples/ioexit-IBMi` vers le répertoire `exits` de l'agent.
2. Ajoutez com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit à la propriété IOExitClasses .
3. Redémarrez l'agent.

com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit
L'exemple d'exit utilisateur com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit se comporte comme un moniteur de fichiers MFT et transfère automatiquement des membres de fichiers physiques à partir d'une bibliothèque IBM i .

Pour exécuter cet exit, spécifiez une valeur pour la zone de métadonnées "library.qsys.monitor" (à l'aide du paramètre **-md** , par exemple). Ce paramètre prend un chemin de style IFS vers un membre de fichier et peut contenir des caractères génériques de fichier et de membre. Par exemple, /QSYS.LIB/FOO.LIB/BAR.FILE/*.MBR, /QSYS.LIB/FOO.LIB/*.FILE/BAR.MBR, /QSYS.LIB/FOO.LIB/*.FILE/*.Région d'équilibrage de charge.

Cet exemple d'exit comporte également une zone de métadonnées facultative "naming.scheme.qsys.monitor", que vous pouvez utiliser pour déterminer le schéma de dénomination utilisé lors du transfert. Par défaut, cette zone est définie sur "unix", ce qui entraîne le nom du fichier de destination F00`.MBR`. Vous pouvez également spécifier la valeur "ibmi" pour utiliser le fichier IBM i FTP FILE.MEMBER , par exemple, /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR est transféré en tant que fichier d'archives du courtier BAR.BAZ.

Pour installer cet exit, procédez comme suit:

1. Copiez le fichier `com.ibm.wmqfte.samples.ibmi.userexits.jar` du répertoire *WMQFTE_install_dir*`/samples/userexit-IBMi` vers le répertoire `exits` de l'agent.
2. Ajoutez com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit à la propriété sourceTransferStartExitClasses dans le fichier `agent.properties` .
3. Redémarrez l'agent.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit
L'exemple d'exit utilisateur com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit supprime un objet fichier vide lorsque le membre de fichier source est supprimé dans le cadre du transfert. Etant donné que les objets de fichier IBM i peuvent potentiellement contenir de nombreux membres, les objets

de fichier sont traités comme des répertoires par MFT. Par conséquent, vous ne pouvez pas effectuer d'opération de déplacement sur un objet fichier à l'aide de MFT; les opérations de déplacement sont prises en charge au niveau du membre uniquement. Par conséquent, lorsque vous effectuez une opération de déplacement sur un membre, le fichier désormais vide est laissé derrière. Utilisez cet exemple d'exit si vous souhaitez supprimer ces fichiers vides dans le cadre de la demande de transfert.

Si vous spécifiez "true" pour les métadonnées "empty.file.delete" et que vous transférez un FTEFileMember, l'exemple d'exit supprime le fichier parent si le fichier est vide.

Pour installer cet exit, procédez comme suit:

1. Copiez le fichier com.ibm.wmqfte.samples.ibmi.userexits.jar depuis *WMQFTE_install_dir*/samples/userexit-IBMi dans le répertoire exits de l'agent.
2. Ajoutez com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit à la propriété sourceTransferStartExitClasses dans le fichier agent.properties .
3. Redémarrez l'agent.

Référence associée

«Utilisation des exits utilisateur d'E-S de transfert MFT», à la page 1263
Vous pouvez utiliser les exits utilisateur d'E-S de transfert Managed File Transfer pour configurer le code personnalisé afin d'effectuer le travail d'E-S de système de fichiers sous-jacent pour les transferts Managed File Transfer .

Propriétés de l'agent MFT pour les exits utilisateur

Activation du débogage à distance pour les exits utilisateur MFT

Lorsque vous développez vos exits utilisateur, vous pouvez utiliser un débogueur pour localiser les problèmes dans votre code.

Etant donné que les exits s'exécutent dans la machine virtuelle Java qui exécute l'agent, vous ne pouvez pas utiliser le support de débogage direct qui est généralement inclus dans un environnement de développement intégré. Toutefois, vous pouvez activer le débogage à distance de la machine virtuelle Java, puis connecter un débogueur à distance approprié.

Pour activer le débogage à distance, utilisez les paramètres JVM standard **-Xdebug** et **-Xrunjdwp**. Ces propriétés sont transmises à la machine virtuelle Java qui exécute l'agent par la variable d'environnement **BFG_JVM_PROPERTIES** . Par exemple, sous AIX and Linux , les commandes suivantes démarrent l'agent et entraînent la machine virtuelle Java à écouter les connexions du débogueur sur le port TCP 8765.

```
export BFG_JVM_PROPERTIES="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,address=8765"
fteStartAgent -F TEST_AGENT
```

L'agent ne démarre pas tant que le débogueur ne se connecte pas. Utilisez la commande **set** sous Windows à la place de la commande **export** .

Vous pouvez également utiliser d'autres méthodes de communication entre le débogueur et la machine virtuelle Java. Par exemple, la machine virtuelle Java peut ouvrir la connexion au débogueur au lieu de l'inverse, ou vous pouvez utiliser la mémoire partagée au lieu de TCP. Pour plus de détails, voir la documentation Java Platform Debugger Architecture .

Vous devez utiliser le paramètre **-F** (avant-plan) lorsque vous démarrez l'agent en mode débogage à distance.

Utilisation du débogueur Eclipse

Les étapes suivantes s'appliquent à la fonction de débogage à distance dans l'environnement de développement Eclipse . Vous pouvez également utiliser d'autres débogueurs distants compatibles avec JPDA.

1. Cliquez sur **Exécuter** > **Ouvrir la boîte de dialogue de débogage** (ou sur **Exécuter** > **Configurations de débogage** ou sur **Exécuter** > **Boîte de dialogue de débogage** en fonction de votre version d' Eclipse).
2. Cliquez deux fois sur **Application Java distante** dans la liste des types de configuration pour créer une configuration de débogage.
3. Renseignez les zones de configuration et sauvegardez la configuration de débogage. Si vous avez déjà démarré la machine virtuelle Java de l'agent en mode débogage, vous pouvez vous connecter à la machine virtuelle Java maintenant.

Exemple d'exit utilisateur de transfert de source MFT

```
/*
 * A Sample Source Transfer End Exit that prints information about a transfer to standard
output.
 * If the agent is run in the background the output will be sent to the agent's event log file.
If
 * the agent is started in the foreground by specifying the -F parameter on the fteStartAgent
 * command the output will be sent to the console.
 *
 * To run the exit execute the following steps:
 *
 * Compile and build the exit into a jar file. You need the following in the class path:
 * {MQ_INSTALLATION_PATH}\mqft\lib\com.ibm.wmqfte.exitroutines.api.jar
 *
 * Put the jar in your agent's exits directory:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\exits\
 *
 * Update the agent's properties file:
 * {MQ_DATA_PATH}\config\coordQmgrName\agents\agentName\agent.properties
 * to include the following property:
 * sourceTransferEndExitClasses=[packageName.]SampleEndExit
 *
 * Restart agent to pick up the exit
 *
 * Send the agent a transfer request:
 * For example: fteCreateTransfer -sa myAgent -da YourAgent -df output.txt input.txt
 */

import java.util.List;
import java.util.Map;
import java.util.Iterator;

import com.ibm.wmqfte.exitroutine.api.SourceTransferEndExit;
import com.ibm.wmqfte.exitroutine.api.TransferExitResult;
import com.ibm.wmqfte.exitroutine.api.FileTransferResult;

public class SampleEndExit implements SourceTransferEndExit {

    public String onSourceTransferEnd(TransferExitResult transferExitResult,
            String sourceAgentName,
            String destinationAgentName,
            Map<String, String>environmentMetaData,
            Map<String, String>transferMetaData,
            List<FileTransferResult>fileResults) {

        System.out.println("Environment Meta Data: " + environmentMetaData);
        System.out.println("Transfer Meta Data:    " + transferMetaData);

        System.out.println("Source agent:          " +
                sourceAgentName);
        System.out.println("Destination agent:     " +
                destinationAgentName);

        if (fileResults.isEmpty()) {
            System.out.println("No files in the list");
            return "No files";
        }
        else {

            System.out.println( "File list: ");

            final Iterator<FileTransferResult> iterator = fileResults.iterator();
```

```
            while (iterator.hasNext()){
                final FileTransferResult thisFileSpec = iterator.next();
                System.out.println("Source file spec: "  +
                        thisFileSpec.getSourceFileSpecification() +
                        ", Destination file spec: " +
                        thisFileSpec.getDestinationFileSpecification());
            }
        }
        return "Done";
    }
}
```

Exemple d'exit utilisateur de données d'identification de pont de protocole

Pour plus d'informations sur l'utilisation de cet exemple d'exit utilisateur, voir Mappage des données d'identification pour un serveur de fichiers à l'aide de classes d'exit.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.wmqfte.exitroutine.api.CredentialExitResult;
import com.ibm.wmqfte.exitroutine.api.CredentialExitResultCode;
import com.ibm.wmqfte.exitroutine.api.CredentialPassword;
import com.ibm.wmqfte.exitroutine.api.CredentialUserId;
import com.ibm.wmqfte.exitroutine.api.Credentials;
import com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit;

/**
 * A sample protocol bridge credential exit
 *
 * This exit reads a properties file that maps mq user ids to server user ids
 * and server passwords. The format of each entry in the properties file is:
 *
 * mqUserId=serverUserId,serverPassword
 *
 * The location of the properties file is taken from the protocol bridge agent
 * property protocolBridgeCredentialConfiguration.
 *
 * To install the sample exit compile the class and export to a jar file.
 * Place the jar file in the exits subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * In the agent.properties file of the protocol bridge agent set the
 * protocolBridgeCredentialExitClasses to SampleCredentialExit
 * Create a properties file that contains the mqUserId to serverUserId and
 * serverPassword mappings applicable to the agent. In the agent.properties
 * file of the protocol bridge agent set the protocolBridgeCredentialConfiguration
 * property to the absolute path name of this properties file.
 * To activate the changes stop and restart the protocol bridge agent.
 *
 * For further information on protocol bridge credential exits refer to
 * the WebSphere MQ Managed File Transfer documentation online at:
 * https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html
 */
public class SampleCredentialExit implements ProtocolBridgeCredentialExit {

    // The map that holds mq user ID to serverUserId and serverPassword mappings
    final private Map<String,Credentials> credentialsMap = new HashMap<String, Credentials>();

    /* (non-Javadoc)
      * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#initialize(java.util.Map)
      */
     public synchronized boolean initialize(Map<String, String> bridgeProperties) {

        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the mq user ID mapping properties file
```

```
         final String propertiesFilePath = bridgeProperties.get("protocolBridgeCredentialConfiguration");

         if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
             // The properties file path has not been specified. Output an error and return false
             System.err.println("Error initializing SampleCredentialExit.");
              System.err.println("The location of the mqUserID mapping properties file has not been
specified in the
       protocolBridgeCredentialConfiguration property");
             initialisationResult = false;
         }

        if (initialisationResult) {

            // The Properties object that holds mq user ID to serverUserId and serverPassword
             // mappings from the properties file
             final Properties mappingProperties = new Properties();

             // Open and load the properties from the properties file
             final File propertiesFile = new File (propertiesFilePath);
             FileInputStream inputStream = null;
             try {
                 // Create a file input stream to the file
                 inputStream = new FileInputStream(propertiesFile);

                 // Load the properties from the file
                 mappingProperties.load(inputStream);
             }
             catch (FileNotFoundException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Unable to find the mqUserId mapping properties file: " +
propertiesFilePath);
                 initialisationResult = false;
             }
             catch (IOException ex) {
                 System.err.println("Error initializing SampleCredentialExit.");
                  System.err.println("Error loading the properties from the mqUserId mapping properties
file: " + propertiesFilePath);
                 initialisationResult = false;
             }
             finally {
                 // Close the inputStream
                 if (inputStream != null) {
                     try {
                         inputStream.close();
                     }
                     catch (IOException ex) {
                         System.err.println("Error initializing SampleCredentialExit.");
                          System.err.println("Error closing the mqUserId mapping properties file: " +
propertiesFilePath);
                         initialisationResult = false;
                     }
                 }
             }

             if (initialisationResult) {
                 // Populate the map of mqUserId to server credentials from the properties
                 final Enumeration<?> propertyNames = mappingProperties.propertyNames();
                  while ( propertyNames.hasMoreElements()) {
                     final Object name = propertyNames.nextElement();
                     if (name instanceof String ) {
                         final String mqUserId = ((String)name).trim();
                          // Get the value and split into serverUserId and serverPassword
                        final String value = mappingProperties.getProperty(mqUserId);
                        final StringTokenizer valueTokenizer = new StringTokenizer(value, ",");
                         String serverUserId = "";
                         String serverPassword = "";
                         if (valueTokenizer.hasMoreTokens()) {
                             serverUserId = valueTokenizer.nextToken().trim();
                         }
                         if (valueTokenizer.hasMoreTokens()) {
                             serverPassword = valueTokenizer.nextToken().trim();
                         }
                         // Create a Credential object from the serverUserId and serverPassword
final Credentials credentials = new Credentials(new CredentialUserId(serverUserId), new
CredentialPassword(serverPassword));
                         // Insert the credentials into the map
                         credentialsMap.put(mqUserId, credentials);
                     }
                 }

            }
         }
```

```
        return initialisationResult;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#mapMQUserId(java.lang.String)
     */
    public synchronized CredentialExitResult mapMQUserId(String mqUserId) {
        CredentialExitResult result = null;
        // Attempt to get the server credentials for the given mq user id
        final Credentials credentials = credentialsMap.get(mqUserId.trim());
        if ( credentials == null) {
            // No entry has been found so return no mapping found with no credentials
            result = new CredentialExitResult(CredentialExitResultCode.NO_MAPPING_FOUND, null);
        }
        else {
            // Some credentials have been found so return success to the user along with the credentials
            result = new CredentialExitResult(CredentialExitResultCode.USER_SUCCESSFULLY_MAPPED,
credentials);
        }
        return result;
    }
     /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgeCredentialExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
        // Nothing to do in this method because there are no resources that need to be released
    }
}
```

Exemple d'exit utilisateur de propriétés de pont de protocole

Pour plus d'informations sur l'utilisation de cet exemple d'exit utilisateur, voir ProtocolBridgePropertiesExit2: Recherche des propriétés du serveur de fichiers de protocole

SamplePropertiesExit2.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Properties;

import com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2;
import com.ibm.wmqfte.exitroutine.api.ProtocolServerPropertyConstants;

/**
 * A sample protocol bridge properties exit. This exit reads a properties file
 * that contains properties for protocol servers.
 * <p>
 * The format of each entry in the properties file is:
 * {@literal serverName=type://host:port}
 * Ensure there is a default entry such as
 * {@literal default=type://host:port}
 * otherwise the agent will fail to start with a BFGBR0168 as it must have a
 * default server.
 * <p>
 * The location of the properties file is taken from the protocol bridge agent
 * property {@code protocolBridgePropertiesConfiguration}.
 * <p>
 * The methods {@code getCredentialLocation} returns the location of the associated
 * ProtocolBridgeCredentials.xml, this sample it is defined to be stored in a directory
 * defined by the environment variable CREDENTIALSHOME
 * <p>
 * To install the sample exit:
 * <ol>
 * <li>Compile the class and export to a jar file.
 * <li>Place the jar file in the {@code exits} subdirectory of the agent data directory
 * of the protocol bridge agent on which the exit is to be installed.
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the {@code protocolBridgePropertiesExitClasses} to
 * {@code SamplePropertiesExit2}.
 * <li>Create a properties file that contains the appropriate properties to specify the
 * required servers.
```

```
 * <li>In the {@code agent.properties} file of the protocol bridge agent
 * set the <code>protocolBridgePropertiesConfiguration</code> property to the
 * absolute path name of this properties file.
 * <li>To activate the changes stop and restart the protocol bridge agent.
 * </ol>
 * <p>
 * For further information on protocol bridge properties exits refer to the
 * WebSphere MQ Managed File Transfer documentation online at:
 * <p>
 * {@link https://www.ibm.com/docs/SSEP7X_7.0.4/welcome/WelcomePagev7r0.html}
 */
public class SamplePropertiesExit2 implements ProtocolBridgePropertiesExit2 {

    /**
     * Helper class to encapsulate protocol server information.
     */
    private static class ServerInformation {
        private final String type;
        private final String host;
        private final int port;

        public ServerInformation(String url) {
            int index = url.indexOf("://");
            if (index == -1) throw new IllegalArgumentException("Invalid server URL: "+url);
            type = url.substring(0, index);

            int portIndex = url.indexOf(":", index+3);
            if (portIndex == -1) {
                host = url.substring(index+3);
                port = -1;
            } else {
                host = url.substring(index+3,portIndex);
                port = Integer.parseInt(url.substring(portIndex+1));
            }
        }

        public String getType() {
            return type;
        }

        public String getHost() {
            return host;
        }

        public int getPort() {
            return port;
        }
    }

    /** A {@code Map} that holds information for each configured protocol server */
    final private Map<String, ServerInformation> servers = new HashMap<String, ServerInformation>();


    /* (non-Javadoc)
     * @see
com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#getProtocolServerProperties(java.lang.String)
     */
    public Properties getProtocolServerProperties(String protocolServerName) {
        // Attempt to get the protocol server information for the given protocol server name
        // If no name has been supplied then this implies the default.
        final ServerInformation info;
        if (protocolServerName == null || protocolServerName.length() == 0) {
            protocolServerName = "default";
        }
        info = servers.get(protocolServerName);

        // Build the return set of properties from the collected protocol server information, when
available.
        // The properties set here is the minimal set of properties to be a valid set.
        final Properties result;
        if (info != null) {
            result = new Properties();
            result.setProperty(ProtocolServerPropertyConstants.SERVER_NAME, protocolServerName);
            result.setProperty(ProtocolServerPropertyConstants.SERVER_TYPE, info.getType());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_HOST_NAME, info.getHost());
            if (info.getPort() != -1)
result.setProperty(ProtocolServerPropertyConstants.SERVER_PORT_VALUE, ""+info.getPort());
            result.setProperty(ProtocolServerPropertyConstants.SERVER_PLATFORM, "UNIX");
            if (info.getType().toUpperCase().startsWith("FTP")) {        // FTP & FTPS
                result.setProperty(ProtocolServerPropertyConstants.SERVER_TIMEZONE, "Europe/London");
                result.setProperty(ProtocolServerPropertyConstants.SERVER_LOCALE, "en-GB");
            }
```

```
            result.setProperty(ProtocolServerPropertyConstants.SERVER_FILE_ENCODING, "UTF-8");
        } else {
            System.err.println("Error no default protocol file server entry has been supplied");
            result = null;
        }

        return result;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#initialize(java.util.Map)
     */
    public boolean initialize(Map<String, String> bridgeProperties) {
        // Flag to indicate whether the exit has been successfully initialized or not
        boolean initialisationResult = true;

        // Get the path of the properties file
        final String propertiesFilePath = bridgeProperties.get("protocolBridgePropertiesConfiguration");
        if (propertiesFilePath == null || propertiesFilePath.length() == 0) {
            // The protocol server properties file path has not been specified. Output an error and
return false
            System.err.println("Error initializing SamplePropertiesExit.");
            System.err.println("The location of the protocol server properties file has not been
specified in the
      protocolBridgePropertiesConfiguration property");
            initialisationResult = false;
        }

        if (initialisationResult) {
            // The Properties object that holds protocol server information
            final Properties mappingProperties = new Properties();

            // Open and load the properties from the properties file
            final File propertiesFile = new File (propertiesFilePath);
            FileInputStream inputStream = null;
            try {
                // Create a file input stream to the file
                inputStream = new FileInputStream(propertiesFile);

                // Load the properties from the file
                mappingProperties.load(inputStream);
            } catch (final FileNotFoundException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Unable to find the protocol server properties file: " +
propertiesFilePath);
                initialisationResult = false;
            } catch (final IOException ex) {
                System.err.println("Error initializing SamplePropertiesExit.");
                System.err.println("Error loading the properties from the protocol server properties
file: " + propertiesFilePath);
                initialisationResult = false;
            } finally {
                // Close the inputStream
                if (inputStream != null) {
                    try {
                        inputStream.close();
                    } catch (final IOException ex) {
                        System.err.println("Error initializing SamplePropertiesExit.");
                        System.err.println("Error closing the protocol server properties file: " +
propertiesFilePath);
                        initialisationResult = false;
                    }
                }
            }

            if (initialisationResult) {
                // Populate the map of protocol servers from the properties
                for (Entry<Object, Object> entry : mappingProperties.entrySet()) {
                    final String serverName = (String)entry.getKey();
                    final ServerInformation info = new ServerInformation((String)entry.getValue());
                    servers.put(serverName, info);
                }
            }
        }

        return initialisationResult;
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit#shutdown(java.util.Map)
     */
    public void shutdown(Map<String, String> bridgeProperties) {
```

```
        // Nothing to do in this method because there are no resources that need to be released
    }

    /* (non-Javadoc)
     * @see com.ibm.wmqfte.exitroutine.api.ProtocolBridgePropertiesExit2#getCredentialLocation()
     */
    public String getCredentialLocation() {
        String envLocationPath;
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            // Windows style
            envLocationPath = "%CREDENTIALSHOME%\\ProtocolBridgeCredentials.xml";
        }
        else {
            // Unix style
            envLocationPath = "$CREDENTIALSHOME/ProtocolBridgeCredentials.xml";
        }
        return envLocationPath;
    }

}
```

Contrôle de MFT via le placement de messages dans la file d'attente de commandes d'agent

Vous pouvez écrire une application qui contrôle Managed File Transfer via le placement de messages dans des files d'attente de commandes d'agent.

Vous pouvez placer un message dans la file d'attente de commandes d'un agent pour demander que l'agent effectue l'une des actions suivantes :

- Créer un transfert de fichier
- Créer un transfert de fichier planifié
- Annuler un transfert de fichier
- Annuler un transfert de fichier planifié
- Appeler une commande
- Créer un moniteur
- Suppimer un moniteur
- Envoyer une commande ping pour vérifier que l'agent est actif

Pour que l'agent effectue l'une de ces actions, le message doit être dans un format XML conforme à l'un des schémas suivants :

FileTransfer.xsd
Vous pouvez utiliser des messages dans ce format pour créer un transfert de fichier ou un transfert de fichier planifié, appeler une commande, ou annuler un transfert de fichier ou un transfert de fichier planifié. Pour plus d'informations, voir Format de message de demande de transfert de fichier.

Monitor.xsd
Vous pouvez utiliser des messages dans ce format pour créer ou supprimer un moniteur de ressources. Pour plus d'informations, voir Formats de message de demande de moniteur MFT.

PingAgent.xsd
Vous pouvez utiliser des messages dans ce format pour envoyer une commande ping à un agent afin de vérifier qu'il est actif. Pour plus d'informations, voir Format de message de demande d'envoi d'une commande ping à un agent MFT.

L'agent renvoie une réponse aux messages de demande. Le message de réponse est placé dans une file d'attente de réponses qui est définie dans le message de demande. Son format est le format XML défini par le schéma suivant :

Reply.xsd
Pour plus d'informations, voir Format de message de réponse d'agent MFT.

Développement d'applications pour MQ Telemetry

Les applications de télémétrie intègrent des dispositifs de détection et de contrôle à d'autres sources d'informations disponibles sur Internet et dans les entreprises.

Développez des applications pour MQ Telemetry à l'aide de modèles de conception, d'exemples travaillés, d'exemples de programmes, de concepts de programmation et d'informations de référence.

Concepts associés

MQ Telemetry

Scénarios d'utilisation de la télémétrie

Tâches associées

Installation de MQ Telemetry

Administration de MQ Telemetry

Traitement des incidents liés à MQ Telemetry

Référence associée

Référence MQ Telemetry

IBM MQ Telemetry Transport exemples de programmes

Des exemples de script sont fournis avec un exemple d'application client IBM MQ Telemetry Transport v3 (`mqttv3app.jar`). Pour IBM MQ 8.0.0 et les versions ultérieures, l'exemple d'application client n'est plus inclus dans MQ Telemetry. Il faisait partie du IBM Messaging Telemetry Clients SupportPac(qui n'est plus disponible). Des exemples d'application similaires restent disponibles gratuitement depuis Eclipse Paho et MQTT.org.

Pour les informations et les téléchargements les plus récents, voir les ressources suivantes :

- Le projet Eclipse Paho et MQTT.org permettent de télécharger gratuitement les derniers clients et des exemples de télémétrie pour plusieurs langages de programmation. Utilisez ces sites afin de développer des exemples de programme pour la publication et l'abonnement IBM MQ Telemetry Transport ainsi que pour l'ajout de fonctions de sécurité.
- IBM Messaging Telemetry Clients SupportPac ne peut plus être téléchargé. Si vous disposez d'une copie que vous avez téléchargée précédemment, celle-ci contient les éléments suivants :
  - La version MA9B de IBM Messaging Telemetry Clients SupportPac inclut un exemple d'application compilé (`mqttv3app.jar`) et une bibliothèque client associée (`mqttv3.jar`). Ils sont accessibles dans les répertoires suivants :
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Dans la version MA9C de ce SupportPac, le répertoire et le contenu `/SDK/` ont été supprimés :
    - Seule la source de l'exemple d'application (`mqttv3app.jar`) a été fournie. Elle se trouvait dans le répertoire suivant :

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La bibliothèque client compilée était toujours fournie. Elle se trouvait dans le répertoire suivant :

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Si vous disposez toujours d'une copie du IBM Messaging Telemetry Clients SupportPac(qui n'est plus disponible), des informations sur l'installation et l'exécution du modèle d'application sont fournies dans Vérification de l'installation de MQ Telemetry à l'aide de la ligne de commande.

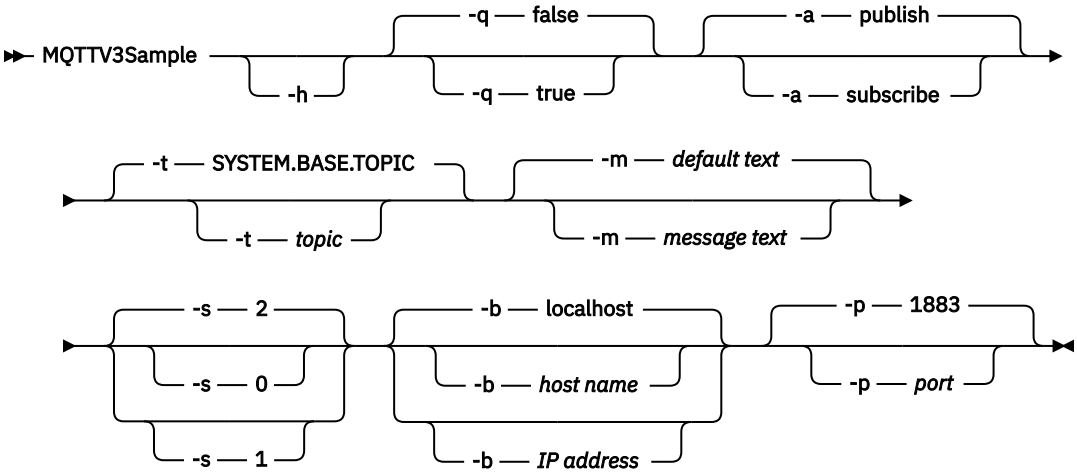
Programme MQTTV3Sample

Informations de référence sur les exemples de syntaxe et de paramètres du programme `MQTTV3Sample` .

Objet

Le programme `MQTTV3Sample` peut être utilisé pour publier un message et s'abonner à une rubrique. Pour plus d'informations sur l'obtention de cet exemple de programme, voir «IBM MQ Telemetry Transport exemples de programmes», à la page 1274.

MQTTV3Sample syntax



Paramètres

-h
Imprime ce texte d'aide et quitte

-q
Définit le mode silencieux au lieu d'utiliser le mode par défaut false.

-a
Définissez la publication ou l'abonnement, au lieu d'utiliser l'action par défaut de publication.

-t
Publier ou s'abonner à la rubrique, au lieu de publier ou de s'abonner à la rubrique par défaut

-m
Publiez le texte du message au lieu d'envoyer le texte de publication par défaut, "Hello from an MQTT v3 application".

-s
Définissez QoS au lieu d'utiliser la valeur par défaut QoS, 2.

-b
Connectez-vous à ce nom d'hôte ou à cette adresse IP au lieu de vous connecter au nom d'hôte par défaut, localhost.

-p
Utilisez ce port à la place de la valeur par défaut, 1883.

Exécutez le programme MQTTV3Sample

Pour vous abonner à une rubrique sous Windows, utilisez la commande suivante:

```
run MQTTV3Sample -a subscribe
```

Pour publier un message sur Windows, utilisez la commande suivante:

```
run MQTTV3Sample
```

MQTT Concepts de programmation du client

Les concepts décrits dans cette section vous aident à comprendre les bibliothèques client de MQTT protocol. Ces concepts complètent la documentation d'API accompagnant les bibliothèques client.

Pour les informations et les téléchargements les plus récents, voir les ressources suivantes :

- Le projet Eclipse Paho et MQTT.org permettent de télécharger gratuitement les derniers clients et des exemples de télémétrie pour plusieurs langages de programmation. Utilisez ces sites afin de développer des exemples de programme pour la publication et l'abonnement IBM MQ Telemetry Transport ainsi que pour l'ajout de fonctions de sécurité.
- IBM Messaging Telemetry Clients SupportPac ne peut plus être téléchargé. Si vous disposez d'une copie que vous avez téléchargée précédemment, celle-ci contient les éléments suivants :
  - La version MA9B de IBM Messaging Telemetry Clients SupportPac inclut un exemple d'application compilé (`mqttv3app.jar`) et une bibliothèque client associée (`mqttv3.jar`). Ils sont accessibles dans les répertoires suivants :
    - `ma9b/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9b/SDK/clients/java/org.eclipse.paho.client.mqttv3.jar`
  - Dans la version MA9C de ce SupportPac, le répertoire et le contenu `/SDK/` ont été supprimés :
    - Seule la source de l'exemple d'application (`mqttv3app.jar`) a été fournie. Elle se trouvait dans le répertoire suivant :

      ```
      ma9c/clients/java/samples/org/eclipse/paho/sample/mqttv3app/*.java
      ```

    - La bibliothèque client compilée était toujours fournie. Elle se trouvait dans le répertoire suivant :

      ```
      ma9c/clients/java/org.eclipse.paho.client.mqttv3-1.0.2.jar
      ```

Pour développer et exécuter un client MQTT , vous devez copier ou installer ces ressources sur l'unité client. Il n'est pas nécessaire d'installer un environnement d'exécution client distinct.

Les conditions de la licence applicables aux clients sont associées au serveur auquel ils sont connectés.

Les bibliothèques client MQTT sont des implémentations de référence de MQTT protocol. Vous pouvez implémenter vos propres clients dans différents langages adaptés à différentes plateformes de dispositif. Voir IBM MQ Telemetry Transport format et protocole.

La documentation de l'API ne pose aucune hypothèse quant au serveur MQTT auquel le client est connecté. Le comportement du client peut varier légèrement lorsqu'il est connecté à des serveurs différents. Les descriptions qui suivent décrivent le comportement du client lorsqu'il est connecté au service de télémétrie IBM MQ .

Rappels et synchronisation dans les applications client MQTT

Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Rappels

Remarque : Consultez le site Web Eclipse Paho pour connaître les dernières modifications apportées à `MqttCallback`. Par exemple, `MqttCallback` est défini en tant qu'interface dans la version Paho du client et les méthodes asynchrones sont fournies par la classe Paho `MqttAsyncClient` .

L'interface `MqttCallback` comporte trois méthodes de rappel:

connectionLost(java.lang.Throwable cause)

connectionLost est appelé lorsqu'une erreur de communication provoque la suppression de la connexion. Cette méthode est également appelée si le serveur supprime la connexion à la suite d'une erreur liée au serveur après l'établissement de la connexion. Les erreurs de serveur sont consignées dans le journal des erreurs du gestionnaire de files d'attente. Le serveur supprime la connexion vers le client, et le client appelle MqttCallback.connectionLost.

Les seules erreurs distantes émises en tant qu'exceptions sur la même unité d'exécution que l'application client sont les exceptions provenant de MqttClient.connect. Les erreurs détectées par le serveur une fois la connexion établie sont signalées à la méthode de rappel MqttCallback.connectionLost en tant que throwables.

Les erreurs de serveur classiques résultant de connectionLost sont des erreurs d'autorisation. Par exemple, le serveur de télémétrie tente de diffuser une publication sur une rubrique au nom d'un client qui n'est pas autorisé à diffuser de publication sur cette rubrique. Toute action entraînant le renvoi d'un code de condition MQCC_FAIL au serveur de télémétrie peut provoquer la suppression de la connexion.

deliveryComplete(IMqttDeliveryToken token)

deliveryComplete est appelé par le client MQTT pour renvoyer un jeton de distribution à l'application client ; voir «Jetons de distribution», à la page 1283. Lorsque vous utilisez un jeton de distribution, la fonction de rappel peut accéder au message publié avec la méthode token.getMessage.

Lorsque le rappel d'application renvoie le contrôle au client MQTT après avoir été appelé par la méthode deliveryComplete , la distribution est terminée. Jusqu'à l'achèvement de la distribution, les messages avec la qualité de service QoS 1 ou 2 sont conservés dans la classe de persistance.

L'appel à deliveryComplete est un point de synchronisation entre l'application et la classe de persistance. La méthode deliveryComplete n'est jamais appelée deux fois pour le même message.

Lorsque le rappel d'application est renvoyé de deliveryComplete au client MQTT , le client appelle MqttClientPersistence.remove pour les messages avec QoS 1 ou 2. MqttClientPersistence.remove supprime la copie stockée en local du message publié.

D'un point de vue du traitement de transaction, l'appel de deliveryComplete est une transaction à phase unique qui valide la distribution. Si le traitement échoue au cours du rappel, lors du redémarrage du client, MqttClientPersistence.remove est rappelé pour supprimer la copie locale du message publié. Il n'est pas fait appel une nouvelle fois à la procédure de rappel. Si vous utilisez le rappel pour stocker un journal des messages distribués, vous ne pouvez pas synchroniser le journal avec le client MQTT . Si vous voulez stocker un journal de façon fiable, mettez à jour le journal dans la classe MqttClientPersistence.

Le jeton de distribution et le message sont référencés par l'unité d'exécution principale de l'application et le client MQTT. Le client MQTT déréférence l'objet MqttMessage lorsque la distribution est terminée et l'objet de jeton de distribution lorsque le client se déconnecte. L'objet MqttMessage peut faire l'objet d'une récupération de place une fois la distribution terminée si l'application client la déréférence. Le jeton de distribution peut faire l'objet d'un nettoyage de mémoire une fois la session déconnectée.

Vous pouvez obtenir les attributs IMqttDeliveryToken et MqttMessage après la publication d'un message. Si vous tentez de définir des attributs MqttMessage après la publication du message, le résultat est indéfini.

Le client MQTT continue de traiter les accusés de réception de distribution si le client se reconnecte à la session précédente avec le même ClientIdentifier ; voir «Sessions propres», à la page 1280. L'application client MQTT doit définir MqttClient.CleanSession sur false pour la session précédente et sur false dans la nouvelle session. Le client MQTT crée de nouveaux jetons de distribution et les objets message dans la nouvelle session pour les distributions en attente. Il récupère les objets à l'aide de la classe MqttClientPersistence. Si le client d'application fait encore référence aux anciens jetons de distribution et messages, supprimez ces références. La fonction de rappel de l'application est appelée dans la nouvelle session pour toute livraison initiée dans la session précédente et terminé dans cette session.

La fonction de rappel de l'application est appelée après la connexion du client d'application lorsqu'une distribution en attente est terminée. Avant la connexion du client d'application, il peut extraire les distributions en attente à l'aide de la méthode `MqttClient.getPendingDeliveryTokens`.

Notez que l'application client a créé à l'origine l'objet de message qui est publié et son tableau d'octets de contenu. Le client MQTT fait référence à ces objets. L'objet message renvoyé par le jeton de distribution dans la méthode `token.getMessage` n'est pas nécessairement le même objet message créé par le client. Si une nouvelle instance client MQTT recrée le jeton de distribution, la classe `MqttClientPersistence` recrée l'objet `MqttMessage`. Pour plus de cohérence, `token.getMessage` renvoie la valeur `null` si `token.isCompleted` a la valeur `true`, indépendamment du fait que l'objet message a été créé par le client d'application ou par la classe `MqttClientPersistence`.

`messageArrived(String topic, MqttMessage message)`

`messageArrived` est appelé lorsqu'une publication arrive pour le client qui correspond à une rubrique d'abonnement. `topic` est la rubrique de publication, et non le filtre d'abonnement. Ces deux éléments peuvent être différents si le filtre contient des caractères génériques.

Si la rubrique correspond à plusieurs abonnements créés par le client, ce dernier reçoit plusieurs copies de la publication. Si un client diffuse une publication sur une rubrique à laquelle il est également abonné, il reçoit une copie de sa propre publication.

Si le message est envoyé avec un QoS de 1 ou 2, il est stocké par la classe `MqttClientPersistence` avant que le client MQTT n'appelle `messageArrived`. `messageArrived` se comporte comme `deliveryComplete` : il est appelé une seule fois pour une publication, et la copie locale de la publication est retirée par `MqttClientPersistence.remove` lorsque `messageArrived` retourne au client MQTT. Le client MQTT supprime ses références à la rubrique et au message lorsque `messageArrived` revient au client MQTT . Les rubriques et les objets message font l'objet d'un nettoyage de mémoire, si le client d'application n'a pas placé une référence sur les objets.

Rappels, unités d'exécution et synchronisation des applications client

Le client MQTT appelle une méthode de rappel sur une unité d'exécution distincte de l'unité d'exécution de l'application principale. L'application client ne crée pas d'unité d'exécution pour le rappel, elle est créée par le client MQTT .

Le client MQTT synchronise les méthodes de rappel. Seule une instance de méthode de rappel peut s'exécuter à la fois. La synchronisation facilite la mise à jour des objets qui pointent les publications qui ont été distribuées. Une seule instance de `MqttCallback.deliveryComplete` s'exécute à la fois, ce qui permet de rendre plus sûre la mise à jour du pointage sans effectuer de synchronisation supplémentaire. Il se peut également qu'une seule publication arrive à la fois. Votre code se trouvant dans la méthode `messageArrived` peut mettre à jour un objet sans le synchroniser. Si vous faites référence au pointage ou que l'objet est en cours de mise à jour, dans une autre unité d'exécution synchronisez le pointage ou l'objet.

Le jeton de distribution fournit un mécanisme de synchronisation entre l'unité d'exécution de l'application principale et la distribution d'une publication. La méthode `token.waitForCompletion` attend la fin de la distribution d'une publication spécifique ou l'expiration d'un délai d'attente facultatif. Vous pouvez utiliser `token.waitForCompletion` de la manière suivante pour traiter une publication à la fois.

Pour réaliser la synchronisation avec la méthode `MqttCallback.deliveryComplete`. Ce n'est que lorsque `MqttCallback.deliveryComplete` revient au client MQTT que `token.waitForCompletion` reprend. L'utilisation de ce mécanisme vous permet de synchroniser le code s'exécutant dans `MqttCallback.deliveryComplete` avant que le code ne s'exécute dans l'unité d'exécution de l'application principale.

Que se passe-t-il si vous voulez diffuser une publication sans attendre que chaque publication soit distribuée, mais que vous voulez obtenir la confirmation que toutes les publications ont été distribuées ? Si vous diffusez une publication sur une seule unité d'exécution, la dernière publication à être envoyée est également la dernière à être distribuée.

Synchronisation des demandes envoyées au serveur

Tableau 188, à la page 1279 décrit les méthodes du client MQTT Java qui envoient une demande au serveur. A moins que le client d'application ne définisse un délai d'attente indéfini, le client n'attend jamais indéfiniment le serveur. Si le client se bloque, il s'agit soit d'un problème de programmation d'application, soit d'un défaut du client MQTT .

Tableau 188. Comportement de synchronisation de méthodes entraînant l'envoi de demandes au serveur

Méthode	Synchronisation	Intervalle de délai d'attente
`MqttClient.Connect`	Attend qu'une connexion soit établie avec le serveur.	La valeur par défaut est 30 secondes, ou définie par un paramètre, puis émet une exception.
`MqttClient.Disconnect`	Attend que le client MQTT termine le travail qu'il doit effectuer et que la session TCP/IP se déconnecte.	
`MqttClient.Subscribe`	Attend la fin de la méthode d'abonnement ou de UnSubscribe .	
`MqttClient.UnSubscribe`		
`MqttClient.Publish`	Revient immédiatement à l'unité d'exécution d'application de l'application après avoir transmis la requête au client MQTT.	Néant.
`IMqttDeliveryToken.waitForCompletion`	Attend le retour du jeton de distribution.	Indéfinie ou définie en tant que paramètre.

Concepts associés

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur

le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Sessions propres

Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Lorsque vous connectez une application client MQTT à l'aide de la méthode `MqttClient.connect` , le client identifie la connexion à l'aide de l'identificateur client et de l'adresse du serveur. Le serveur vérifie si les informations de session ont été enregistrées à partir de la précédente connexion au serveur. Si une session précédente existe, et que `cleanSession=true`, alors les précédentes informations de session côté client et côté serveur sont supprimées. Si `cleanSession=false` la session précédente est reprise. Si aucune session précédente n'existe, une nouvelle session est démarrée.

Remarque : L'administrateur IBM MQ peut forcer la fermeture d'une session ouverte et supprimer toutes les informations de session. Si le client ouvre à nouveau la session avec `cleanSession=false`, une nouvelle session est démarrée.

Publications

Si vous utilisez l'objet par défaut `MqttConnectOptions`, ou que vous attribuez à `MqttConnectOptions.cleanSession` la valeur `true` avant de connecter le client, toutes les distributions de publication en attente pour le client sont supprimées lorsque le client se connecte.

Le paramètre de session propre n'a aucun impact sur les publications envoyées avec la qualité de service `QoS=0`. Pour `QoS=1` et `QoS=2`, l'utilisation de `cleanSession=true` peut entraîner la perte d'une publication.

Abonnements

Si vous utilisez la valeur par défaut `MqttConnectOptions`ou que vous définissez `MqttConnectOptions.cleanSession` sur `true` avant de connecter le client, les anciens abonnements du client sont supprimés lorsque le client se connecte. Les nouveaux abonnements effectués par le client lors de la session sont supprimés lorsque celui-ci se déconnecte.

Si vous définissez `MqttConnectOptions.cleanSession` sur `false` avant de se connecter, tous les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour le client avant sa connexion. Tous les abonnements restent actifs au moment de la déconnexion du client.

Un autre moyen de comprendre l'impact de l'attribut `cleanSession` sur les abonnements consiste à le considérer comme un attribut modal. Dans le cadre de son mode par défaut `cleanSession=true`, le client crée des abonnements et reçoit des publications uniquement dans la portée de la session. En mode alternatif, `cleanSession=false`, les abonnements sont durables. Le client peut se connecter et se déconnecter, et ses abonnements restent actifs. Lorsque le client se reconnecte, il reçoit les publications non distribuées. Lors de la connexion, il peut modifier l'ensemble des abonnements qui sont actifs en son nom.

Vous devez définir le mode `cleanSession` avant de vous connecter ; le mode dure toute la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez de l'utilisation de `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimés.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une

demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Identificateur de client

L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

L'identificateur de client est utilisé dans l'administration d'un système MQTT . Avec potentiellement des centaines de milliers de clients à administrer, vous devez pouvoir identifier rapidement un client particulier. Par exemple, supposons qu'un périphérique ait mal fonctionné et que vous soyez averti, par exemple par un client qui sonne un centre d'assistance. Le client doit être en mesure d'identifier l'unité et vous devez être en mesure de corréler cette identification avec le serveur qui est généralement connecté au client.

Lorsque vous parcourez les connexions client MQTT , chaque connexion est libellée avec l'identificateur client. Pour vous aider à déterminer la meilleure façon de mapper cet identificateur au périphérique et au serveur, posez-vous les questions suivantes:

- Convient-il de gérer et d'utiliser une base de données qui mappe chaque unité à un identificateur client et à un serveur?
- Le nom de l'unité peut-il identifier le serveur auquel elle est connectée?
- Avez-vous besoin d'une table de correspondance qui mappe un identificateur client à une unité physique?
- L'identificateur client identifie-t-il un dispositif particulier ou une application s'exécutant sur ce client ?
- Si un client remplace une unité défectueuse par une nouvelle, la nouvelle unité a-t-elle le même identificateur que l'ancienne unité ou attribuez-vous un nouvel identificateur? (Si vous modifiez une unité physique et que vous conservez le même identificateur, les publications en attente et les abonnements actifs sont automatiquement transférés vers la nouvelle unité.)

Vous avez également besoin d'un système pour vous assurer que les identificateurs de client sont uniques et vous devez disposer d'un processus fiable pour définir l'identificateur sur le client. Si l'unité client est une "boîte noire", sans interface utilisateur, vous pouvez fabriquer l'unité avec un identificateur client, ou vous pouvez avoir un processus d'installation et de configuration de logiciel qui configure l'unité avant qu'elle ne soit activée.

Pour que l'identificateur reste court et unique, vous pouvez créer un identificateur client à partir de l'adresse MAC du périphérique 48 bits. Si la taille de la transmission n'est pas un problème critique, vous pouvez utiliser les 17 octets restants pour faciliter l'administration de l'adresse.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Jetons de distribution

Lorsqu'un client diffuse une publication sur un sujet, un jeton de distribution est créé. Utilisez le jeton de distribution pour surveiller la distribution d'une publication ou pour bloquer l'application client jusqu'à ce que la distribution soit terminée.

Le jeton est un objet `MqttDeliveryToken`. Il est créé en appelant la méthode MqttTopic.publish() et conservé par le client MQTT jusqu'à ce que la session client soit déconnectée et que la distribution soit terminée.

L'utilisation normale d'un jeton consiste à vérifier si la distribution a été effectuée. Bloquez l'application client jusqu'à ce que la distribution soit terminée en utilisant le jeton renvoyé pour appeler `token.waitForCompletion`. Vous pouvez également indiquer un gestionnaire `MqttCallBack`. Lorsque le client MQTT a reçu tous les accusés réception, qu'il attend dans le cadre de la distribution de la publication, il appelle `MqttCallBack.deliveryComplete` en envoyant le jeton de distribution sous la forme d'un paramètre.

Jusqu'à l'achèvement de la distribution, vous pouvez inspecter la publication à l'aide du jeton de distribution renvoyé en appelant `token.getMessage`.

Distributions terminées

L'achèvement des distributions est asynchrone et dépend de la qualité de service associée à la publication.

Au plus une fois

`QoS=0`
La distribution est terminée immédiatement en retour de `MqttTopic.publish`. `MqttCallback.deliveryComplete` est appelé immédiatement.

Au moins une fois

`QoS=1`
La distribution est terminée lorsqu'un accusé de réception de la publication a été reçu du gestionnaire de files d'attente. `MqttCallback.deliveryComplete` est appelé lors de la réception de l'accusé de réception. Le message peut être distribué plusieurs fois avant que `MqttCallback.deliveryComplete` ne soit appelé si les communications sont lentes ou peu fiables.

Une seule fois

`QoS=2`
La distribution est achevée lorsque le client reçoit un message d'achèvement indiquant que la publication a été diffusée aux abonnés. `MqttCallback.deliveryComplete` est appelé dès la réception du message de publication. Il n'attend pas le message d'achèvement.

Dans de rares cas, votre application client risque de ne pas revenir au client MQTT depuis `MqttCallback.deliveryComplete` normalement. Vous savez alors que la distribution est terminée parce que `MqttCallback.deliveryComplete` a été appelé. Si le client redémarre la même session, `MqttCallback.deliveryComplete` n'est pas rappelé.

Distributions incomplètes

Si la distribution est incomplète après la déconnexion de la session du client, vous pouvez connecter le client à nouveau et terminer la distribution. Vous pouvez uniquement terminer la distribution d'un message si ce dernier a été publié dans une session avec l'attribut `MqttConnectionOptions` à la valeur `false`.

Créez le client à l'aide du même identificateur de client et adresse de serveur, puis connectez-vous en attribuant une nouvelle fois à l'attribut `cleanSession` `MqttConnectionOptions` la valeur `false`. Si vous attribuez à `cleanSession` la valeur `true`, les jetons de distribution en attente sont rebutés.

Vous pouvez vérifier s'il existe des distributions en attente en appelant `MqttClient.getPendingDeliveryTokens`. Vous pouvez appeler `MqttClient.getPendingDeliveryTokens` avant de connecter le client.

Concepts associés

Rappels et synchronisation dans les applications client MQTT

Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres

Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client

L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Publication Last will and testament (dernières volontés et testament)

Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT

Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT

Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT

Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements

Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT

Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Publication Last will and testament (dernières volontés et testament)

Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Créez une rubrique pour "dernières volontés et testament". Vous pouvez créer une rubrique telle que `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Configurez une "dernière volonté et testament" à l'aide de la méthode `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Pensez à créer un horodatage dans le message `lastWillPayload`. Incluez d'autres informations client qui permettent d'identifier le client et les circonstances de la connexion. Transmettez l'objet `MqttConnectionOptions` au constructeur `MqttClient`.

Définissez `lastWillQos` sur 1 ou 2, pour rendre le message persistant dans IBM MQet pour garantir la distribution. Pour conserver les dernières informations de connexion perdues, attribuez à `lastWillRetained` la valeur `true`.

La publication de type "dernières volontés et testament" est envoyée aux abonnés si la connexion est interrompue de manière inattendue. Elle est envoyée si la connexion s'arrête sans que le client appelle la méthode `MqttClient.disconnect`.

Pour surveiller les connexions, complétez la publication de type "dernières volontés et testament" avec d'autres publications afin d'enregistrer des connexions et des déconnexions programmées.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Persistance des messages dans les clients MQTT

Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Dans MQTT, la persistance des messages présente deux aspects: comment le message est transféré et s'il est mis en file d'attente dans IBM MQ en tant que message persistant.

1. Le client MQTT combine la persistance des messages et la qualité de service. Selon la qualité de service que vous choisissez pour un message, celui-ci devient persistant. La persistance de message est nécessaire pour implémenter la qualité de service requise.

   Si vous indiquez "au moins une fois", `QoS=0`, le client supprime le message dès qu'il est publié. S'il se produit un incident lors du traitement en amont du message, ce dernier n'est pas renvoyé. Même si le client reste actif, le message n'est pas renvoyé. Le comportement des messages `QoS=0` est identique à celui des messages non persistants rapides IBM MQ .

   Si un message est publié par un client avec une qualité de service `QoS` de 1 ou 2, il devient persistant. Le message est stocké localement et n'est supprimé du client que lorsqu'il n'est plus nécessaire de garantir la distribution "au moins une fois", `QoS=1`ou "une seule fois", `QoS=2`.
2. Si un message est marqué comme `QoS` 1 ou 2, il est mis en file d'attente dans IBM MQ en tant que message persistant. S'il est marqué comme `QoS=0`, il est mis en file d'attente dans IBM MQ en tant que message non persistant. Dans IBM MQ , les messages non persistants sont transférés entre les gestionnaires de files d'attente "une seule fois", sauf si l'attribut `NPMSPEED` du canal de message est défini sur FAST.

Une publication persistante est stockée sur le client jusqu'à ce qu'elle soit reçue par une application client. Pour `QoS=2`, la publication est supprimée du client lorsque le rappel de l'application renvoie le contrôle. Pour `QoS=1` l'application peut recevoir une nouvelle fois la publication si un incident se produit. Pour `QoS=0`, le rappel reçoit la publication une seule fois. Il est possible qu'il ne reçoive pas de publication en cas d'incident ou si le client est déconnecté au moment de la publication.

Lorsque vous vous abonnez à une rubrique, vous pouvez diminuer la qualité de service QoS avec laquelle l'abonné reçoit les messages afin de correspondre à ses aptitudes à la persistance. Les publications créées avec une QoS supérieure sont envoyées avec la QoS la plus élevée demandée par l'abonné.

Stockage de messages

L'implémentation de stockage de données sur des dispositifs de petites taille varie considérablement. Le modèle d'enregistrement temporaire des messages persistants dans le stockage géré par le client MQTT peut être trop lent ou nécessiter trop de stockage. Sur les périphériques mobiles, le système d'exploitation mobile peut fournir un service de stockage idéal pour les messages MQTT .

Pour vous permettre de répondre aux contraintes des petits périphériques, le client MQTT dispose de deux interfaces de persistance. Ces interfaces définissent les opérations impliquées dans le stockage de messages persistants. Ces interfaces sont décrites dans la documentation d'API du MQTT client for Java. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference. Vous pouvez implémenter les interfaces afin de les adapter au dispositif. Le client MQTT qui s'exécute sur Java SE possède une implémentation par défaut des interfaces qui stockent les messages persistants dans le système de fichiers. Il utilise le package `java.io`.

Classes de persistance

`MqttClientPersistence`
: Envoyez une instance de l'implémentation de `MqttClientPersistence` au client MQTT sous la forme d'un paramètre du constructeur `MqttClient`. Si vous omettez le paramètre `MqttClientPersistence` dans le constructeur `MqttClient` , le client MQTT stocke les messages persistants à l'aide de la classe `MqttDefaultFilePersistence`.

`MqttPersistable`
: `MqttClientPersistence` extrait et insère les objets `MqttPersistable` à l'aide d'une clé de protection. Vous devez fournir une implémentation de `MqttPersistable` ainsi que l'implémentation de `MqttClientPersistence` si vous n'utilisez pas `MqttDefaultFilePersistence`.

`MqttDefaultFilePersistence`

Le client MQTT fournit la classe `MqttDefaultFilePersistence`. Si vous instanciez `MqttDefaultFilePersistence` dans votre application client, vous pouvez fournir le répertoire dans lequel stocker les messages persistants en tant que paramètre du constructeur `MqttDefaultFilePersistence` .

Le client MQTT peut également instancier `MqttDefaultFilePersistence` et placer les fichiers dans le répertoire par défaut suivant:

`client identifier` `-tcp` *`hostname portnumber`*

Les caractères suivants sont supprimés de la chaîne de nom de répertoire:

`"\", "\\", "/", ":"` et `" "`

Le chemin d'accès au répertoire est la valeur de la propriété système `rcp.data`; si `rcp.data` n'est pas défini, le chemin d'accès est la valeur de la propriété système `usr.data`, où

- `rcp.data` est une propriété associée à l'installation d'une initiative OSGi ou d'une application RCP (Rich Client Platform) Eclipse.
- `usr.data` est le répertoire dans lequel la commande Java qui a démarré l'application a été lancée.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Un `MqttMessage` comporte un tableau d'octets en tant que charge. Faites en sorte que la taille des messages soit la plus petite possible. La longueur maximale de message autorisée par MQTT protocol est de 250 Mo.

Généralement, un programme client MQTT utilise `java.lang.String` ou `java.lang.StringBuffer` pour manipuler le contenu des messages. Par commodité, la classe `MqttMessage` comprend une

méthode `toString` pour convertir sa charge en chaîne. Pour créer une charge de tableau d'octets à partir de `java.lang.String` ou de `java.lang.StringBuffer`, utilisez la méthode `getBytes`.

La méthode `getBytes` convertir une chaîne en jeu de caractères par défaut pour la plateforme. Le jeu de caractères par défaut utilisé est généralement `UTF-8`. Les publications MQTT qui ne contiennent que du texte sont généralement codées en `UTF-8`. Utilisez la méthode `getBytes("UTF8")` pour remplacer le jeu de caractères par défaut.

Dans IBM MQ, une publication MQTT est reçue en tant que message `jms-bytes`. Le message inclut un dossier `MQRFH2` contenant un dossier `<mqtt>`et un dossier `<mqps>` . Le dossier `<mqtt>` contient `clientId`, `msgId`et `qos`, mais ce contenu peut changer ultérieurement.

Une méthode `MqttMessage` comprend trois attributs supplémentaires : qualité de service, s'il est conservé et s'il est dupliqué. Un indicateur dupliqué est défini uniquement si la qualité de service est "au moins une fois" once" ou "une seule fois". Si le message a déjà été envoyé, et si le client MQTT n'en accuse par réception assez rapidement, il est renvoyé avec la valeur `true` pour l'attribut duplicate.

Publication

Pour créer une publication dans une application client MQTT , créez une `MqttMessage`. Définissez son contenu, la qualité du service et son éventuelle conservation, puis appelez la méthode `MqttTopic.publish(MqttMessage message)`. `MqttDeliveryToken` est renvoyé et la fin de la publication est asynchrone.

Une autre solution consiste à faire créer par le client MQTT un objet message temporaire à partir des paramètres de la méthode `MqttTopic.publish(byte [] payload, int qos, boolean retained)` lorsqu'il crée une publication.

Si la publication est associée à la qualité de service "au moins une fois" ou "une seule fois", `QoS=1` ou `QoS=2`, le client MQTT appelle l'interface `MqttClientPersistence`. Il appelle `MqttClientPersistence` pour stocker le message avant de renvoyer un jeton de distribution à l'application.

L'application peut choisir de se bloquer jusqu'à la distribution du message au serveur, à l'aide de la méthode `MqttDeliveryToken.waitForCompletion`. L'application peut également continuer sans blocage. Si vous voulez vérifier la distribution effective des publications sans créer de blocage, enregistrez une instance d'une classe de rappel implémentant `MqttCallback` auprès du client MQTT. Le client MQTT appelle la méthode `MqttCallback.deliveryComplete` dès que la publication a été distribuée. En fonction de la qualité de service, la distribution peut être presque immédiate pour `QoS=0` ou peut prendre un certain temps pour `QoS=2`.

Utilisez la méthode `MqttDeliveryToken.isComplete` pour vérifier si la distribution est terminée. Alors que la valeur de `MqttDeliveryToken.isComplete` est `false`, vous pouvez appeler `MqttDeliveryToken.getMessage` pour obtenir le contenu du message. Si le résultat de l'appel de `MqttDeliveryToken.isComplete` est `true`, le message a été supprimé et l'appel de `MqttDeliveryToken.getMessage` a envoyé une exception de pointeur Null. Il n'existe pas de synchronisation intégrée entre `MqttDeliveryToken.getMessage` et `MqttDeliveryToken.isComplete`.

Si le client se déconnecte avant de recevoir tous les jetons de distribution en attente, une nouvelle instance du client peut effectuer une requête pour les jetons de distribution en attente avant la connexion. Aucune nouvelle distribution n'est effectuée jusqu'à ce que le client se connecte et il est plus sûr d'appeler la méthode `MqttDeliveryToken.getMessage`. Utilisez la méthode `MqttDeliveryToken.getMessage` pour trouver les publications qui n'ont pas été distribuées. Les jetons de distributions sont supprimés si vous vous connectez avec `MqttConnectOptions.cleanSession` défini à sa valeur par défaut `true`.

Abonnement

Un gestionnaire de files d'attente est chargé de créer des publications à envoyer à un abonné MQTT . Le gestionnaire de files d'attente vérifie si le filtre de sujet dans un abonnement créé par un client MQTT correspond à la chaîne de sujet dans une publication. La correspondance peut être exacte ou

comprendre des caractères génériques. Avant de transférer la publication à l'abonné par le gestionnaire de files d'attente, ce dernier vérifie les attributs associés à la publication. Il suit la procédure de recherche décrite à la rubrique Abonnement à l'aide d'une chaîne de sujet contenant des caractères génériques pour identifier si un objet sujet d'administration accorde à l'utilisateur le droit de s'abonner.

Lorsque le client MQTT reçoit une publication avec la qualité de service "au moins une fois", il appelle la méthode `MqttCallback.messageArrived` pour traiter la publication. Si la qualité de service de la publication est "une seule fois", `QoS=2`, le client MQTT appelle l'interface `MqttClientPersistence` pour stocker le message lorsqu'il est reçu. Il appelle la méthode `MqttCallback.messageArrived`.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au

filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Qualités de service fournies par un client MQTT

Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

La qualité de service d'une publication est un attribut de `MqttMessage`. Elle est définie par la méthode `MqttMessage.setQos`.

La méthode `MqttClient.subscribe` peut réduire la qualité de service appliquée aux publications envoyées à un client sur une rubrique. La qualité de service d'une publication transférée à un abonné peut être différente de la qualité de service de la publication. La plus faible des deux valeurs est utilisée pour transférer une publication.

Au plus une fois
`QoS=0`

Le message est distribué une fois tout au plus, ou n'est pas distribué du tout. Sa distribution via le réseau n'est pas accompagnée d'un accusé de réception.

Le message n'est pas stocké. Le message peut être perdu si le client est déconnecté ou si le serveur échoue.

`QoS=0` est le mode de transfert le plus rapide. Il est parfois appelé "autonome après diffusion".

Le MQTT protocol ne requiert pas que les serveurs transmettent les publications sur `QoS=0` à un client. Si le client est déconnecté au moment où le serveur reçoit la publication, cette dernière peut être supprimée en fonction du serveur. Le service de télémétrie (MQXR) ne supprime pas les messages envoyés avec `QoS=0`. Ils sont stockés en tant que messages non persistants et sont uniquement supprimés en cas d'arrêt du gestionnaire de files d'attente.

Au moins une fois
`QoS=1`

`QoS=1` est le mode de transfert par défaut.

Le message est toujours distribué au moins une fois. Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur `DUP` défini jusqu'à la réception de l'accusé de réception. Par conséquent, le destinataire peut être envoyé le même message plusieurs fois et peut le traiter plusieurs fois.

Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité.

Le message est supprimé du destinataire après son traitement. Si le destinataire est un courtier, le message est publié pour ses abonnés. Si le destinataire est un client, le message est distribué à l'application de l'abonné. Une fois que le message est supprimé, le destinataire envoie un accusé de réception à l'expéditeur.

Le message est supprimé de l'expéditeur une fois qu'il a reçu un accusé de réception de la part du destinataire.

Une seule fois
`QoS=2`

Le message est toujours distribué une seule fois.

Le message doit être stocké localement au niveau de l'expéditeur et du destinataire jusqu'à ce qu'il soit traité.

QoS=2 est le mode de transfert le plus sûr, mais le plus lent. Il faut au moins deux paires de transmissions entre l'expéditeur et le destinataire avant la suppression du message côté expéditeur. Le message peut être traité côté destinataire après la première transmission.

Dans la première paire de transmissions, l'expéditeur transmet le message et reçoit un accusé de réception du destinataire indiquant qu'il a stocké le message. Si l'expéditeur ne reçoit pas d'accusé de réception, le message est renvoyé avec l'indicateur DUP défini jusqu'à la réception de l'accusé de réception.

Dans la deuxième paire de transmissions, l'expéditeur indique au destinataire qu'il peut terminer le traitement du message, `"PUBREL"`. Si l'expéditeur ne reçoit pas d'accusé de réception pour le message `"PUBREL"`, le message `"PUBREL"` est envoyé à nouveau jusqu'à ce qu'un accusé de réception soit reçu. L'expéditeur supprime le message qu'il a sauvegardé lors de la réception de l'accusé de réception du message `"PUBREL"`.

Le destinataire peut traiter le message lors de la première ou de la seconde phase à condition de ne pas traiter à nouveau le message. Si le destinataire est un courtier, il publie le message pour les abonnés. Si le destinataire est un client, il livre le message à l'application de l'abonné. Le destinataire renvoie un message d'achèvement à l'expéditeur indiquant qu'il a terminé le traitement du message.

Concepts associés

Rappels et synchronisation dans les applications client MQTT

Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres

Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client

L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)

Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT

Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Publications conservées et clients MQTT

Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Publications conservées et clients MQTT

Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Utilisez la méthode `MqttMessage.setRetained` pour indiquer si une publication sur une rubrique est conservée.

Lorsque vous créez ou mettez à jour une publication conservée, envoyez la publication avec un QoS de 1 ou 2. Si vous l'envoyez avec un QoS de 0, IBM MQ crée une publication conservée non permanente. La publication n'est pas conservée en cas d'arrêt du gestionnaire de files d'attente.

Si vous publiez une publication non conservée dans une rubrique ayant une publication conservée, cette dernière n'est pas affectée. Les abonnés actuels reçoivent la nouvelle publication. Les nouveaux abonnés reçoivent d'abord la publication conservée, puis les nouvelles publications.

Vous pouvez utiliser une publication conservée pour enregistrer la valeur la plus récente d'une mesure. Les nouveaux abonnés à une rubrique reçoivent immédiatement la valeur la plus récente de la mesure. Si aucune mesure n'a été prise depuis le dernier abonnement à la rubrique par l'abonné et que l'abonné s'abonne de nouveau, il reçoit de nouveau la dernière publication conservée dans la rubrique.

Pour supprimer une publication conservée, vous disposez de deux options:

- Exécutez la commande **CLEAR TOPICSTR** MQSC.
- Créez une publication conservée de longueur nulle. Comme indiqué dans la spécification MQTT 3.1.1 , si un message conservé de longueur nulle est publié dans une rubrique, tout message conservé pour cette rubrique est effacé.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des

caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Abonnements

Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Créez des abonnement à l'aide des méthodes `MqttClient.subscribe`, et transmettez un ou plusieurs filtres de rubrique et des paramètres de qualité de service. Le paramètre de qualité de service définit la qualité de service maximale à laquelle un abonné est préparé pour recevoir un message. Les messages envoyés à ce client ne peuvent pas être distribués avec la qualité de service la plus élevée. La qualité de service est définie au niveau le plus bas de la valeur d'origine lorsque le message a été publié et que le niveau a été spécifié pour l'abonnement. La qualité de service par défaut pour la réception de messages est `QoS=1`, au moins une fois.

La demande d'abonnement elle-même est envoyée avec la qualité de service `QoS=1`.

Les publications sont reçues par un abonné lorsque le client MQTT appelle la méthode `MqttCallback.messageArrived`. La méthode `messageArrived` transmet également la chaîne de sujet avec laquelle le message a été publié à l'abonné.

Vous pouvez supprimer l'abonnement ou un ensemble d'abonnements à l'aide des méthodes `MqttClient.unsubscribe`.

Une commande IBM MQ peut supprimer un abonnement. Répertorie les abonnements à l'aide de IBM MQ Explorer, ou à l'aide de commandes **runmqsc** ou PCF. Tous les abonnements client MQTT sont nommés. Ils reçoivent un nom au format suivant: *ClientIdentifier*:*Topic name*

Si vous utilisez la valeur par défaut `MqttConnectOptions`ou que vous définissez `MqttConnectOptions.cleanSession` sur `true` avant de connecter le client, les anciens abonnements du client sont supprimés lorsque le client se connecte. Les nouveaux abonnements effectués par le client lors de la session sont supprimés lorsque celui-ci se déconnecte.

Si vous définissez `MqttConnectOptions.cleanSession` sur `false` avant de se connecter, tous les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour le client avant sa connexion. Tous les abonnements restent actifs au moment de la déconnexion du client.

Un autre moyen de comprendre l'impact de l'attribut `cleanSession` sur les abonnements consiste à le considérer comme un attribut modal. Dans le cadre de son mode par défaut `cleanSession=true`, le client crée des abonnements et reçoit des publications uniquement dans la portée de la session. En mode alternatif, `cleanSession=false`, les abonnements sont durables. Le client peut se connecter et se déconnecter, et ses abonnements restent actifs. Lorsque le client se reconnecte, il reçoit les publications non distribuées. Lors de la connexion, il peut modifier l'ensemble des abonnements qui sont actifs en son nom.

Vous devez définir le mode `cleanSession` avant de vous connecter ; le mode dure toute la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez de l'utilisation de `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimés.

Les publications qui correspondent aux abonnements actifs sont envoyées au client dès leur publication. Si le client est déconnecté, elles sont envoyées au client si ce dernier se reconnecte au même serveur avec le même identificateur client et qu'il est attribué à `MqttConnectOptions.cleanSession` la valeur `false`.

Les abonnements pour un client particulier sont identifiés par l'identificateur client. Vous pouvez vous reconnecter au client à partir d'un dispositif client au même serveur et continuer avec les mêmes abonnements et recevoir des publications non distribuées.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation

des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT
Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Chaînes de rubrique et filtres de rubrique dans les clients MQTT

Les chaînes de sujet et les filtres sont utilisés pour la publication et l'abonnement. La syntaxe des chaînes et des filtre de rubrique dans les clients MQTT est globalement la même que celle des chaînes de rubrique dans IBM MQ.

Les chaînes de rubrique sont utilisées pour envoyer des publications aux abonnés. Créez une chaîne de rubrique à l'aide de la méthode `MqttClient.getTopic(java.lang.String topicString)`.

Les filtres de rubrique permettent de s'abonner à des rubriques et de recevoir des publications. Les filtres de rubrique peuvent contenir des caractères génériques. Ces derniers vous permettent de vous abonner à plusieurs rubriques. Créez un filtre à l'aide d'une méthode d'abonnement ; par exemple `MqttClient.subscribe(java.lang.String topicFilter)`.

Chaînes de rubrique

La syntaxe d'une chaîne de rubrique IBM MQ est décrite dans Chaînes de rubrique. La syntaxe des chaînes de rubrique MQTT est décrite dans la classe `MqttClient` de la documentation de l'API pour MQTT client for Java. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

La syntaxe de chaque type de chaîne de rubrique est presque identique. Il existe quatre différences mineures :

1. Les chaînes de rubrique envoyées à IBM MQ par les clients MQTT doivent suivre la convention de nom des gestionnaires de files d'attente.

2. La longueur maximale est différente. Les chaînes de rubrique IBM MQ sont limitées à 10 240 caractères. Un client MQTT peut créer des chaînes de rubrique pouvant atteindre 65535 octets.
3. Une chaîne de sujet créée par un client MQTT ne peut pas contenir de caractère null.
4. Dans IBM Integration Bus, un niveau de rubrique null, `'...//...'` , n'est pas valide. Les niveaux de rubrique Null sont pris en charge par IBM MQ.

Contrairement à la publication/l'abonnement IBM MQ, le protocole `mqttv3` n'utilise pas le concept d'objet de rubrique d'administration. Vous ne pouvez pas construire une chaîne de sujet à partir d'un objet de rubrique et d'une chaîne de sujet. Toutefois, une chaîne de rubrique est mappée à une rubrique d'administration dans IBM MQ. Le contrôle d'accès associé au sujet d'administration détermine si une publication est diffusée dans le sujet ou si elle et supprimée. Les attributs qui sont appliqués à la publication lorsqu'elle est transférée aux abonnés sont influencés par les attributs de sujet d'administration.

Filtres de rubrique

La syntaxe d'un filtre de rubrique IBM MQ est décrite dans la rubrique Schéma de caractères génériques basés sur des rubriques. La syntaxe des filtres de rubrique que vous pouvez construire avec un client MQTT est décrite dans la classe `MqttClient` de la documentation d'API de MQTT client for Java. Pour des liens vers la documentation de l'API client pour les bibliothèques client MQTT, voir MQTT client programming reference.

Concepts associés

Rappels et synchronisation dans les applications client MQTT
Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , dans la mesure du possible, des délais de transmission des messages vers et depuis le serveur. Les publications, les jetons de distribution et les événements de connexion perdue sont distribués aux méthodes dans une classe de rappel qui implémente `MqttCallback`.

Sessions propres
Le client MQTT et le service de télémétrie (MQXR) gèrent les informations d'état de session. Les informations d'état sont utilisées pour garantir la distribution "au moins une fois" et "une seule fois" et la réception "une seule fois" des publications. L'état de session inclut également les abonnements créés par un client MQTT . Vous pouvez choisir d'exécuter un client MQTT avec ou sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

Identificateur de client
L'identificateur de client est une chaîne de 23 octets qui identifie un client MQTT . Chaque identificateur doit être propre à un seul client connecté à la fois. L'identificateur doit contenir uniquement des caractères valides dans un nom de gestionnaire de files d'attente. Dans le cadre de ces contraintes, vous pouvez utiliser toute chaîne d'identification. Il est important de respecter une procédure d'allocation des identificateurs client et d'utiliser un mode de configuration du client avec l'identificateur choisi pour celui-ci.

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si une connexion client MQTT se termine de manière inattendue, vous pouvez configurer MQ Telemetry pour envoyer une publication "last will and testament". Définissez à l'avance le contenu de la publication et le sujet dans lequel l'envoyer. "Last will and testament" est une propriété de connexion. Créez-la avant de connecter le client.

Persistance des messages dans les clients MQTT
Les messages de publication deviennent persistants s'ils sont envoyés avec une qualité de service "au moins une fois" ou "une seule fois". Vous pouvez implémenter votre propre mécanisme de persistance sur le client ou utiliser le mécanisme de persistance par défaut fourni avec le client. La persistance fonctionne dans les deux directions pour les publications envoyées au client ou reçues par ce dernier.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Les clients MQTT peuvent créer des publications à envoyer à IBM MQet s'abonner à des rubriques sur IBM MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour la distribution des publications à IBM MQ et au client MQTT : "au plus une fois", "au moins une fois" et "une seule fois". Lorsqu'un client MQTT envoie une demande à IBM MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Une rubrique ne peut comporter qu'une seule publication conservée. Si vous créez un abonnement à une rubrique comportant une publication conservée, la publication vous est immédiatement transmise.

Abonnements
Créez des abonnements afin d'enregistrer des informations présentant un intérêt particulier dans des sujets de publication à l'aide d'un filtre. Un client peut créer plusieurs abonnements ou un abonnement contenant un filtre de sujets comprenant des caractères spéciaux, afin d'enregistrer une information présentant un intérêt particulier dans plusieurs sujets. Les publications des sujets correspondant au filtres sont envoyées au client. Les abonnements peuvent rester actifs même si un client est déconnecté. Les publications sont envoyées au client lorsqu'il se reconnecte.

Développement d'applications Microsoft Windows Communication Foundation avec IBM MQ

Le canal personnalisé Microsoft Windows Communication Foundation (WCF) pour IBM MQ envoie et reçoit des messages entre les clients et les services WCF.

Concepts associés

«Présentation du canal personnalisé IBM MQ pour WCF avec .NET», à la page 1299
Le canal personnalisé pour IBM MQ est un canal de transport utilisant le modèle de programmation unifié Microsoft Windows Communication Foundation (WCF).

«Utilisation des canaux personnalisés IBM MQ pour WCF», à la page 1304
Présentation des informations disponibles pour les programmeurs utilisant les canaux personnalisés IBM MQ pour Windows Communication Foundation (WCF).

«Utilisation des exemples WCF», à la page 1323
Les exemples Windows Communication Foundation (WCF) fournissent des exemples simples de la façon dont le canal personnalisé IBM MQ peut être utilisé.

FFST: WCF XMS First Failure Support Technology

Tâches associées

Traçage du canal personnalisé WCF pour IBM MQ

Traitement des incidents liés au canal personnalisé WCF pour IBM MQ

Présentation du canal personnalisé IBM MQ pour WCF avec .NET

Le canal personnalisé pour IBM MQ est un canal de transport utilisant le modèle de programmation unifié Microsoft Windows Communication Foundation (WCF).

L'infrastructure Microsoft Windows Communication Foundation, introduite dans Microsoft.NET 3, permet de développer des applications et des services .NET indépendamment du transport et des protocoles utilisés pour les connecter, ce qui permet d'utiliser d'autres transports ou configurations en fonction de l'environnement dans lequel le service ou l'application est déployé.

Les connexions sont gérées lors de l'exécution par WCF en créant une pile de canaux contenant la combinaison requise de:

- Eléments de protocole: ensemble facultatif d'éléments dans lequel aucun, un ou plusieurs éléments peuvent être ajoutés pour prendre en charge des protocoles tels que les normes WS-*.

- Codeur de message: élément obligatoire dans la pile contrôlant la sérialisation du message dans son format WF.
- Canal de transport: élément obligatoire dans la pile chargé du transport du message sérialisé vers son noeud final.

Le canal personnalisé pour IBM MQ est un canal de transport et, en tant que tel, il doit être associé à un encodeur de message et à des protocoles facultatifs requis par l'application à l'aide d'une liaison personnalisée WCF. De cette manière, les applications qui ont été développées pour utiliser WCF peuvent utiliser le canal personnalisé pour IBM MQ afin d'envoyer et de recevoir des données de la même manière qu'elles utilisent les transports intégrés fournis par Microsoft, ce qui permet une intégration simple aux fonctions de messagerie asynchrone, évolutive et fiable d' IBM MQ. Pour la liste complète des fonctions prises en charge, voir «Fonctions et capacités des canaux personnalisés WCF», à la page 1304.

Quand et pourquoi utiliser le canal personnalisé IBM MQ pour WCF?

Vous pouvez utiliser le canal personnalisé IBM MQ pour envoyer et recevoir des messages entre les clients et les services WCF de la même manière que les transports intégrés fournis par Microsoft, ce qui permet aux applications d'accéder aux fonctions de IBM MQ dans le modèle de programmation unifié WCF.

Un scénario de modèle d'utilisation typique pour le canal personnalisé IBM MQ pour WCF est une interface non SOAP pour la transmission de messages IBM MQ natifs.

Messages transmis à l'aide du format de message non-SOAP/non-JMS (Pure MQMessage)

Lorsque vous utilisez le canal personnalisé IBM MQ pour WCF en tant qu'interface non SOAP pour la transmission de messages IBM MQ natifs, les messages sont transmis à l'aide du format de message non SOAP/nonJMS (Pure MQMessage) de IBM MQ.

Les utilisateurs WCF peuvent démarrer le service ou, en d'autres termes, envoyer un message à une file d'attente IBM MQ à l'aide de MQMessages. Les applications peuvent obtenir et définir les zones et le contenu MQMD. Lorsque le message est disponible dans les files d'attente IBM MQ , il peut être traité par tout service WCF ou toute application non WCF, telle que les applications C ou Java qui s'exécutent sous AIX, Linux, Windowsou z/OS.

Configuration logicielle requise pour le canal personnalisé IBM MQ pour WCF

Cette rubrique décrit la configuration logicielle requise pour le canal personnalisé IBM MQ pour WCF. Le canal personnalisé IBM MQ pour WCF peut uniquement se connecter à des gestionnaires de files d'attente IBM WebSphere MQ 7.0 ou de niveau supérieur.

Conditions requises pour l'environnement d'exécution

- Microsoft.NET Framework v4.7.2 ou version ultérieure doit être installé sur la machine hôte.
- *Java and .NET Messaging and Web Services* est installé par défaut dans le cadre du programme d'installation de IBM MQ . Ce composant installe les assemblages .NET nécessaires pour le canal personnalisé dans le cache d'assemblage global.

Remarque : Si Microsoft .NET Framework V4.7.2 ou ultérieure n'est pas installé avant IBM MQ, l'installation du produit IBM MQ se poursuit sans erreur, mais le IBM MQ classes for .NET n'est pas disponible. Si.NET Framework est installé après l'installation de IBM MQ, les assemblages IBM MQ.NET doivent être enregistrés en exécutant le script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , où *WMQInstallDir* correspond au répertoire dans lequel IBM MQ est installé. Ce script installe les assemblages requis dans le cache d'assemblage global (GAC). Un ensemble de fichiers `amqi*.log` qui enregistrent les actions effectuées sont créés dans le répertoire `%TEMP%` . Il n'est pas nécessaire de réexécuter le script `amqiRegisterdotNet.cmd` si .NET est mis à niveau vers V4.7.2 ou une version ultérieure à partir d'une version antérieure, par exemple, à partir de .NET V3.5.

Conditions requises pour l'environnement de développement

- Microsoft Visual Studio 2015 ou Windows Software Development Kit for .NET 4.7.2 ou version ultérieure.
- Microsoft.NET Framework V4.7.2 ou version ultérieure doit être installé sur la machine hôte afin de générer les exemples de fichiers de solution.

IBM MQ custom channel for WCF: Qu'est-ce qui est installé?

Le canal personnalisé pour IBM MQ est un canal de transport utilisant le modèle de programmation unifié Microsoft Windows Communication Foundation (WCF). Le canal personnalisé est installé par défaut dans le cadre de l'installation.

Canal personnalisé IBM MQ pour WCF

Le canal personnalisé et ses dépendances sont contenus dans le composant `Java and .NET Messaging and Web Services` , qui est installé par défaut. Lors de la mise à niveau de IBM MQ à partir d'une version antérieure àIBM MQ 8.0, la mise à jour installe le canal personnalisé IBM MQ pour WCF par défaut si le composant `Java and .NET Messaging and Web Services` a été précédemment installé dans une installation antérieure.

Le composant `.NET Messaging and Web Services` contient le fichier `IBM.XMS.WCF.dll` et le fichier `IBM.WMQ.WCF.dll` , et ces fichiers constituent l'assemblage de canal personnalisé principal, qui contient les classes d'interface WCF. Ces fichiers sont installés dans le cache d'assemblage global (GAC) et sont également disponibles dans le répertoire suivant: *MQ_INSTALLATION_PATH* `\bin` où *MQ_INSTALLATION_PATH* est le répertoire dans lequel IBM MQ est installé.

Le tableau suivant récapitule les classes de clé requises pour l'utilisation du canal personnalisé.

Tableau 189. Classes de clé requises pour l'utilisation du canal personnalisé

	Interface SOAP/JMS (existante)	**Interface non SOAP/nonJMS (depuis IBM MQ 8.0)**
Assemblage de canal personnalisé	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Nom de la liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Importateur de liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Configuration de liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Exemples (Oneway)	SimpleOneWay_Client, SimpleOneWay_Service	MQMessaging_OneWay_Client, MQMessaging_OneWay_Service
Exemples (RequestReply)	SimpleRequestReply_Client, SimpleRequestReply_Service	MQMessaging_RequestReply_Client, MQMessaging_RequestReply_Service

IBM.WMQ.WCF.dll prend en charge les interfaces SOAP/JMS et non-SOAP/Non-JMS . Les nouvelles applications développées sont recommandées pour utiliser IBM.WMQ.WCF car il prend en charge les deux interfaces.

Envoi de messages formatés MQSTR

Si le message de demande est de type MQSTR, vous pouvez choisir d'envoyer le message de réponse au format MQSTR.

Vous devez utiliser un paramètre d'URI supplémentaire **replyMessageFormat** pour modifier le format du message de réponse. Les valeurs prises en charge sont les suivantes :

""

"" est la valeur par défaut.

Le message de réponse est au format octet (MQMFT_NONE). Exemple :

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t= "
```

MQSTR

Le message de réponse est au format MQSTR (MQMFT_STRING). Exemple :

```
"jms:/queue?
destination=SampleQ@QM1&connectionFactory=binding(server)connectQueueManager(QM1)
&initialContextFactory=com.ibm.mq.jms.Nojndi&replyDestination=SampleReplyQ&replyMessageForma
t=MQSTR"
```

Remarques :

1. La valeur de **replyMessageFormat** est insensible à la casse.
2. L'utilisation d'une valeur autre que *""* ou *MQSTR*entraîne une exception de valeur de paramètre non valide.

Exemples de canal personnalisé IBM MQ

Les exemples fournissent des exemples simples de la façon dont le canal personnalisé IBM MQ pour WCF peut être utilisé. Les exemples et leurs fichiers associés se trouvent dans le répertoire *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ. Pour plus d'informations sur les exemples de canal personnalisé IBM MQ , voir «Utilisation des exemples WCF», à la page 1323.

svcutil.exe.config

Le `svcutil.exe.config` est un exemple des paramètres de configuration requis pour permettre à l'outil de génération de proxy client Microsoft WCF `svcutil` de reconnaître le canal personnalisé. Le fichier `svcutil.exe.config` se trouve dans le répertoire *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ. Pour plus d'informations sur l'utilisation du `svcutil.exe.config`, voir «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec des métadonnées à partir d'un service en cours d'exécution», à la page 1321.

Architecture WCF

Le canal personnalisé IBM MQ pour WCF est intégré par-dessus l'API `IBM Message Service Client for .NET (XMS .NET)` .

Interface SOAP/JMS

L'architecture WCF est illustrée dans le diagramme suivant:

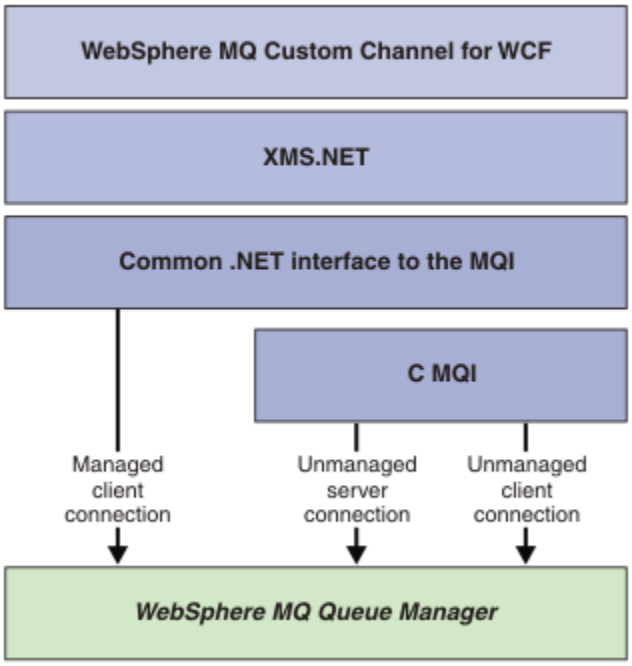


Figure 149. Architecture WCF pour l'interface SOAP/JMS

Tous les composants requis sont installés par défaut avec l'installation du produit.

Les trois connexions sont les suivantes:

- Connexions client gérées
- Connexions de serveur non gérées
- Connexions client non gérées

Pour plus d'informations sur ces connexions, voir «Options de connexion WCF», à la page 1310.

Interface non SOAP/non-JMS

Le canal personnalisé IBM MQ pour WCF prend en charge l'interface SOAP/JMS (disponible dans IBM WebSphere MQ 7.0.1) et l'interface non-SOAP/non-JMS .

L'architecture WCF est illustrée dans le diagramme suivant:

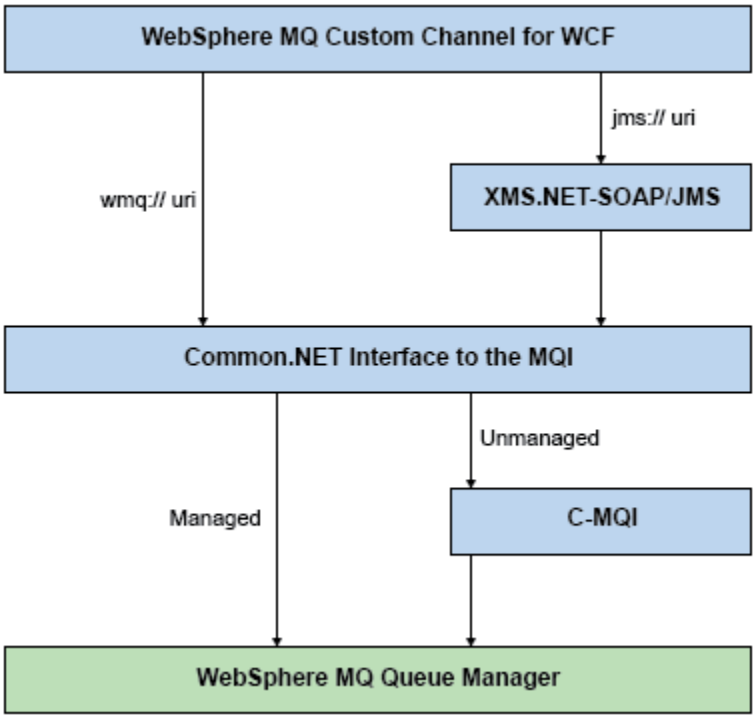


Figure 150. Architecture WCF pour l'interface non-SOAP/non-JMS

Utilisation des canaux personnalisés IBM MQ pour WCF

Présentation des informations disponibles pour les programmeurs utilisant les canaux personnalisés IBM MQ pour Windows Communication Foundation (WCF).

Microsoft Windows Communication Foundation sous-tend la prise en charge des services Web et de la messagerie dans Microsoft.NET Framework 3. IBM MQ peut être utilisé comme canal personnalisé dans WCF dans .NET Framework 3 de la même manière que les canaux intégrés proposés par Microsoft.

Les messages transportés via le canal personnalisé sont formatés en fonction de l'implémentation SOAP sur JMS de IBM MQ. Les applications peuvent ensuite communiquer avec les services hébergés par WCF ou par l'infrastructure de service WebSphere SOAP sur JMS .

Fonctions et capacités des canaux personnalisés WCF

Utilisez les rubriques suivantes pour obtenir des informations sur les fonctions et capacités des canaux personnalisés WCF.

Formes de canal personnalisé WCF

Présentation des formes de canal personnalisées que IBM MQ peut utiliser dans les canaux personnalisés Microsoft Windows Communication Foundation (WCF).

Le canal personnalisé IBM MQ pour WCF prend en charge deux formes de canal:

- Unidirectionnel
- Demande - Réponse

WCF sélectionne automatiquement la forme du canal en fonction du contrat de service hébergé.

Les contrats qui incluent des méthodes qui utilisent uniquement le paramètre **IsOneWay** sont traités par la forme de canal unidirectionnel, par exemple:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Les contrats qui incluent soit une combinaison de méthodes unidirectionnelles et de méthodes de réponse aux demandes, soit toutes les méthodes de réponse aux demandes, sont traités par la forme du canal de réponse aux demandes. Exemple :

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Remarque : Lorsque vous combinez des méthodes unidirectionnelles et des méthodes de demande-réponse dans le même contrat, vous devez vous assurer que le comportement est celui prévu, en particulier lorsque vous travaillez dans un environnement mixte, car les méthodes unidirectionnelles attendent qu'elles reçoivent une réponse null du service.

Canal unidirectionnel

Le canal personnalisé unidirectionnel IBM MQ pour WCF est utilisé, par exemple, pour envoyer des messages à partir d'un client WCF à l'aide d'une forme de canal unidirectionnel. Le canal peut envoyer des messages dans une seule direction, par exemple, à partir d'un gestionnaire de files d'attente client vers une file d'attente d'un service WCF.

Canal demande-réponse

Le canal personnalisé de demande-réponse IBM MQ pour WCF est utilisé, par exemple, pour envoyer des messages dans deux directions de manière asynchrone. La même instance client doit être utilisée pour la messagerie asynchrone. Le canal peut envoyer des messages dans un sens, par exemple, à partir d'un gestionnaire de files d'attente client vers une file d'attente sur un service WCF, puis envoyer un message de réponse à partir de WCF vers une file d'attente sur le gestionnaire de files d'attente client.

Noms et valeurs des paramètres d'URI WCF

Noms et valeurs des paramètres URI de l'interface SOAP/JMS et de l'interface non-SOAP/Non JMS .

Interface SOAP/JMS

connectionFactory
Le paramètre connectionFactory est obligatoire.

initialContextFactory
Le paramètre de fabrique initialContextest obligatoire et doit être défini sur "com.ibm.mq.jms.Nojndi" pour la compatibilité avec WebSphere Application Server et d'autres produits.

Interface non SOAP/non JMS

Le format de l'URI est celui des spécifications MA93 . Voir SupportPac - MA93 pour plus de détails sur les spécifications IRI IBM MQ .

IBM MQ Syntaxe de l'identificateur URI

```
wmq-iri = "wmq:" [ "//" connection-name ] "/" wmq-dest ["?" parm *("&" parm)]
connection-name = tcp-connection-name / other-connection-name
tcp-connection-name = ihost [ ":" port ]
other-connection-name = 1*(iunreserved / pct-encoded)
wmq-dest = queue-dest / topic-dest
queue-dest = "msg/queue/" wmq-queue ["@" wmq-qmgr]
```

```
wmq-queue = wmq-name
wmq-qmgr = wmq-name
wmq-name = 1*48( wmq-char )
topic-dest = "msg/topic/" wmq-topic
wmq-topic = segment *( "/" segment )
```

Exemple IRI IBM MQ

L'exemple IRI suivant indique à un demandeur de service qu'il peut utiliser une connexion de liaison de client IBM MQ TCP à une machine appelée example.com sur le port 1414 et insérer des messages de demande persistants dans une file d'attente appelée SampleQ sur le gestionnaire de files d'attente QM1. L'IRI indique que le fournisseur de services va placer les réponses dans une file d'attente appelée SampleReplyQ.

```
1)wmq://example.com:1414/msg/queue/SampleQ@QM1?
ReplyTo=SampleReplyQ&persistence=MQPER_NOT_PERSISTENT
2)wmq://localhost:1414/msg/queue/Q1?
connectQueueManager=QM1&replyTo=Q2&connectionmode=managed
```

Pour les connexions TLS activées

Pour établir des connexions sécurisées (TLS) à l'aide du client / service WCF, définissez les propriétés suivantes avec les valeurs appropriées dans l'URI. Toutes les propriétés avec le préfixe "*" sont obligatoires pour établir une connexion sécurisée.

- **sslKeyRepository**: *SYSTEM ou *USER
- * **sslCipherSpec**: un CipherSpecvalide, par exemple TLS_RSA_WITH_AES_128_CBC_SHA256.
- **sslCertRevocationCheck**: true ou false.
- **sslKeyResetCount**: valeur supérieure à 32kb.
- **sslPeerName:** : nom distinctif du certificat serveur

Exemple :

```
"wmq://localhost:1414/msg/queue/SampleQ?
connectQueueManager=QM1&sslkeyrepository=*SYSTEM&sslcipherspec=
TLS_RSA_WITH_AES_128_CBC_SHA&sslcertrevocationcheck=true&"sslpe
ername=" + "" + "CN=ibmwebspheremqqmm&sslkeyresetcount=45000"
```

Distribution garantie de canal personnalisé WCF

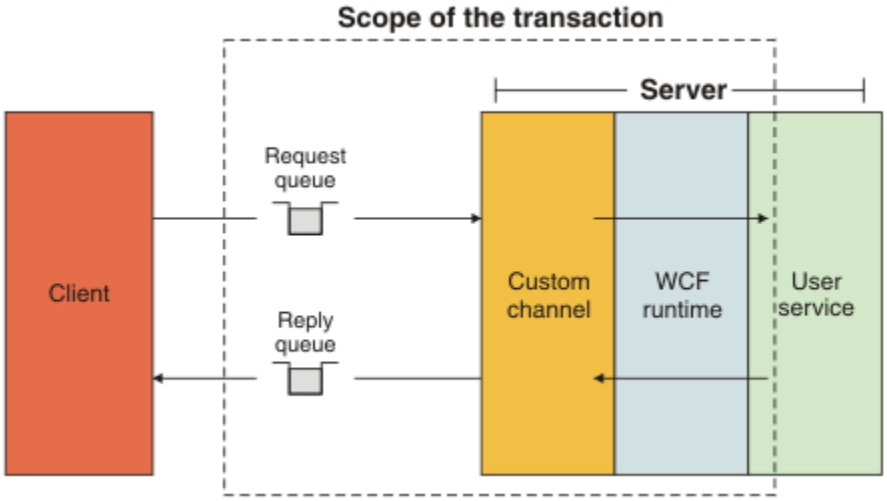
La livraison assurée garantit qu'une demande de service ou une réponse est activée et qu'elle n'est pas perdue.

Un message de demande est reçu et tout message de réponse est envoyé sous un point de synchronisation de transaction locale, qui peut être annulé en cas d'échec de l'exécution. Exemples de ces échecs: une exception non gérée émise par le service, l'échec de la distribution du message au service ou l'échec de la distribution du message de réponse.

`AssuredDelivery` est l'attribut de distribution assurée qui peut être spécifié dans un contrat de service pour garantir que les messages de demande reçus par un service et les messages de réponse envoyés par un service ne sont pas perdus en cas d'échec d'exécution.

Pour vous assurer que les messages sont également conservés en cas de panne du système ou de panne de courant, les messages doivent être envoyés en tant que messages persistants. Pour utiliser des messages persistants, cette option doit être spécifiée dans l'URI de noeud final de l'application client.

Les transactions réparties ne sont pas prises en charge et la portée de la transaction ne s'étend pas au-delà du traitement des messages de demande et de réponse effectué par IBM MQ. Tout travail effectué dans le service peut être réexécuté suite à un incident qui entraîne la réception du message à nouveau. Le diagramme suivant illustre la portée de la transaction:



La distribution assurée est activée en appliquant l'attribut `AssuredDelivery` à la classe de service, comme illustré dans l'exemple suivant:

```
[AssuredDelivery]
class TestCalculatorService : IWMQSampleCalculatorContract
{
    public int add(int a, int b)
    {
        int ans = a + b;
        return ans;
    }
}
```

Lorsque vous utilisez l'attribut `AssuredDelivery` , vous devez tenir compte des points suivants:

- Lorsqu'un canal détermine qu'un échec est susceptible de se reproduire si un message a été annulé et reçu à nouveau, le message est traité comme un message incohérent et n'est pas renvoyé à la file d'attente des demandes en vue de son retraitement. Par exemple: si le message reçu n'est pas correctement formaté ou ne peut pas être envoyé à un service. Les exceptions non traitées émises à partir d'une opération de service sont toujours renvoyées jusqu'à ce que le message ait été redistribué le nombre maximal de fois spécifié par la propriété de seuil d'annulation de la file d'attente des demandes. Pour plus d'informations, voir «Messages incohérents de canal personnalisé WCF», à la page 1308
- Le canal effectue la lecture, le traitement et la réponse de chaque message de demande en tant qu'opération atomique à l'aide d'une seule unité d'exécution pour appliquer l'intégrité transactionnelle. Pour permettre aux opérations de service de s'exécuter simultanément, le canal permet à WCF de créer plusieurs instances du canal. Le nombre d'instances de canal disponibles pour le traitement des demandes est contrôlé par la propriété de liaison `MaxConcurrentCalls`. Pour plus d'informations, voir «Options de configuration de liaison WCF», à la page 1316
- La fonction de distribution assurée utilise à la fois les points d'extensibilité WCF IOperationInvoker et IErrorHandler . Si ces points d'extensibilité sont utilisés en externe par une application, celle-ci doit s'assurer que tous les points d'extensibilité précédemment enregistrés sont appelés. Si vous ne le faites pas pour IErrorHandler , des erreurs peuvent être signalées. L'échec de cette opération pour IOperationInvoker peut entraîner l'arrêt de la réponse de WCF.

Sécurité de canal personnalisé WCF

Le canal personnalisé IBM MQ pour WCF prend en charge l'utilisation de TLS uniquement pour les connexions client non gérées au gestionnaire de files d'attente.

Spécifiez TLS à l'aide d'une entrée dans la table de définition de canal du client (CCDT). Pour plus d'informations sur les tables de définition de canal du client, voir Table de définition de canal du client.

Tables de définition de canal du client WCF (CCDT)

Le canal personnalisé IBM MQ pour WCF prend en charge l'utilisation des tables de définition de canal du client (CCDT) pour configurer les informations de connexion pour les connexions client.

Les CDTc sont contrôlées à l'aide de ces deux variables d'environnement:

- *MQCHLLIB* indique le répertoire dans lequel se trouve la table.
- *MQCHLTAB* indique le nom de fichier de la table.

Si ces variables d'environnement sont définies, elles sont prioritaires sur les détails de connexion client spécifiés dans l'URI.

Pour plus d'informations sur les tables de définition de canal du client, voir Table de définition de canal du client.

Messages incohérents de canal personnalisé WCF

Lorsqu'un service ne parvient pas à traiter un message de demande ou à distribuer un message de réponse dans une file d'attente de réponses, le message est traité comme un message incohérent.

Messages de demande incohérents

Si un message de demande ne peut pas être traité, il est traité comme un message incohérent. Cette action empêche le service de recevoir à nouveau le même message intraitable. Pour qu'un message de demande intraitable soit traité comme un message incohérent, l'une des situations suivantes doit être vraie:

- Le nombre d'annulations de messages a dépassé le seuil d'annulation indiqué dans la file d'attente des demandes, qui se produit uniquement si une distribution assurée a été spécifiée pour le service. Pour plus d'informations sur la distribution assurée, voir: «Distribution garantie de canal personnalisé WCF», à la page 1306
- Le message n'a pas été formaté correctement et n'a pas pu être interprété comme un message SOAP sur JMS .

Messages de réponse incohérents

Si un service ne parvient pas à distribuer un message de réponse dans la file d'attente des réponses, le message de réponse est traité comme un message incohérent. Pour les messages de réponse, cette action permet d'extraire les messages de réponse ultérieurement pour faciliter l'identification des incidents.

Traitement des messages incohérents

L'action effectuée pour un message incohérent dépend de la configuration du gestionnaire de files d'attente et des valeurs définies dans les options de rapport du message. Pour SOAP sur JMS, les options de rapport suivantes sont définies par défaut sur les messages de demande et ne sont pas configurables:

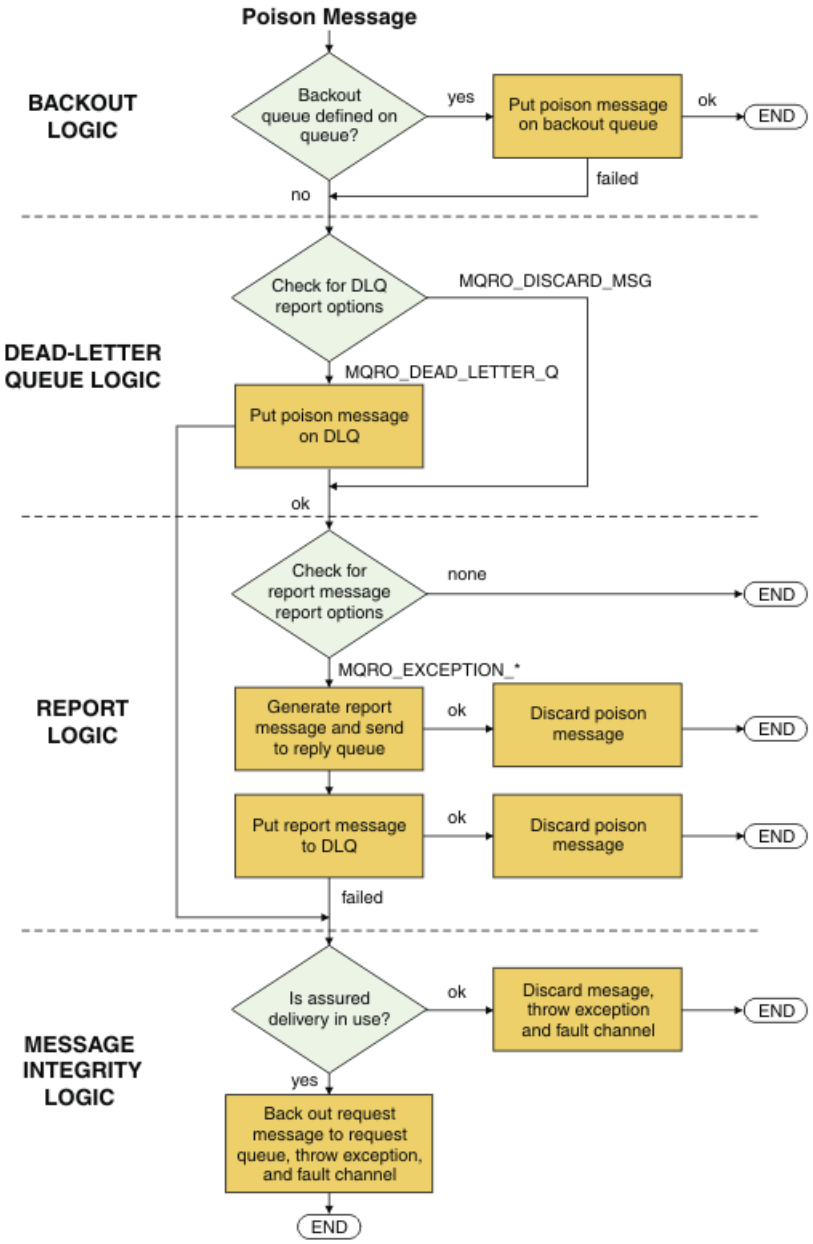
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

Pour SOAP sur JMS, l'option de rapport suivante est définie par défaut sur les messages de réponse et n'est pas configurable:

- MQRO_DEAD_LETTER_Q

Si les messages proviennent d'une source non WCF, consultez la documentation de cette source.

Le diagramme suivant illustre les actions possibles et les étapes à suivre en cas d'échec du traitement des messages incohérents:



Fonctions de message IBM MQ pour les applications WCF

Non-SOAP/Non-JMS (c'est-à-dire, IBM MQ) fonctions de message pour les applications WCF.

Pour l'interface non-SOAP/non-JMS , les fonctions de message IBM MQ pour les applications WCF sont les suivantes:

- Les applications WCF peuvent envoyer et recevoir les messages IBM MQ de base qui peuvent être traités par n'importe quelle application IBM MQ .
- Les applications WCF disposent d'un contrôle total pour mettre à jour le MQMD et le contenu.
- Le client WCF peut envoyer des messages IBM MQ qui peuvent être consommés par n'importe quel client IBM MQ , par exemple les clients C, Java, JMSet .NET .

L'interface WCF for Non-SOAP/Non-JMS doit utiliser les classes suivantes pour définir la charge de message et MQMD pour le message:

- WmqStringMessage pour un contenu de type String
- WmqBytesMessage pour un contenu de type Bytes
- WmqXmlMessage pour un contenu de type XML

Pour définir la charge du message, utilisez la propriété **Data** pour le message WmqString, le message WmqBytesou la classe de message WmqXml, en fonction du type de charge. Par exemple, utilisez le code suivant pour définir un contenu de type String:

```
WmqStringMessage strMsg = new WmqStringMessage();
//Setting the Message PayLoad
strMsg.Data = "Hello World";
//MQMD property
strMsg.Format = WmqMessageFormat.MQFMT_STRING;
```

Options de connexion WCF

Il existe trois modes de connexion d'un canal personnalisé IBM MQ pour WCF à un gestionnaire de files d'attente. Déterminez le type de connexion qui convient le mieux à vos besoins.

Pour plus d'informations sur les options de connexion, voir: «Différences de connexion», à la page 593

Pour plus d'informations sur l'architecture WCF, voir «Architecture WCF», à la page 1302

Connexion client non gérée

Une connexion établie dans ce mode se connecte en tant que client IBM MQ à un serveur IBM MQ s'exécutant sur la machine locale ou sur une machine distante.

Pour utiliser le canal personnalisé IBM MQ pour WCF en tant que client IBM MQ , vous pouvez l'installer avec IBM MQ MQI clientsur le serveur IBM MQ ou sur une machine distincte.

Connexion à un serveur non géré

Lorsqu'il est utilisé en mode liaisons de serveur, le canal personnalisé IBM MQ pour WCF utilise l'API du gestionnaire de files d'attente au lieu de communiquer via un réseau. L'utilisation de connexions de liaisons offre de meilleures performances pour les applications IBM MQ que l'utilisation de connexions réseau.

Pour utiliser la connexion de liaisons, vous devez installer le canal personnalisé IBM MQ pour WCF sur le serveur IBM MQ .

Connexion client gérée

Une connexion établie dans ce mode se connecte en tant que client IBM MQ à un serveur IBM MQ s'exécutant sur la machine locale ou sur une machine distante.

Les classes de canal personnalisées IBM MQ pour .NET 3 qui se connectent dans ce mode restent dans le code géré .NET et n'effectuent aucun appel aux services natifs. Pour plus d'informations sur le code géré, voir la documentation Microsoft .

L'utilisation du client géré est soumise à un certain nombre de limitations. Pour plus d'informations sur ces limitations, voir «Connexions client gérées», à la page 593.

Création et configuration du canal personnalisé IBM MQ pour WCF

Les canaux personnalisés IBM MQ pour WCF fonctionnent de la même manière que les canaux WCF de transport offerts par Microsoft. Le canal personnalisé IBM MQ pour WCF peut être créé de l'une des deux manières suivantes.

Pourquoi et quand exécuter cette tâche

Le canal personnalisé IBM MQ s'intègre à WCF en tant que canal de transport WCF et, en tant que tel, doit être associé à un encodeur de message et à des canaux de protocole facultatifs, afin de pouvoir créer une pile de canaux complète pouvant être utilisée par une application. Deux éléments sont requis pour que la création d'une pile de canaux complète aboutisse:

1. Une définition de liaison: indique les éléments requis pour générer la pile de canaux d'applications, y compris le canal de transport, l'encodeur de message et tous les protocoles, ainsi que les paramètres de configuration généraux. Pour le canal personnalisé, la définition de liaison doit être créée sous la forme d'une liaison personnalisée WCF.
2. Une définition de noeud final: lie le contrat de service à la définition de liaison et fournit également l'URI de connexion réel qui décrit où l'application peut se connecter. Pour le canal personnalisé, l'URI se présente sous la forme d'un URI SOAP sur JMS .

Ces définitions peuvent être créées de l'une des deux manières suivantes:

- Administrativement, les définitions sont créées en fournissant les détails dans un fichier de configuration d'application (par exemple: `app.config`).
- A l'aide d'un programme ; les définitions sont créées directement à partir du code de l'application.

La décision quant à la méthode à utiliser pour créer les définitions doit être basée sur les exigences de l'application comme suit:

- La méthode d'administration de la configuration offre la souplesse nécessaire pour modifier les détails du post-déploiement du service et du client sans régénérer l'application.
- La méthode de programmation pour la configuration offre une meilleure protection contre les erreurs de configuration et la possibilité de générer dynamiquement une configuration lors de l'exécution.

Création d'un canal personnalisé WCF de manière administrative en fournissant des informations de liaison et de noeud final dans un fichier de configuration d'application

Le canal personnalisé IBM MQ pour WCF est un canal WCF de niveau transport. Un noeud final et une liaison doivent être définis pour utiliser le canal personnalisé, et ces définitions peuvent être effectuées en fournissant les informations de liaison et de noeud final dans un fichier de configuration d'application.

Pour configurer et utiliser le canal personnalisé IBM MQ pour WCF, qui est un canal WCF de niveau transport, une liaison et une définition de noeud final doivent être définies. La liaison contient les informations de configuration du canal et la définition de noeud final contient les détails de connexion. Ces définitions peuvent être créées de deux manières:

- A l'aide d'un programme directement à partir du code de l'application, comme décrit ici: «Création d'un canal personnalisé WCF en fournissant des informations de liaison et de noeud final à l'aide d'un programme», à la page 1313
- Administrativement, en fournissant les détails dans un fichier de configuration d'application, comme décrit dans la procédure suivante.

Le fichier de configuration de l'application client ou de service est généralement nommé *yourappname*`.exe.config` , où *nom_de_votre_application* est le nom de votre application. Le fichier de configuration d'application est le plus facilement modifié à l'aide de l'éditeur de configuration de service Microsoft appelé `SvcConfigEditor.exe` de la manière suivante:

- Démarrez l'éditeur de configuration `SvcConfigEditor.exe` . L'emplacement d'installation par défaut de l'outil est: *`Drive:`*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` où *Unité:* est le nom de l'unité d'installation.

Etape 1: Ajoutez une extension d'élément de liaison pour permettre à WCF de localiser le canal personnalisé

1. Cliquez avec le bouton droit de la souris sur **Avancé** > **Extension** > **élément de liaison** pour ouvrir le menu et sélectionnez **Nouveau** .
2. Renseignez les zones comme indiqué dans le tableau suivant:

Tableau 190. Nouvelles zones d'élément de liaison

Zone	**Valeur**
Nom	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Tableau 190. Nouvelles zones d'élément de liaison (suite)	
Zone	**Valeur**
Type	Accédez à `IBM.XMS.WCF.dll` dans le cache d'assemblage global (GAC) et sélectionnez `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

Etape 2: Créez une définition de liaison personnalisée qui associe le canal personnalisé à un encodeur de message WCF

1. Cliquez avec le bouton droit de la souris sur **Liaisons** pour ouvrir le menu et sélectionnez **Nouvelle configuration de liaison** .
2. Renseignez les zones comme indiqué dans le tableau suivant:

Tableau 191. Nouvelles zones de configuration de liaison	
Zone	**Valeur**
Nom	`CustomBinding_WMQ`
BindingElement 1	`textMessageEncoding (MessageVersion: Soap11)`
BindingElement 2	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`

Etape 3: Spécifiez les propriétés de liaison

1. Sélectionnez *IBM.XMS.WCF.SoapJmsIbmTransportChannel* à partir de la liaison que vous avez créée dans: «Etape 2: Créez une définition de liaison personnalisée qui associe le canal personnalisé à un encodeur de message WCF», à la page 1312
2. Apportez les modifications requises aux valeurs par défaut des propriétés, comme décrit dans: «Options de configuration de liaison WCF», à la page 1316

Etape 4: Création d'une définition de noeud final

Créez une définition de noeud final qui fait référence à la liaison personnalisée que vous avez créée dans: «Etape 2: Créez une définition de liaison personnalisée qui associe le canal personnalisé à un encodeur de message WCF», à la page 1312 et qui fournit les détails de connexion du service. La manière dont ces informations sont spécifiées varie selon que la définition concerne une application client ou une application de service.

Pour une application client, ajoutez une définition de noeud final à la section client comme suit:

1. Cliquez avec le bouton droit de la souris sur **Client** > **Noeuds finaux** pour ouvrir le menu et sélectionnez **Nouveau noeud final client** .
2. Renseignez les zones comme indiqué dans le tableau suivant:

Tableau 192. Nouvelles zones de noeud final client	
Zone	**Valeur**
Nom	`Endpoint_WMQ`
Address	*URI SOAP/JMS décrivant les détails de connexion WMQ requis pour accéder au service.* Pour plus de détails, voir: «Canal personnalisé IBM MQ pour le format d'adresse URI de noeud final WCF», à la page 1315
Liaison en cours	`customBinding`

Tableau 192. Nouvelles zones de noeud final client (suite)	
Zone	**Valeur**
BindingConfiguration	`CustomBinding_WMQ`
Contrat	*Nom de votre interface de contrat de service*

Pour une application de service, ajoutez une définition de service à la section des services comme suit:

1. Cliquez avec le bouton droit de la souris sur **Services** pour ouvrir le menu, puis sélectionnez **Nouveau service**, puis sélectionnez la classe de service à héberger.
2. Ajoutez une définition de noeud final à la section **Noeuds finaux** pour votre nouveau service et renseignez les zones comme indiqué dans le tableau suivant:

Tableau 193. Nouvelles zones de noeud final de service	
Zone	**Valeur**
Nom	`Endpoint_WMQ`
Address	*URI SOAP/JMS décrivant les détails de connexion WMQ requis pour accéder au service.* Pour plus de détails, voir: «Canal personnalisé IBM MQ pour le format d'adresse URI de noeud final WCF», à la page 1315
Liaison en cours	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contrat	*Nom de votre classe d'implémentation de service*

Création d'un canal personnalisé WCF en fournissant des informations de liaison et de noeud final à l'aide d'un programme

Le canal personnalisé IBM MQ pour WCF est un canal WCF de niveau transport. Un noeud final et une liaison doivent être définis pour utiliser le canal personnalisé et ces définitions peuvent être effectuées à l'aide d'un programme directement à partir du code de l'application.

Pour configurer et utiliser le canal personnalisé IBM MQ pour WCF, qui est un canal WCF de niveau transport, une liaison et une définition de noeud final doivent être définies. La liaison contient les informations de configuration du canal et la définition de noeud final contient les détails de connexion. Pour plus d'informations, voir «Utilisation des exemples WCF», à la page 1323.

Ces définitions peuvent être créées de deux manières:

- Administrativement, en fournissant les détails dans un fichier de configuration d'application, comme décrit dans «Création d'un canal personnalisé WCF de manière administrative en fournissant des informations de liaison et de noeud final dans un fichier de configuration d'application», à la page 1311.
- A l'aide d'un programme, directement à partir du code de l'application, comme décrit dans les sous-rubriques suivantes.

Définition des informations de liaison et de noeud final à l'aide d'un programme: interface SOAP/JMS

Pour l'interface SOAP/JMS , vous pouvez définir un noeud final et une liaison à l'aide d'un programme directement à partir du code d'application.

Pourquoi et quand exécuter cette tâche

Pour fournir des informations de liaison et de noeud final à l'aide d'un programme, ajoutez le code requis à votre application en procédant comme suit.

Procédure

1. Créez une instance de l'élément de liaison de transport du canal en ajoutant le code suivant à votre application:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

2. Définissez les propriétés de liaison requises, par exemple, en ajoutant le code suivant à votre application pour définir le `ClientConnectionMode`:

```
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
```

3. Créez une liaison personnalisée qui associe le canal de transport à un encodeur de message en ajoutant le code suivant à votre application:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

4. Créez l'URI SOAP/JMS .

   L'URI SOAP/JMS qui décrit les détails de connexion IBM MQ requis pour accéder au service doit être fourni en tant qu'adresse de noeud final. L'adresse que vous spécifiez varie selon que le canal est utilisé pour une application de service ou une application client.

   - Pour les applications client, l'URI SOAP/JMS doit être créé en tant que `EndpointAddress` comme suit:

     ```
     EndpointAddress address = new EndpointAddress("jms:/queue?
     destination=SampleQ@QM1&connectionFactory
     =connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

   - Pour les applications de service, l'URI SOAP/JMS doit être créé en tant qu'URI comme suit:

     ```
     Uri address = new Uri("jms:/queue?destination=SampleQ@QM1&connectionFactory=
     connectQueueManager(QM1)&initialContextFactory=com.ibm.mq.jms.Nojndi");
     ```

     Pour plus d'informations sur les adresses de noeud final, voir «Canal personnalisé IBM MQ pour le format d'adresse URI de noeud final WCF», à la page 1315.

Définition des informations de liaison et de noeud final à l'aide d'un programme: interface non-SOAP/ non-JMS

Pour l'interface non-SOAP/Non-JMS , vous pouvez définir un noeud final et une liaison à l'aide d'un programme directement à partir du code de l'application.

Pourquoi et quand exécuter cette tâche

Pour fournir des informations de liaison et de noeud final à l'aide d'un programme, ajoutez le code requis à votre application en procédant comme suit.

Procédure

1. Créez une liaison WmqBinding en ajoutant le code suivant à votre application:

```
WmqBinding binding = new WmqBinding();
```

   Ce code crée une liaison qui associe l'élément WmqMsgEncodingElement et l'élément WmqIbmTransportBindingrequis pour l'interface non-SOAP/Non-JMS .

2. Indiquez l'URI wmq: // qui décrit les détails de connexion IBM MQ requis pour accéder au service.

La manière dont vous fournissez l'URI wmq: // varie selon que le canal est utilisé pour une application de service ou une application client.

- Pour les applications client, l'URI wmq: // doit être créé en tant qu' EndpointAddress comme suit:

```
EndpointAddress address = new EndpointAddress
("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

- Pour les applications de service, l'URI wmq: // doit être créé en tant qu'URI comme suit:

```
Uri sampleAddress = new Uri(
"wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
```

Canal personnalisé IBM MQ pour le format d'adresse URI de noeud final WCF

Un service Web est spécifié à l'aide d'un identificateur URI (Universal Resource Identifier) qui fournit des détails d'emplacement et de connexion. Le format d'URI varie selon que vous utilisez l'interface SOAP/JMS ou l'interface non-SOAP/Non-JMS .

Interface SOAP/JMS

Le format d'URI pris en charge dans le transport IBM MQ pour SOAP permet un degré de contrôle complet sur les paramètres et les options spécifiques à SOAP/ IBM MQ lors de l'accès aux services cible. Ce format est compatible avec WebSphere Application Server et avec CICS, ce qui facilite l'intégration de IBM MQ à ces deux produits.

La syntaxe de l'URI est la suivante:

```
jms:/queue? name=value&name=value...
```

où name est un nom de paramètre et *value* est une valeur appropriée, et l'élément name = *value* peut être répété autant de fois que nécessaire, la deuxième occurrence et les suivantes étant précédées d'une perluète (&).

Les noms de paramètre sont sensibles à la casse, tout comme les noms des objets IBM MQ . Si un paramètre est spécifié plusieurs fois, l'occurrence finale du paramètre prend effet, ce qui signifie que les applications client peuvent remplacer les valeurs de paramètre en les ajoutant à l'URI. Si d'autres paramètres non reconnus sont inclus, ils sont ignorés.

Si vous stockez un URI dans une chaîne XML, vous devez représenter le caractère perluète sous la forme "&". De même, si un URI est codé dans un script, prenez soin de mettre en échappement les caractères tels que **&** qui, autrement, seraient interprétés par le shell.

Voici un exemple d'URI simple pour un service Axis:

```
jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Voici un exemple d'URI simple pour un service .NET :

```
jms:/queue?destination=myQ&connectionFactory=()&targetService=MyService.asmx
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Seuls les paramètres requis sont fournis (`targetService` est requis pour les services .NET uniquement) et `connectionFactory` ne dispose d'aucune option.

Dans cet exemple Axis, `connectionFactory` contient un certain nombre d'options:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

Dans cet exemple d'axe, l'option `sslPeerName` de `connectionFactory` a également été spécifiée. La valeur du nom sslPeercontient elle-même des paires nom-valeur et des blancs imbriqués significatifs:

```
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
```

Interface NON-SOAP/Non-JMS

Le format d'URI de l'interface NON-SOAP/Non-JMS permet un contrôle complet des paramètres et des options spécifiques à IBM MQ lors de l'accès aux services cible.

La syntaxe de l'URI est la suivante:

```
wmq://example.com:1415/msg/queue/INS.QUOTE.REQUEST@MOTOR.INS ?ReplyTo=msg/queue/
INS.QUOTE.REPLY@BRANCH452&persistence=MQPER_NOT_PERSISTENT
```

Cet IRI indique à un demandeur de service qu'il peut utiliser une connexion de liaison client IBM MQ TCP à une machine appelée example.com sur le port 1415 et insérer des messages de demande persistants dans une file d'attente appelée INS.QUOTE.REQUEST sur le gestionnaire de files d'attente MOTOR.INS. L'IRI indique que le fournisseur de services insère des réponses dans une file d'attente appelée INS.QUOTE.REPLY sur le gestionnaire de files d'attente BRANCH452. Le format d'URI est celui spécifié pour SupportPac MA93. Pour plus d'informations sur les spécifications IRI IBM MQ , voir SupportPac MA93: IBM MQ -Service Definition .

Options de configuration de liaison WCF

Il existe deux manières d'appliquer des options de configuration aux informations de liaison des canaux personnalisés. Vous pouvez soit définir les propriétés de manière administrative, soit les définir à l'aide d'un programme.

Les options de configuration de liaison peuvent être définies de l'une des deux manières suivantes:

1. Administrativement: les paramètres de propriété de liaison doivent être spécifiés dans la section transport de la définition de liaison personnalisée dans le fichier de configuration des applications, par exemple: `app.config`.
2. A l'aide d'un programme: le code d'application doit être modifié pour spécifier la propriété lors de l'initialisation de la liaison personnalisée.

Définition administrative des propriétés de liaison

Les paramètres de propriété de liaison peuvent être spécifiés dans le fichier de configuration de l'application, par exemple: `app.config`. Le fichier de configuration est généré par **svcutil**, comme illustré dans les exemples suivants.

Interface SOAP/JMS

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Interface non SOAP/non-JMS

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Définition des propriétés de liaison à l'aide d'un programme

Pour ajouter une propriété de liaison WCF afin de spécifier le mode de connexion client, vous devez modifier le code de service afin de spécifier la propriété lors de l'initialisation de la liaison personnalisée.

Utilisez l'exemple suivant pour spécifier le mode de connexion client non géré:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

Propriétés de liaison WCF

Tableau 194. Valeurs des propriétés de liaison lors de la définition administrative ou à l'aide d'un programme

Nom de la propriété	Application client ou de service	Valeur d'administration	Valeur programmatique	Description
maxBufferPoolSize	Les deux	Entier signé de 0 à 64 bits	Entier signé de 0 à 64 bits	Indique la taille maximale de la mémoire pouvant être utilisée pour stocker les mémoires tampon de messages WCF pour une instance du canal.
MaxMessageSize	Les deux	Entier signé de 1 à 32 bits	Entier signé de 1 à 32 bits	Indique la mémoire maximale pouvant être utilisée pour un message WCF individuel.
Mode clientConnection	Les deux	0 (valeur par défaut) 1	AS_URI (valeur par défaut) CLIENT_UNMANAGED	Indique le mode de connexion client du canal de transport. 0 signifie que le mode de connexion client est celui spécifié dans l'URI. Utilisé uniquement si la connexion client est utilisée. Indique que le mode de connexion client est celui spécifié dans l'URI. 0 est la valeur par défaut si aucun mode de connexion client n'est défini. 1 signifie que le mode de connexion client est un client non géré. Utilisé uniquement si la connexion client est utilisée.

Tableau 194. Valeurs des propriétés de liaison lors de la définition administrative ou à l'aide d'un programme (suite)

Nom de la propriété	Application client ou de service	Valeur d'administration	Valeur programmatique	Description
Appels MaxConcurrent	Environnement	La plage est comprise entre 0 et 2 147 483 647 16 est la valeur par défaut	La plage est comprise entre 0 et 2 147 483 647 16 est la valeur par défaut	Cette propriété définit le nombre maximal d'opérations simultanées pouvant être effectuées sur un proxy client individuel à un moment donné. Si d'autres opérations sont démarrées, elles sont mises en file d'attente jusqu'à ce qu'une opération en cours se termine ou expire. Ce paramètre peut être utilisé pour contrôler le nombre maximal d'unités d'exécution et de ressources pouvant être consommées par un proxy individuel. `0` supprime cette limite, ce qui permet à toutes les opérations d'être tentées simultanément.
Appels MaxConcurrent	Service	La plage est comprise entre 1 et 2 147 483 647 16 est la valeur par défaut	La plage est comprise entre 1 et 2 147 483 647 16 est la valeur par défaut	Cette propriété est utilisée uniquement si la fonction de distribution assurée est activée (pour plus d'informations sur la distribution assurée, voir «Distribution garantie de canal personnalisé WCF», à la page 1306). Il indique le nombre maximal d'opérations simultanées pouvant être en cours simultanément pour le noeud final donné. Des précautions sont nécessaires lors de la modification de ce paramètre. Chaque opération simultanée nécessite des ressources supplémentaires, en particulier une nouvelle instance du canal personnalisé et les unités d'exécution associées du pool d'unités d'exécution pour traiter les demandes. La surallocation peut être contre-productive et affecter gravement les performances. La configuration appropriée du pool d'unités d'exécution doit être effectuée pour prendre en charge cette propriété.

Services de construction et d'hébergement pour WCF

Présentation des services Microsoft Windows Communication Foundation (WCF) expliquant comment créer et configurer des services WCF.

Le canal personnalisé IBM MQ pour WCF et les services WCF qui l'utilisent peuvent être hébergés par les méthodes suivantes:

- Auto-hébergement
- Service Windows

Le canal personnalisé IBM MQ pour WCF ne peut pas être hébergé dans le service d'activation de processus Windows .

Les rubriques suivantes fournissent des exemples simples d'auto-hébergement pour illustrer les étapes impliquées. La documentation en ligne d' Microsoft WCF, qui contient des informations supplémentaires et les informations les plus récentes, est disponible sur le site Web Microsoft MSDN à l'adresse https://msdn.microsoft.com.

Génération d'applications de service WCF à l'aide de la méthode 1: auto-hébergement administrativement à l'aide d'un fichier de configuration d'application

Après avoir créé un fichier de configuration d'application, ouvrez une instance du service et ajoutez le code spécifié à votre application.

Avant de commencer

Créez ou éditez un fichier de configuration d'application pour le service, comme décrit dans: «Création d'un canal personnalisé WCF de manière administrative en fournissant des informations de liaison et de noeud final dans un fichier de configuration d'application», à la page 1311

Pourquoi et quand exécuter cette tâche

1. Instanciez et ouvrez une instance du service dans l'hôte de service. Le type de service doit être identique au type de service spécifié dans le fichier de configuration de service.
2. Ajoutez le code suivant à votre application:

```
ServiceHost service = new ServiceHost(typeof(MyService));
service.Open();
...
service.Close();
```

Génération d'applications de service WCF à l'aide de la méthode 2: Hébergement automatique à l'aide d'un programme directement à partir de l'application

Ajoutez les propriétés de liaison, créez l'hôte de service avec une instance de la classe de service requise et ouvrez le service.

Avant de commencer

1. Ajoutez une référence au fichier `IBM.XMS.WCF.dll` du canal personnalisé au projet. `IBM.XMS.WCF.dll` se trouve dans *`WMQInstallDir`*`\bin` , où *WMQInstallDir* est le répertoire dans lequel IBM MQ est installé.
2. Ajoutez une instruction *using* à l'espace de nom `IBM.XMS.WCF` , par exemple: `using IBM.XMS.WCF`
3. Créez une instance de l'élément de liaison de canaux et du noeud final comme décrit dans: «Création d'un canal personnalisé WCF en fournissant des informations de liaison et de noeud final à l'aide d'un programme», à la page 1313

Pourquoi et quand exécuter cette tâche

Si des modifications doivent être apportées aux propriétés de liaison du canal, procédez comme suit:

1. Ajoutez les propriétés de liaison à `transportBindingElement` comme illustré dans l'exemple suivant:

```
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. Créez l'hôte de service avec une instance de la classe de service requise:

```
ServiceHost service = new ServiceHost(typeof(MyService));
```

3. Ouvrez le service:

```
service.AddServiceEndpoint(typeof(IMyServiceContract), binding, address);
service.Open();
...
service.Close();
```

Exposition de métadonnées à l'aide d'un noeud final HTTP

Instructions d'exposition des métadonnées d'un service configuré pour utiliser le canal personnalisé IBM MQ pour WCF.

Pourquoi et quand exécuter cette tâche

Si les métadonnées des services doivent être exposées (de sorte que les outils tels que `svcutil` puissent y accéder directement à partir du service en cours d'exécution plutôt qu'à partir d'un fichier WSDL hors ligne, par exemple), vous devez exposer les métadonnées des services avec un noeud final HTTP. Les étapes suivantes peuvent être utilisées pour ajouter ce noeud final supplémentaire.

1. Ajoutez l'adresse de base de l'emplacement où les métadonnées doivent être exposées à ServiceHost, par exemple:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Ajoutez le code suivant à ServiceHost avant d'ouvrir le service:

```
ServiceMetadataBehavior metadataBehavior = new ServiceMetadataBehavior();
  metadataBehavior.HttpGetEnabled = true;
  service.Description.Behaviors.Add(metadataBehavior);
  service.AddServiceEndpoint(typeof(IMetadataExchange),
     MetadataExchangeBindings.CreateMexHttpBinding(), "mex");
```

Résultats

Les métadonnées sont désormais disponibles à l'adresse suivante: `http://localhost:8000/MyService`

Génération d'applications client pour WCF

Présentation de la génération et de la génération d'applications client Microsoft Windows Communication Foundation (WCF).

Une application client peut être créée pour un service WCF ; les applications client sont généralement générées à l'aide de l'outil Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) pour créer les fichiers de configuration et de proxy requis qui peuvent être utilisés directement par l'application.

Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec des métadonnées à partir d'un service en cours d'exécution

Instructions d'utilisation de l'outil Microsoft svcutil.exe pour générer un client pour un service configuré pour utiliser le canal personnalisé IBM MQ pour WCF.

Avant de commencer

Il existe trois conditions préalables à l'utilisation de l'outil svcutil pour créer les fichiers de configuration et de proxy requis qui peuvent être utilisés directement par l'application:

- Le service WCF doit être en cours d'exécution avant le démarrage de l'outil svcutil.
- Le service WCF doit exposer ses métadonnées à l'aide d'un port HTTP en plus des références de noeud final de canal personnalisé IBM MQ pour générer un client directement à partir d'un service en cours d'exécution.
- Le canal personnalisé doit être enregistré dans les données de configuration de svcutil.

Pourquoi et quand exécuter cette tâche

Les étapes suivantes expliquent comment générer un client pour un service configuré pour utiliser le canal personnalisé IBM MQ , mais exposent également ses métadonnées lors de l'exécution via un port HTTP distinct:

1. Démarrez le service WCF (le service doit être en cours d'exécution avant le démarrage de l'outil svcutil).
2. Ajoutez les détails du fichier de configuration `svcutil.exe` à partir de la racine de l'installation dans le fichier de configuration svcutil actif, généralement `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` afin que svcutil reconnaisse le canal personnalisé IBM MQ .
3. Exécutez svcutil à partir d'une invite de commande, par exemple:

```
svcutil /language:C# /r: installlocation\bin\IBM.XMS.WCF.dll
/config:app.config http://localhost:8000/IBM.XMS.WCF/samples
```

4. Copiez les fichiers `app.config` et `YourService.cs` générés dans le projet client Microsoft Visual Studio.

Que faire ensuite

Si les métadonnées des services ne peuvent pas être extraites directement, svcutil peut être utilisé pour générer les fichiers client à partir de wsdl à la place. Pour plus d'informations, voir: «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec WSDL», à la page 1321

Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec WSDL

Instructions de génération de clients WCF à partir de WSDL si les métadonnées du service ne sont pas disponibles.

Si les métadonnées du service ne peuvent pas être directement extraites pour générer un client à partir des métadonnées d'un service en cours d'exécution, svcutil peut être utilisé pour générer les fichiers client à partir de WSDL à la place. Les modifications suivantes doivent être apportées au WSDL pour indiquer que le canal personnalisé IBM MQ doit être utilisé:

1. Ajoutez les définitions d'espace de nom et les informations de règle suivantes:

```
<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. Modifiez la section bindings pour faire référence à la nouvelle section policy et supprimez toute définition `transport` de l'élément de liaison sous-jacent:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Exécutez svcutil à partir d'une invite de commande, par exemple:

```
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
```

Génération d'applications client WCF à l'aide d'un proxy client avec un fichier de configuration d'application

Avant de commencer

Créez ou éditez un fichier de configuration d'application pour le client, comme décrit dans: «Création d'un canal personnalisé WCF de manière administrative en fournissant des informations de liaison et de noeud final dans un fichier de configuration d'application», à la page 1311

Pourquoi et quand exécuter cette tâche

Instanciez et ouvrez une instance du proxy client. Le paramètre transmis au proxy généré doit être identique au nom de noeud final spécifié dans le fichier de configuration du client, par exemple `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();
            }
```

Génération d'applications client WCF à l'aide d'un proxy client avec une configuration par programmation

Avant de commencer

1. Ajoutez une référence au fichier `IBM.XMS.WCF.dll` du canal personnalisé au projet. `IBM.XMS.WCF.dll` se trouve dans le répertoire *WMQInstallDir*`\bin` , où *WMQInstallDir* est le répertoire dans lequel IBM MQ est installé.
2. Ajoutez une instruction *using* à l'espace de nom `IBM.XMS.WCF` , par exemple: `using IBM.XMS.WCF`
3. Créez une instance de l'élément de liaison et du noeud final du canal, comme décrit dans: «Création d'un canal personnalisé WCF en fournissant des informations de liaison et de noeud final à l'aide d'un programme», à la page 1313

Pourquoi et quand exécuter cette tâche

Si des modifications doivent être apportées aux propriétés de liaison du canal, procédez comme suit.

1. Ajoutez les propriétés de liaison à `transportBindingElement` comme illustré dans la figure suivante:

```
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. Créez le proxy client comme illustré dans la figure suivante, où *binding* et *endpoint address* correspondent à la liaison et à l'adresse de noeud final configurées à l'étape 1 et transmises:

```
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();
           }
```

Utilisation des exemples WCF

Les exemples Windows Communication Foundation (WCF) fournissent des exemples simples de la façon dont le canal personnalisé IBM MQ peut être utilisé.

Pour générer les exemples de projets, le SDK Microsoft.NET 3.5 ou Microsoft Visual Studio 2008 est nécessaire.

Exemple WCF client et serveur unidirectionnel simple

Cet exemple illustre le canal personnalisé IBM MQ utilisé pour démarrer un service Windows Communication Foundation (WCF) à partir d'un client WCF à l'aide d'une forme de canal unidirectionnel.

Pourquoi et quand exécuter cette tâche

Le service implémente une méthode unique qui génère une chaîne sur la console. Le client a été généré à l'aide de l'outil `svcutil` pour extraire les métadonnées de service d'un noeud final HTTP exposé séparément, comme décrit dans «Génération d'un proxy client WCF et de fichiers de configuration

d'application à l'aide de l'outil svcutil avec des métadonnées à partir d'un service en cours d'exécution», à la page 1321

L'exemple a été configuré avec des noms de ressource spécifiques, comme décrit dans la procédure suivante. Si vous devez modifier les noms de ressource, vous devez également modifier la valeur correspondante sur l'application client dans le fichier *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\app.config` et sur l'application de service dans le fichier *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\TestServices.cs` , où *MQ_INSTALLATION_PATH* correspond au répertoire d'installation de IBM MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS , voir *IBM MQ Transport for SOAP* dans la documentation du produit IBM MQ . Si vous devez modifier l'exemple de solution et la source, vous avez besoin d'un environnement de développement intégré, par exemple Microsoft Visual Studio 8 ou version ultérieure.

Procédure

1. Créez un gestionnaire de files d'attente appelé *QM1*
2. Créez une destination de file d'attente appelée *SampleQ*
3. Démarrez le service pour que le programme d'écoute attende les messages: exécutez le fichier *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.
4. Exécutez le client une seule fois: exécutez le fichier *MQ_INSTALLATION_PATH* `\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\client\bin\Release\TestClient.exe` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.

   L'application client boucle cinq fois en envoyant cinq messages à *SampleQ*

Résultats

L'application de service extrait les messages de *SampleQ* et affiche `Hello World` à l'écran cinq fois.

Que faire ensuite

Exemple WCF client et serveur de demande-réponse simple

Cet exemple illustre le canal personnalisé IBM MQ utilisé pour démarrer un service Windows Communication Foundation (WCF) à partir d'un client WCF à l'aide d'une forme de canal de demande-réponse.

Pourquoi et quand exécuter cette tâche

Ce service fournit des méthodes de calcul simples permettant d'ajouter et de soustraire deux nombres, puis de renvoyer le résultat. Le client a été généré à l'aide de l'outil `svcutil` pour extraire les métadonnées de service d'un noeud final HTTP exposé séparément, comme décrit dans «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec des métadonnées à partir d'un service en cours d'exécution», à la page 1321

L'exemple a été configuré avec des noms de ressource spécifiques, comme dans la procédure décrite ci-après. Si vous devez modifier les noms de ressource, vous devez également modifier la valeur correspondante sur l'application client dans le fichier *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\app.config` et sur l'application de service dans le fichier *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\RequestReplyService.cs` , où *MQ_INSTALLATION_PATH* correspond au répertoire d'installation de IBM MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS , voir *IBM MQ Transport for SOAP* dans la documentation du produit IBM MQ . Si vous devez modifier l'exemple de solution et la source, vous avez besoin d'un environnement de développement intégré, par exemple Microsoft Visual Studio 8 ou version ultérieure.

Procédure

1. Créez un gestionnaire de files d'attente appelé *QM1*
2. Créez une destination de file d'attente appelée *SampleQ*
3. Créez une destination de file d'attente appelée *SampleReplyQ*
4. Démarrez le service pour que le programme d'écoute attende les messages: exécutez le fichier *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.
5. Exécutez le client une seule fois: exécutez le fichier *MQ_INSTALLATION_PATH* `\Tools\wcf\samples\WCF\requestreply\client\bin\Release\SimpleRequestReply_Client.exe` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.

Résultats

Une fois que le client a été exécuté, le processus suivant est démarré et se répète quatre fois, de sorte qu'un total de cinq messages sont envoyés de chaque côté:

1. Le client place un message de demande sur *SampleQ* et attend une réponse.
2. Le service obtient le message de demande de *SampleQ*.
3. Le service ajoute et soustrait des valeurs à l'aide du contenu du message.
4. Le service place ensuite les résultats dans un message sur *SampleReplyQ*et attend que le client insère un nouveau message.
5. Le client reçoit le message de *SampleReplyQ* et affiche les résultats à l'écran.

Que faire ensuite

Client WCF vers un service .NET hébergé par l'exemple IBM MQ

Des exemples d'applications client et des exemples d'applications proxy de service sont fournis pour .NET et Java. Les exemples sont basés sur un service Stock Quote qui prend une demande de cotation boursière, puis fournit la cotation boursière.

Avant de commencer

L'exemple nécessite que l'environnement d'hébergement de service .NET SOAP sur JMS soit correctement installé et configuré dans IBM MQ et qu'il soit accessible à partir d'un gestionnaire de files d'attente local.

Lorsque l'environnement d'hébergement de service .NET SOAP sur JMS est correctement installé et configuré dans IBM MQ et qu'il est accessible à partir d'un gestionnaire de files d'attente local, des étapes de configuration supplémentaires doivent être effectuées.

1. Définissez la variable d'environnement **WMQSOAP_HOME** sur le répertoire d'installation IBM MQ , par exemple: `C:\Program Files\IBM\MQ`
2. Vérifiez que le Java compilateur `javac` est disponible et dans la variable PATH.
3. Copiez le fichier `axis.jar` du répertoire `prereqs/axis` de l'image d'installation vers le répertoire de production IBM MQ , par exemple: `C:\Program Files\IBM\MQ\java\lib\soap`
4. Ajoutez à la variable PATH: *MQ_INSTALLATION_PATH*`\Java\lib` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé, par exemple: `C:\Program Files\IBM\MQ`
5. Vérifiez que l'emplacement de .NET est spécifié correctement dans *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé, par exemple: `C:\Program Files\IBM\MQ`. L'emplacement de .NET peut être spécifié, par exemple: `set msfwdir=%ProgramFiles%\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Une fois les étapes précédentes terminées, testez et exécutez le service:

1. Accédez à votre répertoire de travail SOAP sur JMS .
2. Entrez l'une des commandes suivantes pour exécuter le test de vérification et laisser le programme d'écoute du service en cours d'exécution:
   - Pour .NET: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt dotnet hold , où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé.
   - Pour AXIS: *MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé.

L'argument hold maintient les programmes d'écoute en cours d'exécution une fois le test terminé.

Si des erreurs sont signalées lors de cette configuration, vous pouvez supprimer toutes les modifications afin que la procédure puisse être redémarrée de la manière suivante:

1. Supprimez le répertoire SOAP sur JMS généré.
2. Supprimez le gestionnaire de files d'attente.

Pourquoi et quand exécuter cette tâche

Cet exemple illustre une connexion entre un client WCF et l'exemple de service .NET SOAP sur JMS fourni dans IBM MQ à l'aide d'une forme de canal unidirectionnel. Le service implémente un exemple StockQuote simple, qui génère une chaîne de texte sur la console.

Le client a été généré à l'aide de WSDL pour générer des fichiers client, comme décrit dans «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec WSDL», à la page 1321

L'exemple a été configuré avec des noms de ressource spécifiques, comme décrit dans la procédure suivante. Si vous devez modifier les noms de ressource, vous devez également modifier la valeur correspondante sur l'application client dans le fichier *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\app.config et sur l'application de service dans le fichier *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl , où *MQ_INSTALLATION_PATH* représente le répertoire d'installation de IBM MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS , voir *IBM MQ Transport for SOAP* dans la documentation du produit IBM MQ .

Procédure

Exécutez le client une seule fois: exécutez le fichier *MQ_INSTALLATION_PATH* \tools\wcf\samples\WMQNET\default\client\bin\Release\TestClient.exe , où *MQ_INSTALLATION_PATH* représente le répertoire d'installation de IBM MQ.

L'application client effectue une boucle cinq fois en envoyant cinq messages à l'exemple de file d'attente.

Résultats

L'application de service extrait les messages de l'exemple de file d'attente et affiche Hello World cinq fois à l'écran.

Client WCF vers un service Axis Java hébergé par l'exemple IBM MQ

Des exemples d'applications client et des exemples d'applications proxy de service sont fournis pour Java et .NET. Les exemples sont basés sur un service Stock Quote qui prend une demande de cotation boursière, puis fournit la cotation boursière.

Avant de commencer

Cet exemple requiert que l'environnement d'hébergement de service .NET SOAP sur JMS soit correctement installé et configuré dans IBM MQ et qu'il soit accessible à partir d'un gestionnaire de files d'attente local.

Lorsque l'environnement d'hébergement de service .NET SOAP sur JMS est correctement installé et configuré dans IBM MQ et qu'il est accessible à partir d'un gestionnaire de files d'attente local, des étapes de configuration supplémentaires doivent être effectuées.

1. Définissez la variable d'environnement **WMQSOAP_HOME** sur le répertoire d'installation IBM MQ , par exemple: `C:\Program Files\IBM\MQ`
2. Vérifiez que le Java compilateur `javac` est disponible et dans la variable PATH.
3. Copiez le fichier `axis.jar` du répertoire `prereqs/axis` de l'image d'installation vers le répertoire d'installation IBM MQ .
4. Ajoutez à la variable PATH: *MQ_INSTALLATION_PATH*`\Java\lib` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé, par exemple: `C:\Program Files\IBM\MQ`
5. Vérifiez que l'emplacement de .NET est spécifié correctement dans *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé, par exemple: `C:\Program Files\IBM\MQ`. L'emplacement de .NET peut être spécifié, par exemple: `set msfwdir=%ProgramFiles% \Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Une fois les étapes précédentes terminées, testez et exécutez le service:

1. Accédez à votre répertoire de travail SOAP sur JMS .
2. Entrez l'une des commandes suivantes pour exécuter le test de vérification et laisser le programme d'écoute du service en cours d'exécution:
   - Pour .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé.
   - Pour AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel IBM MQ est installé.

L'argument `hold` maintient les programmes d'écoute en cours d'exécution une fois le test terminé.

Si des erreurs sont signalées lors de cette configuration, vous pouvez supprimer toutes les modifications afin que la procédure soit redémarrée de la manière suivante:

1. Supprimez le répertoire SOAP sur JMS généré.
2. Supprimez le gestionnaire de files d'attente.

Pourquoi et quand exécuter cette tâche

L'exemple illustre une connexion d'un client WCF à l'exemple de service Axis Java SOAP over JMS fourni dans IBM MQ à l'aide d'une forme de canal unidirectionnel. Le service implémente un exemple simple StockQuote , qui génère une chaîne de texte dans un fichier sauvegardé dans le répertoire en cours.

Le client a été généré à l'aide de WSDL pour générer des fichiers client, comme décrit dans «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec WSDL», à la page 1321

L'exemple a été configuré avec des noms de ressource spécifiques, comme décrit dans ce paragraphe. Si vous devez modifier les noms de ressource, vous devez également modifier la valeur correspondante sur l'application client dans le fichier *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\app.config` et sur l'application de service dans le fichier *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` , où *MQ_INSTALLATION_PATH* représente le répertoire d'installation de IBM MQ.

Procédure

Exécutez le client une seule fois: exécutez le fichier *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\client\bin\Release\TestClient.exe` , où *MQ_INSTALLATION_PATH* représente le répertoire d'installation de IBM MQ.

L'application client effectue une boucle cinq fois en envoyant cinq messages à l'exemple de file d'attente.

Résultats

L'application de service extrait les messages de l'exemple de file d'attente et ajoute `Hello World` cinq fois à un fichier dans le répertoire en cours.

Client WCF vers le service Java hébergé par l'exemple WebSphere Application Server

Des exemples d'applications client et des exemples d'applications proxy de service sont fournis pour WebSphere Application Server 6. Un service de demande-réponse est également fourni.

Avant de commencer

Cet exemple requiert l'utilisation de la configuration IBM MQ suivante:

Tableau 195. IBM MQ configuration requise

Objet	**Nom requis**
Gestionnaire de files d'attente	QM1
File d'attente locale	HelloWorld
File d'attente locale	Réponse HelloWorld

Cet exemple requiert également qu'un environnement d'hébergement WebSphere Application Server 6 soit correctement installé et configuré. WebSphere Application Server 6 utilise une connexion en mode liaison pour se connecter à IBM MQ par défaut. Par conséquent, WebSphere Application Server 6 doit être installé sur la même machine que le gestionnaire de files d'attente.

Une fois l'environnement WAS configuré, les étapes de configuration supplémentaires suivantes doivent être effectuées:

1. Créez les objets JNDI suivants dans le référentiel JNDI WebSphere Application Server :
   a. Une destination de file d'attente JMS appelée `HelloWorld`
      - Définissez le nom JNDI sur `jms/HelloWorld`
      - Définissez le nom de la file d'attente sur `HelloWorld`
   b. Une fabrique de connexions de file d'attente JMS appelée `HelloWorldQCF`
      - Définissez le nom JNDI sur `jms/HelloWorldQCF`
      - Définissez le nom du gestionnaire de files d'attente sur `QM1`
   c. Une fabrique de connexions de file d'attente JMS appelée `WebServicesReplyQCF`
      - Définissez le nom JNDI sur `jms/WebServicesReplyQCF`
      - Définissez le nom du gestionnaire de files d'attente sur `QM1`
2. Créez un port d'écoute de messages appelé `HelloWorldPort` dans WebSphere Application Server avec la configuration suivante:
   - Définissez le nom JNDI de la fabrique de connexions sur `jms/HelloWorldQCF`
   - Définissez le nom JNDI de la destination sur `jms/HelloWorld`
3. Installez l'application `HelloWorldEJBEAR.ear` de service Web sur votre serveur WebSphere Application Server comme suit:
   a. Cliquez sur **Applications** > **Nouvelle application** > **Nouvelle application d'entreprise**.
   b. Accédez à *MQ_INSTALLATION_PATH*`\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear`, où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.
   c. Ne modifiez aucune des options par défaut de l'assistant et redémarrez le serveur d'applications après l'installation de l'application.

Une fois la configuration WAS terminée, testez le service en l'exécutant une seule fois:

1. Accédez à votre répertoire de travail Soap over JMS .
2. Entrez cette commande pour exécuter l'exemple: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe` où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.

Pourquoi et quand exécuter cette tâche

L'exemple illustre une connexion entre un client WCF et l'exemple de service WebSphere Application Server SOAP sur JMS fourni dans les exemples WCF inclus dans IBM MQ, à l'aide d'une forme de canal de demande-réponse. Flux de messages entre WCF et WebSphere Application Server à l'aide des files d'attente IBM MQ . Le service implémente la méthode `HelloWorld(...)` , qui prend une chaîne et renvoie un message d'accueil au client.

Le client a été généré à l'aide de l'outil svcutil pour extraire les métadonnées de service d'un noeud final HTTP exposé séparément, comme décrit dans «Génération d'un proxy client WCF et de fichiers de configuration d'application à l'aide de l'outil svcutil avec des métadonnées à partir d'un service en cours d'exécution», à la page 1321

L'exemple a été configuré avec des noms de ressource spécifiques, comme décrit dans la procédure suivante. Si vous devez modifier les noms de ressource, vous devez également modifier la valeur correspondante sur l'application client dans le fichier *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\app.config` et sur l'application de service dans le *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , où *MQ_INSTALLATION_PATH* correspond au répertoire d'installation de IBM MQ.

Le service et le client sont basés sur le service et le client décrits dans l'article IBM Developer *Building a JMS web service using SOAP over JMS and WebSphere Studio*. Pour plus d'informations sur le développement de services Web SOAP sur JMS compatibles avec le canal personnalisé WCF IBM MQ , voir https://www.ibm.com/developerworks/websphere/library/techarticles/0402_du/0402_du.html.

Procédure

Exécutez le client une seule fois: exécutez le fichier *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\default\client\bin\Release\TestClient.exe` , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de IBM MQ.

L'application client démarre les deux méthodes de service en même temps, en envoyant deux messages à l'exemple de file d'attente.

Résultats

L'application de service extrait les messages de l'exemple de file d'attente et fournit une réponse à l'appel de méthode `HelloWorld(...)` que l'application client envoie à la console.

Remarques

:NONE.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Consultez votre représentant IBM local pour obtenir des informations sur les produits et services actuellement disponibles dans votre région. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service IBM puisse être utilisé. Tout produit, programme ou service fonctionnellement équivalent qui ne porte pas atteinte à un droit de propriété intellectuelle IBM peut être utilisé à la place. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour obtenir des informations sur les licences relatives aux informations sur deux octets (DBCS), contactez le service de la propriété intellectuelle IBM de votre pays ou envoyez vos demandes de renseignements, par écrit, à :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Le paragraphe suivant ne s’applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE "EN L'ETAT" SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et/ou programmes décrits dans ce document.

Les références à des sites Web non IBM sont fournies uniquement à titre d'information et n'impliquent en aucune façon une adhésion de ces sites Web. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Coordinateur d'interopérabilité logicielle, département 49XA
3605 Autoroute 52 N

Rochester, MN 55901
U.S.A.

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans le présent document et tous les éléments sous disponibles s'y rapportant sont fournis par IBM conformément aux dispositions du Contrat sur les produits et services IBM, aux Conditions Internationales d'Utilisation de Logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

Licence sur les droits d'auteur :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Documentation sur l'interface de programmation

Les informations d'interface de programmation, si elles sont fournies, sont destinées à vous aider à créer un logiciel d'application à utiliser avec ce programme.

Ce manuel contient des informations sur les interfaces de programmation prévues qui permettent au client d'écrire des programmes pour obtenir les services d' IBM MQ.

Toutefois, lesdites informations peuvent également contenir des données de diagnostic, de modification et d'optimisation. Ces données vous permettent de déboguer votre application.

Important : N'utilisez pas ces informations de diagnostic, de modification et d'optimisation en tant qu'interface de programmation car elles sont susceptibles d'être modifiées.

Marques

IBM, le logo IBM , ibm.com, sont des marques d' IBM Corporation dans de nombreux pays. La liste actualisée de toutes les marques d' IBM est disponible sur la page Web "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Les autres noms de produits et de services peuvent être des marques d'IBM ou d'autres sociétés.

Microsoft et Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

UNIX est une marque de The Open Group aux Etats-Unis et dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Ce produit inclut des logiciels développés par le projet Eclipse (https://www.eclipse.org/).

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Référence :

(1P) P/N: