7.5

Développement d'applications pour IBM WebSphere 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 1157.

Cette édition s'applique à la version 7 édition 5 d' IBM® WebSphere MQ et à toutes les éditions et modifications ultérieures, sauf indication contraire dans les nouvelles éditions.

Table des matières

Développement d'applications

IBM WebSphere MQ fournit plusieurs façons de développer des applications pour envoyer et recevoir les 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.

Avant de développer des applications pour IBM WebSphere MQ, assurez-vous de connaître les concepts de la rubrique IBM WebSphere MQ Présentation techniqueIBM WebSphere MQ Présentation technique.

Vous pouvez développer des applications pour IBM WebSphere MQ dans différents langages de programmation. Pour plus d'informations sur les langages de programmation pris en charge et leurs fonctions, voir «Choix du langage de programmation à utiliser», à la page 81.

Consultez les sections suivantes pour connaître les types d'applications que vous pouvez écrire pour IBM WebSphere MQ sur différentes plateformes.

Types d'application que vous pouvez écrire pour IBM WebSphere MQ

Ces informations concernent les types d'application qui peuvent être écrits sur IBM WebSphere MQ.

Les produits IBM WebSphere MQ sont des gestionnaires de files d'attente et des facilitateurs d'application. Ils prennent en charge l'interface IBM Message Queue Interface (MQI) via laquelle les programmes peuvent placer des messages dans une file d'attente et extraire des messages d'une file d'attente.

Avec IBM WebSphere MQ pour les plateformes nonz/OS , vous pouvez écrire des applications qui:

- 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 WebSphere MQ .
- Utilisez la mise en file d'attente de messages depuis CICS for TXSeries for AIX, TXSeries for HP-UX, TXSeries for Solaris et TXSeries pour les applications système Windows .
- Utilisez la mise en file d'attente de messages depuis Encina pour les systèmes AIX, HP-UX, Solaris et Windows .
- Utilisez la mise en file d'attente de messages depuis Tuxedo pour les systèmes AIX, AT & T, HP-UX, Solaris et Windows .
- Utilisez IBM WebSphere 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 WebSphere 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 WebSphere MQ complet ou à partir d'un environnement client IBM WebSphere MQ MQI sur les plateformes suivantes:
  - UNIX and Linux® systèmes
  - Windows

Concepts associés

Sécurité

Concepts de développement d'applications

Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

Avant de commencer à concevoir et à écrire vos applications IBM WebSphere MQ , familiarisez-vous avec les concepts IBM WebSphere MQ de base. Voir les rubriques de la rubrique Présentation technique. Pour plus d'informations sur les types d'application que vous pouvez écrire pour IBM WebSphere MQ, voir «Développement d'applications», à la page 7.

Utilisez les liens suivants pour découvrir les concepts IBM WebSphere MQ spécifiques au développement d'applications:

- «Messages IBM WebSphere MQ», à la page 10
- Messagerie point-à-point
- Présentation de la messagerie de type publication/abonnement WebSphere MQ
- «Utilisation de l'interface de file d'attente de messages (MQI) dans une application client», à la page 367
- «Utilisation des services Web dans WebSphere MQ», à la page 977
- «Programmes d'exit de canal pour les canaux de messagerie», à la page 413
- «Scénarios de support transactionnel», à la page 42

Avant de pouvoir exécuter des applications qui utilisent l'interface MQI, vous devez créer certains objets IBM WebSphere MQ . Pour plus d'informations, voir «Programmes d'application utilisant l'interface MQI», à la page 9.

Concepts associés

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Exemples de programmes WebSphere MQ», à la page 100
Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

«Ecriture d'une application de mise en file d'attente», à la page 202
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 client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Utilisation des classes WebSphere MQ pour JMS», à la page 741
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) est le fournisseur JMS fourni avec WebSphere MQ. Outre l'implémentation des interfaces définies dans le package javax.jms , WebSphere MQ classes for JMS fournit deux ensembles d'extensions à l'API JMS.

«Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX)», à la page 1065
Les WebSphere MQ Automation Classes for ActiveX (MQAX) sont des composants ActiveX qui fournissent des classes que vous pouvez utiliser dans votre application pour accéder à WebSphere MQ.

«Utilisation des classes WebSphere MQ pour Java», à la page 677

Les classes WebSphere MQ for Java vous permettent d'utiliser WebSphere MQ dans un environnement Java. Une application Java peut utiliser des classes WebSphere MQ pour Java ou des classes WebSphere MQ pour JMS pour accéder aux ressources WebSphere MQ .

«Utilisation de .NET», à la page 581
Les classes WebSphere MQ pour .NET permettent à un programme écrit dans l'infrastructure de programmation .NET de se connecter à WebSphere MQ en tant que client WebSphere MQ MQI ou de se connecter directement à un serveur WebSphere MQ .

«Utilisation de C++», à la page 652
WebSphere MQ fournit des classes C++ équivalentes aux objets WebSphere 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.

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

Programmes d'application utilisant l'interface MQI

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

La Figure 1, à la page 9 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.

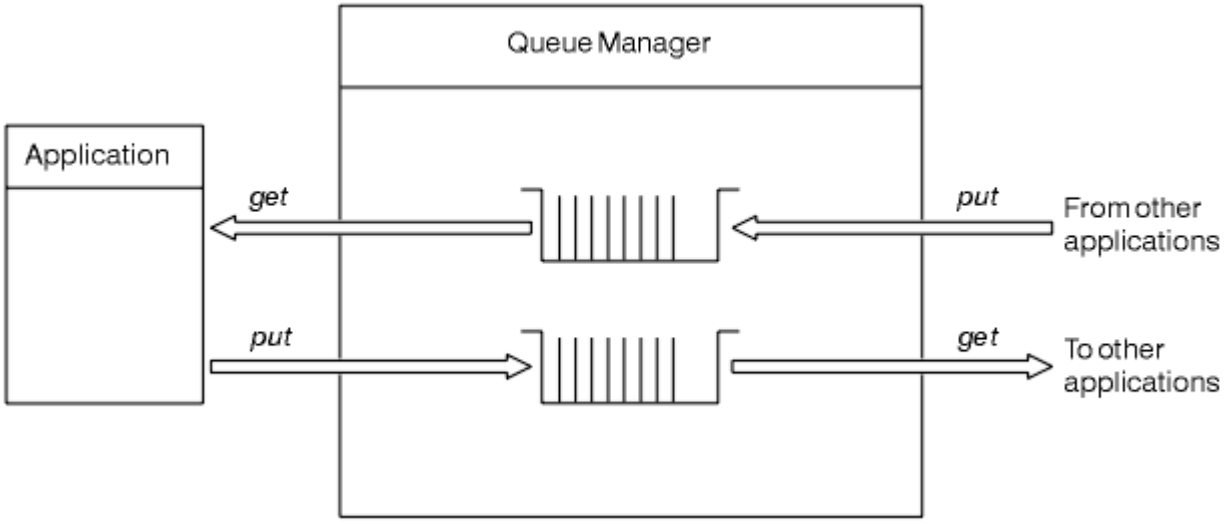


Figure 1. Files d'attente, messages et applications

Alors que les applications peuvent placer des messages dans des files d'attente locales ou distantes (à l'aide de MQPUT), elles peuvent uniquement extraire des messages 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 WebSphere MQ. Voir «Génération d'une application IBM WebSphere MQ», à la page 446.
- 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 9.

Messages IBM WebSphere MQ

Ces informations présentent le concept de message IBM WebSphere MQ , les parties de message et le descripteur de message.

Les messages IBM WebSphere MQ se composent de deux parties:

- Propriétés des messages
- Données d'application

Figure 2, à la page 10 représente un message et montre comment il est logiquement divisé en propriétés de message et en données d'application.

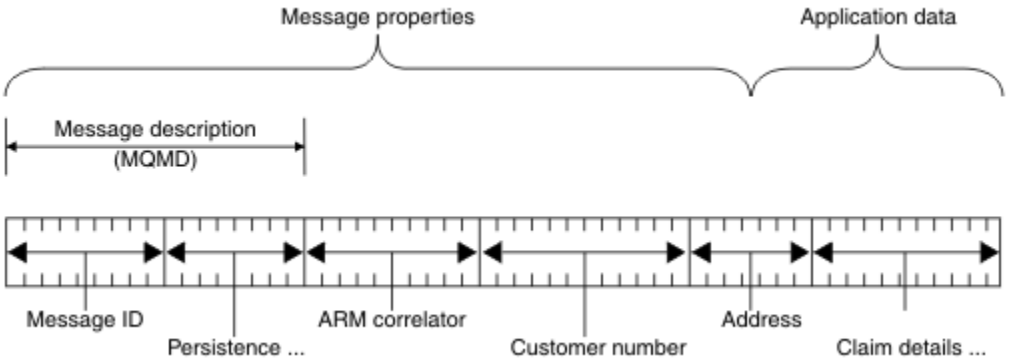


Figure 2. Représentation d'un message

Les données d'application transmises dans un message WebSphere MQ ne sont pas modifiées par un gestionnaire de files d'attente sauf si une conversion de données est effectuée sur ce dernier. En outre, WebSphere 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.

Sous WebSphere MQ for AIX, WebSphere MQ for HP-UX, WebSphere MQ for Linux, WebSphere MQ for Solaris, et WebSphere MQ for Windows, la valeur par défaut de *MaxMsgLength* est 100 Mo (104 857 600 octets).

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 240.

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 39 .

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 36) sont fournies dans la version 2 du descripteur de message (ou MQMDE). Il est identique au descripteur de message de la version 1 mais comporte des zones supplémentaires. Elles sont décrites dans MQMDE-Extension de descripteur de message.

Types de message

Il existe quatre types de message définis par IBM WebSphere 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 WebSphere 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.)

Rapports et messages segmentés

Non pris en charge sur WebSphere MQ for z/OS .

Si un message est segmenté (voir «Segmentation des messages», à la page 274 pour une description des messages segmentés) et que vous demandez la génération de rapports, vous risquez de recevoir plus de rapports que vous ne l'auriez fait si le message n'avait pas été segmenté.

Pour les rapports générés par WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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, MQIMS et 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 de message d'origine, à l'exclusion des longueurs des en-têtes WebSphere MQ que vous avez 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 de rapport, en plus de zéro, 100 octets ou de 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)
- WebSphere MQ , par exemple des formats de commande programmables

  WebSphere 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 432). 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 WebSphere 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, WebSphere 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, WebSphere 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 432 .

Priorités des messages

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 225), 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 WebSphere 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 WebSphere MQ et les applications 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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.

Lorsqu'elles sont transmises à un gestionnaire de files d'attente antérieur à la version 7.0 du produit, les propriétés du message, à l'exception de celles du descripteur de message, sont prises en compte dans la longueur du message. Par conséquent, vous devez augmenter la valeur de l'attribut *MaxMsgLength* des canaux vers un système antérieur à la version 7.0 si nécessaire, afin de compenser le fait que des données supplémentaires peuvent être envoyées pour chaque message. Vous pouvez également réduire la taille de la file d'attente ou du gestionnaire de files d'attente *MaxMsgLength*, de sorte que le niveau global des données envoyées sur le système reste le même.

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 WebSphere 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, sauf que le caractère Unicode U+002E (.) est autorisé dans le nom-mais pas le début. Les règles pour les identificateurs Java correspondent 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.

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

   Ces synonymes permettent à une application MQI d'accéder aux propriétés JMS d'une manière similaire à WebSphere MQ classes for JMS client application. 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 WebSphere 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". Ces sont les 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" et ce 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 - Définir 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. < Champ>	< Champ>

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 la section "MQMDE spécifié sur les appels MQPUT et MQPUT1 " dans 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 pas de conversion Unicode prise en charge par WebSphere 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 24 montre le processus de sélection à l'aide de l'appel MQSUB.

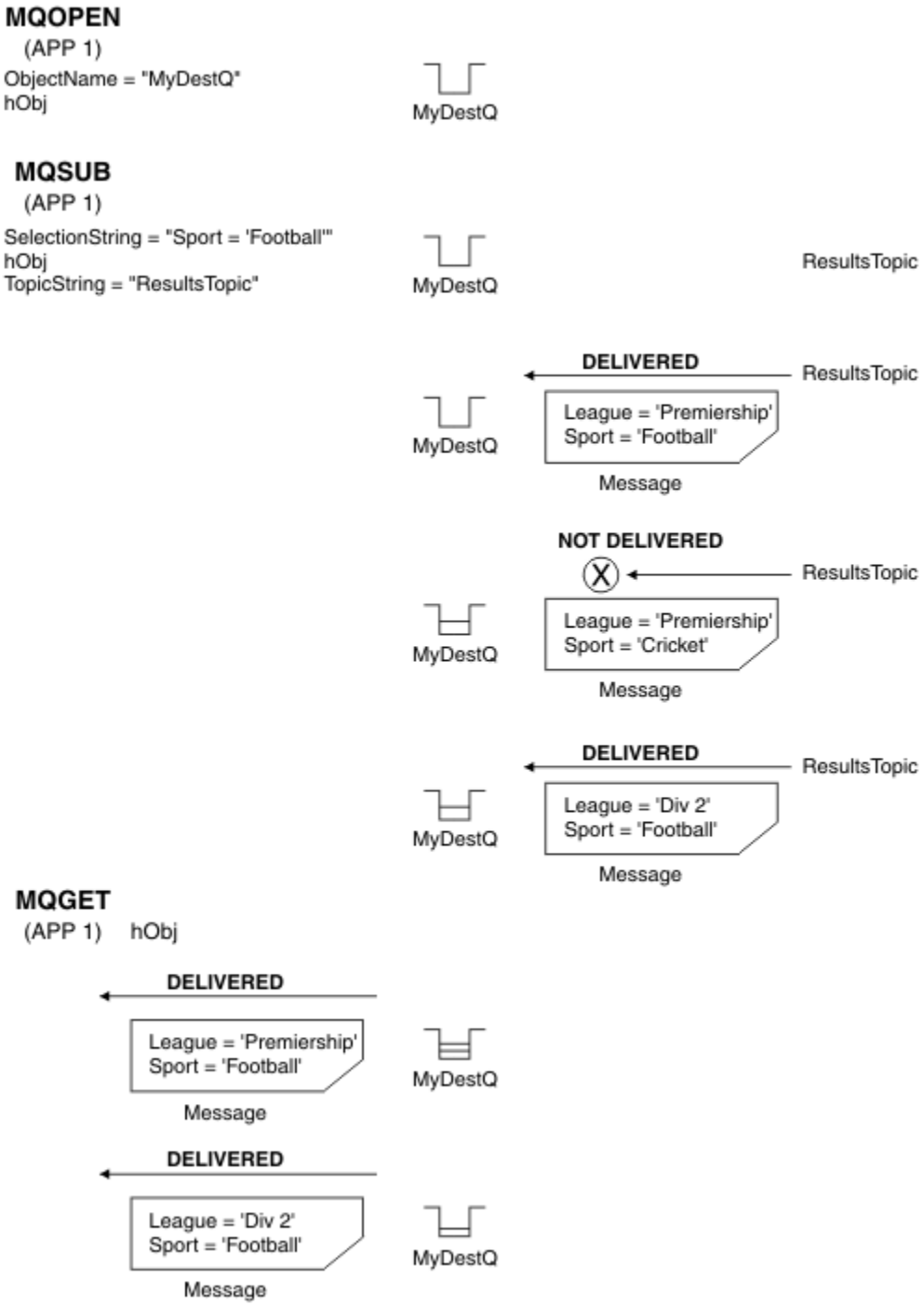


Figure 3. Sélection à l'aide de l'appel MQSUB

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 25 montre le processus de sélection à l'aide de l'appel MQOPEN.

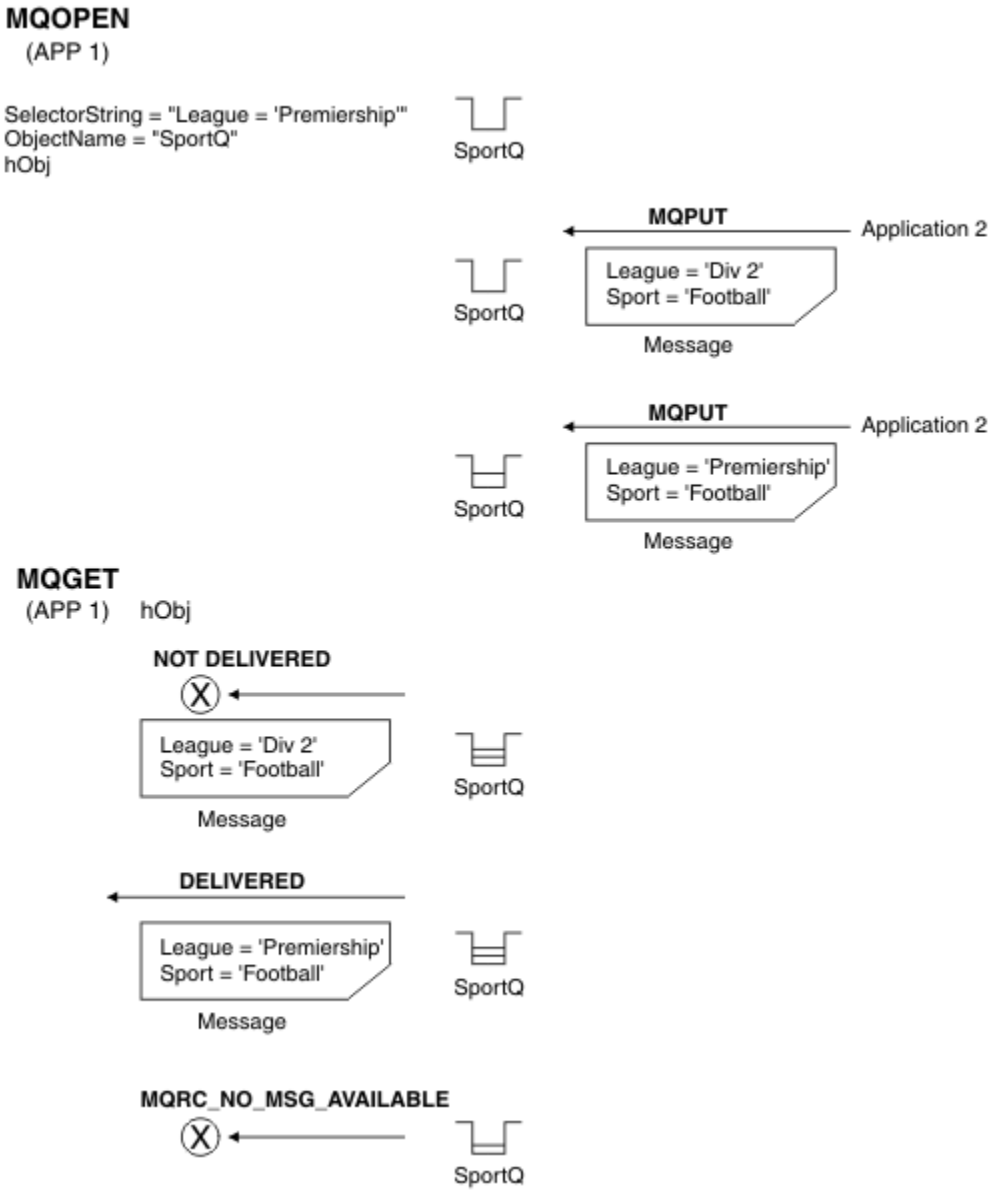


Figure 4. Sélection à l'aide de l'appel MQOPEN

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.

Comportement de la sélection
Présentation du comportement de la sélection IBM WebSphere 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 se résolve en TRUE ou FALSE avant toute correspondance avec des 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 WebSphere 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 de la forme `JMSCorrelationID = ...` ou `JMSMessageID = ...` et référencez une seule propriété.

Cette méthode offre une amélioration significative des performances pour la sélection sur JMSCorrelationID et une amélioration marginale des performances pour JMSMessageID.

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 WebSphere 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 après une centaine de tests sur certaines plateformes, de sorte que les sélecteurs approchant ce nombre de composants peuvent voir des défaillances. Il est recommandé que l'utilisation de sélecteurs avec de nombreux composants soit essayée et testée de manière approfondie sur les plateformes appropriées afin de s'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

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

Sélection du contenu d'un message
Il est possible de s'abonner en fonction d'une sélection de contenu 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 WebSphere MQ; à la place, un fournisseur de sélection de message étendu, par exemple IBM Integration Bus, est requis pour traiter les messages.

Syntaxe du sélecteur de message
Un sélecteur de message WebSphere 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.

WebSphere 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 qu'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.

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`, `-957`et `+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. WebSphere 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.`, `-95.7`ou `+6.2`. WebSphere 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 pour les noms de propriété de message. Pour plus d'informations, voir «Noms de propriétés», à la page 20 et «Restrictions de nom de propriété», à la page 21 .

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. WebSphere 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 836. 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 numériques approximatifs, les éléments suivants ne sont pas définis:

  – 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. 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. 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
T	T	T
F	T	T
F	U	U
F	F	F

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

Tableau 3. 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

Comportement de la sélection
Présentation du comportement de la sélection IBM WebSphere MQ .

Sélection du contenu d'un message
Il est possible de s'abonner en fonction d'une sélection de contenu 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 WebSphere MQ; à la place, un fournisseur de sélection de message étendu, par exemple IBM Integration Bus, est requis pour traiter les messages.

«Propriétés des messages», à la page 18
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 WebSphere MQ et les applications 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.

- 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 de <>; 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

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

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.

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.

Concepts associés

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.

Sélection du contenu d'un message

Il est possible de s'abonner en fonction d'une sélection de contenu 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 WebSphere 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 disposent d'une chaîne de sélection sélectionnée sur le contenu du message, WebSphere MQ demande que le fournisseur de sélection de message étendu analyse la publication et indique à WebSphere 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.

Concepts associés

Comportement de la sélection
Présentation du comportement de la sélection IBM WebSphere MQ .

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

Consommation asynchrone de messages IBM WebSphere 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 les 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 35 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.

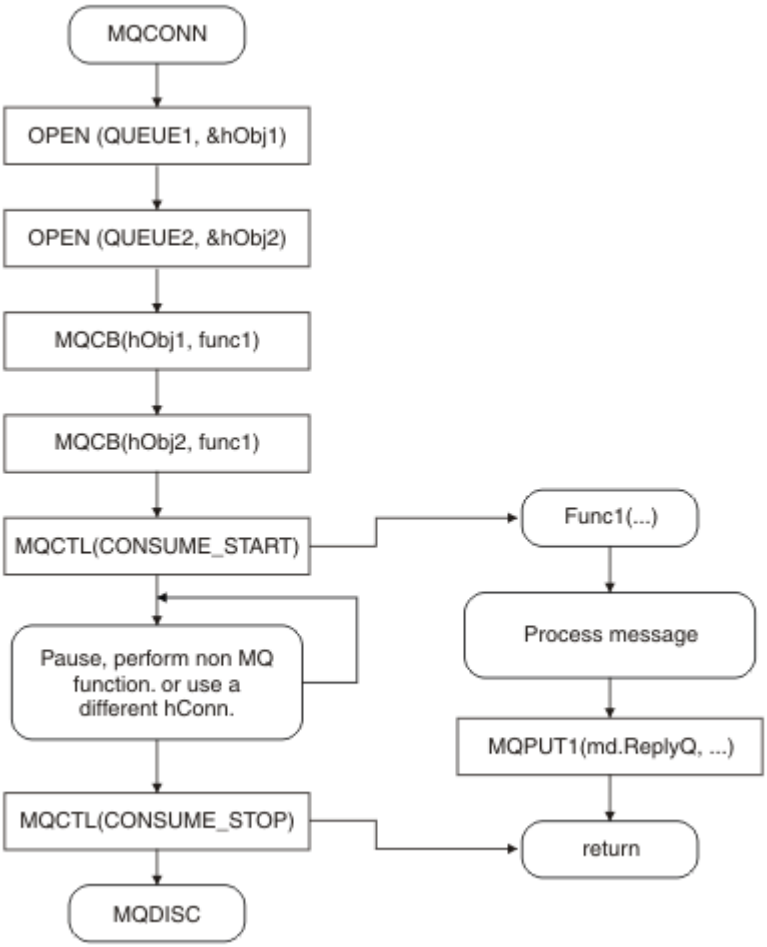


Figure 5. Application gérée par message standard consommant à partir de deux files d'attente

Figure 6, à la page 36 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.

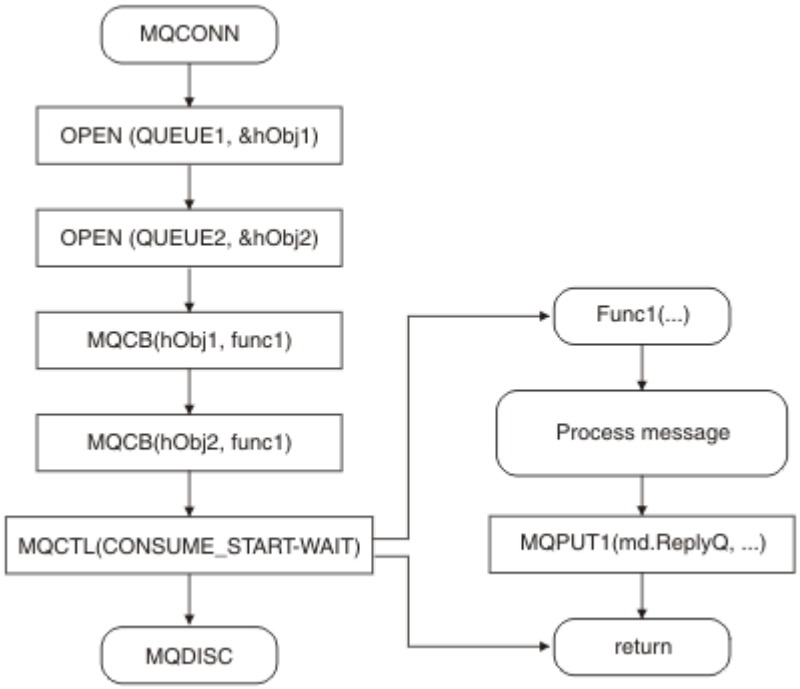


Figure 6. Application gérée par message à unité d'exécution unique consommant à partir de deux files d'attente

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 256). Sur les plateformes autres que z/OS, concept connexe, «Segmentation des messages», à la page 274 permet de fractionnés les messages volumineux en segments plus petits. Vous ne pouvez pas utiliser des 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 36 présente un groupe de messages logiques:

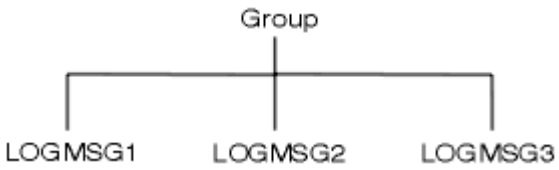


Figure 7. Groupe de messages logiques

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 avoir 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 274.

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 37 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 *SegmentFlag* 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 37 présente un groupe de messages logiques, dont certains sont segmentés:

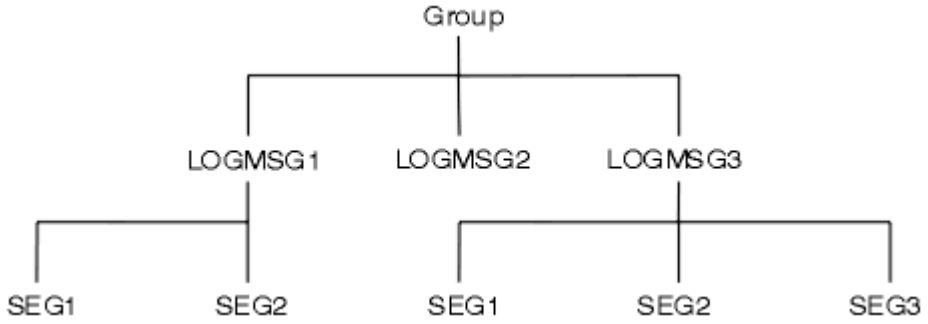


Figure 8. Messages segmentés

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

Pour obtenir une description des messages logiques et physiques, voir «Ordre logique et physique», à la page 256. Pour plus d'informations sur la segmentation des messages, voir «Segmentation des messages», à la page 274.

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.

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 335.
- Sur les systèmes IBM WebSphere MQ sous UNIX , IBM WebSphere MQ sur les systèmes Linux , et IBM WebSphere MQ for Windows, 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», à la page 569.

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.

Sur WebSphere MQ for WebSphere MQ for Windows, WebSphere MQ sur les systèmes UNIX , WebSphere MQ sur les systèmes Linux le nombre d'annulations survit toujours aux redémarrages du gestionnaire de files d'attente. Toute modification apportée à l'attribut *HardenGetBackout* est ignorée.

Pour plus d'informations sur la validation et l'annulation des messages, voir «Validation et annulation d'unités de travail», à la page 335.

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 572).

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», à la page 242.

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 est stocké dans la zone *AccountingToken* lorsqu'un message est créé sous WebSphere 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	Type d'application qui a inséré le message (par exemple, une transaction CICS).
PutApplName	Nom de l'application qui a inséré le message (par exemple, le nom d'un travail ou d'une transaction).
PutDate	Date à laquelle le message a été inséré dans la file d'attente.
PutTime	Heure à laquelle le message a été inséré dans la file d'attente.
ApplOriginData	Toute information supplémentaire 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 WebSphere MQ

Ces informations fournissent des détails sur les objets WebSphere MQ qui incluent 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 WebSphere 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 WebSphere 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», à la page 242
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 232
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.

Préparation et exécution des applications Microsoft Transaction Server

Pour préparer une application MTS à s'exécuter en tant qu'application client WebSphere MQ MQI, suivez les instructions ci-dessous en fonction de votre environnement.

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

Pour préparer une application MTS à s'exécuter en tant qu'application client WebSphere MQ MQI, 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 sous Windows», à la page 477 mais liez le composant à la bibliothèque mqicxa.lib à la place de mqic.lib.
- Si le composant utilise les classes C++ WebSphere MQ , suivez les instructions de la rubrique «Génération de programmes C++ sous Windows», à la page 677 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 sous Windows», à la page 481 , mais lorsque vous définissez le projet Visual Basic, entrez `MqType=3` dans la zone **Arguments de compilation conditionnels** .
- Si le composant utilise WebSphere MQ Automation Classes for ActiveX (MQAX), définissez une variable d'environnement, GMQ_MQ_LIB, avec la valeur `mqic32xa.dll` .

  Vous pouvez définir la variable d'environnement à partir de votre application ou vous pouvez la définir de sorte que sa portée soit à l'échelle du système. Toutefois, sa définition en tant que système peut

entraîner un comportement incorrect de toute application MQAX existante, qui ne définit pas la variable d'environnement à partir de l'application.

Utilisation de IBM WebSphere MQ avec WebSphere Application Server

Utilisez cette rubrique pour comprendre l'utilisation de IBM WebSphere MQ avec WebSphere Application Server.

Les applications écrites en Java qui s'exécutent sous WebSphere Application Server peuvent utiliser la spécification JMS (Java Messaging Service) pour effectuer des opérations de messagerie. La messagerie point-à-point dans cet environnement peut être fournie par un gestionnaire de files d'attente IBM WebSphere MQ

L'avantage d'utiliser un gestionnaire de files d'attente IBM WebSphere MQ pour fournir la messagerie point-à-point est que la connexion d'applications JMS peut participer pleinement aux fonctionnalités d'un réseau IBM WebSphere MQ , qui permet aux applications d'échanger des messages avec des gestionnaires de files d'attente s'exécutant 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. Si le gestionnaire de files d'attente n'est pas local pour l'application, la *connexion client* doit être installée pour permettre à l'application de se connecter à un gestionnaire de files d'attente s'exécutant sur une autre machine ou image.

Par défaut, les messages JMS détenus dans les files d'attente IBM WebSphere MQ utilisent un en-tête MQRFH2 pour contenir certaines informations d'en-tête de message JMS. De nombreuses applications IBM WebSphere MQ existantes ne peuvent pas traiter les messages avec ces en-têtes et requièrent leurs propres en-têtes de caractéristique, par exemple MQCIH pour CICS Bridge ou MQWIH pour les applications de flux de travaux IBM WebSphere MQ . Pour plus de détails sur ces considérations spéciales, voir «Mappage de messages JMS vers des messages WebSphere MQ», à la page 839.

Scénarios de support transactionnel

En utilisant un support transactionnel, vous pouvez activer les applications pour qu'elles fonctionnent efficacement avec les bases de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Cette section présente le support transactionnel. Le travail requis pour permettre à vos applications d'utiliser IBM WebSphere MQ avec un produit de base de données porte sur les domaines de la programmation d'application et de l'administration de système. Utilisez les informations ici avec «Validation et annulation d'unités de travail», à la page 335.

Nous allons commencer par présenter les unités d'oeuvre qui composent les transactions, puis nous décrirons les façons dont vous permettez à IBM WebSphere MQ de coordonner les transactions avec les bases de données.

Concepts associés

«Présentation des unités d'oeuvre», à la page 42
Cette rubrique présente et définit les concepts généraux d'unité de travail, de validation, d'annulation et de point de synchronisation. Elle contient également deux scénarios illustrant les unités d'oeuvre globales.

IBM WebSphere MQ et HP NonStop TMF

Présentation des unités d'oeuvre

Cette rubrique présente et définit les concepts généraux d'unité de travail, de validation, d'annulation et de point de synchronisation. Elle contient également deux scénarios illustrant les unités d'oeuvre globales.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Lorsqu'un programme insère des messages dans des files d'attente au sein d'une unité d'oeuvre, ces messages ne deviennent visibles pour d'autres programmes que 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 de ne pas rendre l'opération d'insertion permanente, il peut *annuler* l'unité d'oeuvre. Lorsqu'un programme effectue une annulation, WebSphere MQ restaure les files d'attente en supprimant les messages qui ont été placés dans les files d'attente par cette unité d'oeuvre.

De même, lorsqu'un programme reçoit des messages d'une ou plusieurs files d'attente dans une unité d'oeuvre, ces messages restent dans les files d'attente jusqu'à ce que le programme valide l'unité d'oeuvre, mais les messages ne sont pas disponible pour être extraits par d'autres programmes. Les messages sont définitivement supprimés des files d'attente lorsque le programme valide l'unité d'oeuvre. Si le programme annule l'unité de travail, WebSphere MQ restaure les files d'attente en rendant les messages disponibles pour être extraits par d'autres programmes.

La décision de validation ou d'annulation des modifications est prise, dans le cas le plus simple, à la fin d'une tâche. Toutefois, il peut être plus utile pour une application de synchroniser les modifications de données à d'autres points logiques dans une tâche. Ces points logiques sont appelés points de synchronisation et la période de traitement d'un ensemble de mises à jour entre deux points de synchronisation est appelé *unité d'oeuvre*. Plusieurs appels MQGET et MQPUT peuvent faire partie d'une seule unité d'oeuvre.

Avec WebSphere MQ, il est nécessaire de faire la distinction entre les unités de travail *locales* et *globales* :

Unités d'oeuvre locales
Il s'agit de celles dans lesquelles les seules actions sont des insertions et des extractions de files d'attente WebSphere MQ et où la coordination de chaque unité de travail est assurée dans le gestionnaire de files d'attente à l'aide d'un processus de *validation en une phase* .

Utilisez des unités d'oeuvre locales lorsque les seules ressources à mettre à jour sont les files d'attente gérées par un seul gestionnaire de files d'attente WebSphere MQ . Les mises à jour sont validées à l'aide de l'instruction MQCMIT ou annulées à l'aide de MQBACK.

Aucune tâche d'administration de système autre que la gestion de journal n'est impliquée dans l'utilisation des unités d'oeuvre locales. Dans vos applications, là où vous utilisez les appels MQPUT et MQGET avec MQCMIT et MQBACK, essayez d'utiliser les options MQPMO_SYNCPOINT et MQGMO_SYNCPOINT. (Pour plus d'informations sur la gestion des journaux, voir Gestion des fichiers journaux .)

Unités d'oeuvre globales
Unités d'oeuvre dans lesquelles d'autres ressources, par exemple les tables d'une base de données relationnelle, sont également mises à jour. Lorsque plusieurs *gestionnaires de ressources* sont impliqués, il faut un logiciel de *gestionnaire de transactions* utilisant un processus de *validation en deux phases* pour coordonner l'unité d'oeuvre globale.

Utilisez des unités d'oeuvre globales lorsque vous devez également inclure des mises à jour de logiciels de gestionnaire de base de données relationnelle, tels que Db2, Oracle, Sybaseet Informix.

Il existe plusieurs scénarios possibles pour l'utilisation d'unités d'oeuvre globales. Deux scénarios sont documentés ici :

1. Dans le premier, le gestionnaire de files d'attente tient lieu de gestionnaire de transactions. Dans ce scénario, des instructions MQI contrôlent les unités d'oeuvre globales. Celles-ci sont démarrées dans les applications à l'aide de l'instruction MQBEGIN, puis validées à l'aide de MQCMIT ou annulées à l'aide de MQBACK.
2. Dans la seconde, le rôle de gestionnaire de transactions est assuré par d'autres logiciels, tels que TXSeries, Encinaou Tuxedo. Dans ce scénario, une API fournie par le logiciel du gestionnaire de

transactions est utilisée pour contrôler l'unité de travail (par exemple, EXEC CICS SYNCPOINT for TXSeries).

Les sections suivantes décrivent toutes les étapes nécessaires pour utiliser les unités d'oeuvre globales, organisées par les deux scénarios :

- «Scénario 1 : Le gestionnaire de files d'attente effectue la coordination», à la page 44
- «Scénario 2 : Un autre logiciel assure la coordination», à la page 72

Scénario 1 : Le gestionnaire de files d'attente effectue la coordination

Dans le scénario 1, le gestionnaire de files d'attente tient lieu de gestionnaire de transactions. Dans ce scénario, des instructions MQI contrôlent les unités d'oeuvre globales. Celles-ci sont démarrées dans les applications à l'aide de l'instruction MQBEGIN, puis validées à l'aide de MQCMIT ou annulées à l'aide de MQBACK.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Niveau d'isolement

Dans IBM WebSphere MQ, un message dans une file d'attente peut être visible avant une mise à jour de base de données, en fonction de la conception d'isolement de transaction implémentée dans la base de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Lorsqu'un gestionnaire de files d'attente IBM WebSphere MQ fonctionne en tant que gestionnaire de transactions XA pour coordonner les mises à jour vers des gestionnaires de ressources XA, le protocole de validation ci-après est suivi :

1. Préparer tous les gestionnaires de ressources XA.
2. Validez le gestionnaire de ressources du gestionnaire de files d'attente IBM WebSphere MQ .
3. Valider les autres gestionnaires de ressources.

Entre l'étape 2 et 3, une application peut voir un message qui est validé vers la file d'attente, mais la ligne correspondante dans la base de données ne reflète pas ce message.

Cela ne pose pas de problème si la base de données est configurée de sorte que les appels API de base de données de l'application attendent que les mises à jour en attente soient terminées.

Vous pouvez résoudre cette situation en configurant différemment la base de données. Le type de configuration nécessaire est appelé "niveau d'isolement". Pour plus d'informations sur les niveaux d'isolement, reportez-vous à la documentation de la base de données. Vous pouvez également configurer le gestionnaire de files d'attente pour valider les gestionnaires de ressources dans l'ordre inverse suivant :

1. Préparer tous les gestionnaires de ressources XA.
2. Valider les autres gestionnaires de ressources.
3. Validez le gestionnaire de ressources du gestionnaire de files d'attente IBM WebSphere MQ .

Lorsque vous modifiez le protocole, le gestionnaire de files d'attente IBM WebSphere MQ est validé en dernier, de sorte que les applications qui lisent les messages depuis les files d'attente voient un message uniquement après que la mise à jour de base de données correspondante a été terminée.

Pour configurer le gestionnaire de files d'attente pour utiliser ce protocole modifié, définissez la variable d'environnement **AMQ_REVERSE_COMMIT_ORDER**.

Définissez cette variable d'environnement dans l'environnement depuis lequel la commande **strmqm** est exécutée pour démarrer le gestionnaire de files d'attente. Par exemple, exécutez la commande suivante dans le shell juste avant démarrer le gestionnaire de files d'attente :

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

Remarque : La définition de cette variable d'environnement peut entraîner une entrée de journal supplémentaire par transaction. Cela n'aura donc qu'un impact réduit sur les performances de chaque transaction.

Coordination de base de données

Lorsque le gestionnaire de files d'attente coordonne des unités d'oeuvre globale, il devient possible d'intégrer les mises à jour de base de données dans les unités d'oeuvre. En d'autres termes, une application MQI et SQL mixte peut être écrite, et les instructions MQCMIT et MQBACK peuvent être utilisées pour valider ou annuler ensemble les modifications apportées aux files d'attente et aux bases de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Le gestionnaire de files d'attente effectue ceci en utilisant le protocole de validation en deux phases décrit dans *X/Open Distributed Transaction Processing: XA Specification*. Lorsqu'une unité d'oeuvre doit être validée, le gestionnaire de files d'attente demande d'abord à chaque gestionnaire de base de données participant s'il est prêt à valider ses mises à jour. Toutes les mises à jour de file d'attente et de base de données ne sont validées que si tous les participants, y compris le gestionnaire de files d'attente lui-même, sont prêts à effectuer la validation. Si des participants ne peuvent préparer leurs mises à jour, l'unité d'oeuvre est annulée.

En général, une unité d'oeuvre globale est implémentée dans une application par la méthode suivante (en pseudocode) :

```
MQBEGIN
MQGET (inclure l'indicateur MQGMO_SYNCPOINT dans les options de message)
MQPUT (inclure l'indicateur MQPMO_SYNCPOINT dans les options de message)
SQL INSERT
MQCMIT
```

L'instruction MQBEGIN permet d'indiquer de début d'une unité d'oeuvre globale. L'instruction MQCMIT permet de signaler la fin de l'unité d'oeuvre globale et d'exécuter cette dernière avec tous les gestionnaires de ressources participants à l'aide du protocole de validation en deux phases.

Lorsque l'unité de travail (également appelée *transaction*) est correctement exécutée à l'aide de MQCMIT, toutes les actions effectuées dans cette unité de travail deviennent permanentes ou irréversibles. Si, pour une raison quelconque, l'unité d'oeuvre échoue, toutes les mises à jour sont annulées. Il n'est pas possible de rendre une action d'une unité d'oeuvre permanente alors que d'autres actions sont annulées. C'est le principe d'une unité d'oeuvre : toutes ses actions sont rendues permanentes ou aucune ne l'est.

Remarque :

1. Le programmeur d'application peut forcer l'annulation d'une unité d'oeuvre en appelant MQBACK. L'unité d'oeuvre est également annulée par le gestionnaire de files d'attente si l'application ou la base de données *échoue* avant l'appel de MQCMIT.
2. Si une application appelle MQDISC sans appeler MQCMIT, le gestionnaire de files d'attente se comporte comme si MQCMIT avait été appelé et valide l'unité d'oeuvre.

Entre MQBEGIN et MQCMIT, le gestionnaire de files d'attente n'effectue pas d'appels vers la base de données pour mettre à jour ses ressources. En d'autres termes, les tables d'une base de données peuvent uniquement être modifiées par votre code (par exemple, l'instruction SQL INSERT dans le pseudocode).

Un support de reprise complet est fourni si le gestionnaire de files d'attente perd le contact avec l'un des gestionnaires de base de données pendant le protocole de validation. Si un gestionnaire de base de données devient indisponible pendant qu'il est en attente de validation (il a été correctement préparé pour la validation, mais n'a encore pour reçu une décision de validation ou d'annulation), le gestionnaire de files d'attente mémorise le résultat de l'unité d'oeuvre jusqu'à ce que ce résultat ait été distribué avec succès à la base de données. De même, si le gestionnaire de files d'attente s'arrête avec des opérations de validation incomplètes en attente, ces opérations sont mémorisées pendant le redémarrage du gestionnaire de files d'attente. Si une application s'arrête de manière inattendue, l'intégrité de l'unité d'oeuvre n'est pas compromise, mais le résultat dépend de la phase du processus dans laquelle l'application a pris fin, comme indiqué dans Tableau 5, à la page 46.

Ce qui se produit si la base de données ou le programme d'application échoue est résumé dans les tableaux suivants :

Tableau 4. Ce qui se produit lorsqu'un serveur de base de données échoue

Moment de l'échec	**Résultat**
Avant que l'application appelle MQCMIT.	L'unité d'oeuvre est annulée.
Lors de l'appel d'application à MQCMIT, **avant** que toutes les bases de données aient indiqué qu'elles sont correctement préparées.	L'unité d'oeuvre est annulée avec un code anomalie MQRC_BACKED_OUT.
Lors de l'appel d'application à MQCMIT, **après** que toutes les bases de données ont indiqué qu'elles sont correctement préparées, mais avant qu'elles aient toutes signalé qu'elles ont effectué la validation avec succès.	L'unité d'oeuvre est mise en attente à l'état récupérable par le gestionnaire de files d'attente, avec un code anomalie MQRC_OUTCOME_PENDING.
Lors de l'appel d'application à MQCMIT, **avant** que toutes les bases de données aient indiqué qu'elles ont effectué la validation avec succès.	L'unité d'oeuvre est validée avec un code anomalie MQRC_NONE.
Après que l'application appelle MQCMIT.	L'unité d'oeuvre est validée avec un code anomalie MQRC_NONE.

Tableau 5. Ce qui se produit lorsque un programme d'application échoue

Moment de l'échec	**Résultat**
Avant que l'application appelle MQCMIT.	L'unité d'oeuvre est annulée.
Lors de l'appel d'application à MQCMIT, **avant** que le gestionnaire de files d'attente ait reçu la demande MQCMIT de l'application.	L'unité d'oeuvre est annulée.
Lors de l'appel d'application à MQCMIT, **après** que le gestionnaire de files d'attente a reçu la demande MQCMIT de l'application.	Le gestionnaire de files d'attente tente de valider à l'aide de la validation en deux phases (dans la mesure où les produits de base de données ont exécuté et validé leurs parts de l'unité d'oeuvre).

Dans le cas où le code anomalie renvoyé par MQCMIT est MQRC_OUTCOME_PENDING, l'unité d'oeuvre est mémorisée par le gestionnaire de files d'attente jusqu'à ce qu'il puisse rétablir le contact avec le serveur de base de données et lui demander de valider sa part de l'unité d'oeuvre. Reportez-vous à «Considérations à prendre en compte en cas de perte du contact avec le gestionnaire de ressources XA», à la page 64 pour plus d'informations sur la manière dont la reprise est effectuée et à quel moment.

Le gestionnaire de files d'attente communique avec les gestionnaires de base de données à l'aide de l'interface XA, comme indiqué dans *X/Open Distributed Transaction Processing: XA Specification*. xa_open, xa_start, xa_end, xa_prepare et xa_commit constituent des exemples de ces appels de fonction. Les

termes *gestionnaire de transaction* et *gestionnaire de ressources* ont dans le présent document la même signification que dans la spécification XA.

Restrictions
Il existe des restrictions quant à la prise en charge de la coordination de base de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les restrictions suivantes s'appliquent :

- La possibilité de coordonner les mises à jour de base de données dans les unités de travail WebSphere MQ n'est **pas** prise en charge dans une application client MQI. L'utilisation de MQBEGIN dans une application client échoue. Un programme qui appelle MQBEGIN doit s'exécuter en tant qu'application *serveur* sur la même machine que le gestionnaire de files d'attente.

  Remarque : Une application *serveur* est un programme qui a été lié aux bibliothèques serveur WebSphere MQ nécessaires ; une application *client* est un programme qui a été lié aux bibliothèques client WebSphere MQ nécessaires. Pour plus d'informations sur la compilation et la liaison de vos programmes, voir «Génération d'applications pour les clients WebSphere MQ MQI», à la page 372 et «Génération d'une application IBM WebSphere MQ», à la page 446 .
- Le serveur de base de données peuvent résider sur une machine différente de celle du serveur de gestionnaire de files d'attente, dans la mesure où la base de données client est installée sur le même machine que le gestionnaire de files d'attente et où elle prend en charge cette fonction. Consultez la documentation du produit de base de données pour déterminer si son logiciel client peut être utilisé pour les systèmes de validation en deux phases.
- Même si le gestionnaire de files d'attente se comporte comme un gestionnaire de ressources (pour pouvoir être impliqué dans les unités d'oeuvre globales du scénario 2), il n'est pas possible de faire en sorte qu'un gestionnaire de files d'attente coordonne un autre gestionnaire de files d'attente dans ses unités d'oeuvre globales du scénario 1.

Fichiers de commutation de chargement

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Un fichier de commutation de chargement est une bibliothèque partagée (une bibliothèque de liaison dynamique ou DLL sur des systèmes Windows) qui est chargée par le code dans votre application IBM WebSphere MQ et le gestionnaire de files d'attente. Il sert à simplifier le chargement de la bibliothèque partagée du client de la base de données et à renvoyer les pointeurs aux fonctions XA.

Les détails du fichier de commutation de chargement doivent être spécifiés avant que le gestionnaire de files d'attente soit démarré. Les détails sont placés dans le fichier qm.ini sur les systèmes Windows, UNIX and Linux .

- Sur les systèmes Windows et Linux (plateformesx86 et x86-64), utilisez le fichier IBM WebSphere MQ Explorer pour mettre à jour le fichier qm.ini .
- Sur tous les autres systèmes, éditez le directement le fichier qm.ini.

La source C pour le fichier de commutation de chargement est fournie avec l'installation IBM WebSphere MQ si elle prend en charge les unités d'oeuvre globales du scénario 1. La source contient une fonction appelée MQStart. Lorsque le fichier de commutation est chargé, le gestionnaire de files d'attente appelle cette fonction qui renvoie l'adresse d'une structure appelée *commutateur XA*.

La structure de commutateur XA existe dans la bibliothèque partagée du client de base de données et contient un certain nombre de pointeurs de fonction, comme indiqué dans Tableau 6, à la page 48 :

Tableau 6. Pointeurs de fonction du commutateur XA		
nom du pointeur de fonction	**Fonction XA**	**Objet**
xa_open_entry	xa_open	Se connecter à la base de données
xa_close_entry	xa_close	Se déconnecter de la base de données
xa_start_entry	xa_start	Démarrer une branche d'une unité d'oeuvre globale
xa_end_entry	xa_end	Suspendre une branche d'une unité d'oeuvre globale
xa_rollback_entry	xa_rollback	Annuler une branche d'une unité d'oeuvre globale
xa_prepare_entry	xa_prepare	Se préparer à valider une unité d'oeuvre globale
xa_commit_entry	xa_commit	Valider une branche d'une unité d'oeuvre globale
xa_recover_entry	xa_recover	Découvrir depuis la base de données si celle-ci a une unité d'oeuvre en attente de validation
xa_forget_entry	xa_forget	Autoriser une base de données à oublier une branche d'une unité d'oeuvre globale
xa_complete_entry	xa_complete	Terminer une branche d'une unité d'oeuvre globale

Lors du premier appel MQBEGIN dans votre application, le code IBM WebSphere MQ qui s'exécute dans le cadre de MQBEGIN charge le fichier de commutation de chargement et appelle la fonction xa_open dans la bibliothèque partagée de la base de données. De même, lors du démarrage du gestionnaire de files d'attente et dans d'autres situations ultérieures, certains processus de gestionnaire de files d'attente chargent le fichier de commutation de chargement et appellent xa_open.

Vous pouvez réduire le nombre d'appels xa_* en utilisant l'*enregistrement dynamique*. Pour obtenir une description complète de cette technique d'optimisation, voir «Enregistrement dynamique XA», à la page 69.

Configuration de votre système pour la coordination de base de données

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Vous devez effectuer plusieurs tâches pour qu'un gestionnaire de base de données puisse participer à des unités d'oeuvre globales coordonnées par le gestionnaire de files d'attente. Il s'agit des tâches suivantes :

- «Installation et configuration du produit de base de données», à la page 49
- «Création de fichiers de commutation de chargement», à la page 49
- «Ajout d'informations de configuration au gestionnaire de files d'attente», à la page 50
- «Ecriture et modification de vos applications», à la page 52
- «Test du système», à la page 52

Installation et configuration du produit de base de données
Pour installer et configurer votre produit de base de données, reportez-vous à la documentation de votre produit. Les rubriques de cette section décrivent les problèmes de configuration générale et la manière dont ils sont liés à l'interopérabilité entre WebSphere MQ et la base de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Connexions de base de données

Une application qui établit une connexion standard au gestionnaire de files d'attente est associée à une unité d'exécution dans un processus d'agent de gestionnaire de files d'attente local distinct. (Une connexion qui n'est pas une connexion de *raccourci* est une connexion *standard* dans ce contexte. Pour plus d'informations, voir «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 217.)

Lorsque l'application émet `MQBEGIN`, elle et le processus de l'agent appellent la fonction xa_open dans la bibliothèque du client de base de données. En réponse à ceci, le code de bibliothèque du client de base de données se *connecte* à la base de données impliquée dans l'unité d'oeuvre *à partir des processus d'application et de gestionnaire de files d'attente*. Ces connexions de base de données sont conservées tant que l'application reste connectée au gestionnaire de files d'attente.

Ceci est un point important à prendre en compte si la base de données n'accepte qu'un nombre limité d'utilisateurs ou de connexions, car deux connexions sont établies avec la base de données pour prendre en charge un seul programme d'application.

Configuration client-serveur

La bibliothèque du client de base de données chargée dans le WebSphere MQ gestionnaire de files d'attente et les processus d'application **doit** pouvoir envoyer des données à son serveur et en recevoir. Assurez-vous que :

- Les fichiers de configuration client-serveur de la base de données disposent des informations correctes
- Les variables d'environnement appropriées sont définies dans l'environnement des processus de gestionnaire de files d'attente **et** d'application

Création de fichiers de commutation de chargement
WebSphere MQ est fourni avec un exemple de fichier makefile, utilisé pour générer des fichiers de commutation de chargement pour les gestionnaires de base de données pris en charge.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Le fichier makefile exemple, avec tous les fichiers source C requis pour générer les fichiers de commutation de chargement, est installé dans les répertoires suivants :

- Pour WebSphere MQ for Windows, dans le répertoire *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm\`
- Pour les systèmes WebSphere MQ for UNIX and Linux , dans le répertoire *MQ_INSTALLATION_PATH*`/samp/xatm/`

Les modules source exemple utilisés pour générer les fichiers de commutation de chargement sont les suivants :

- Pour DB2, db2swit.c

- Pour Oracle, oraswit.c
- Pour Informix, infswit.c
- Pour Sybase, sybswit.c

Lorsque vous générez des fichiers de commutation de chargement, installez les fichiers 32 bits dans `/var/mqm/exits` et les fichiers 64 bits dans `/var/mqm/exits64`.

Si vous disposez de gestionnaires de files d'attente 32 bits, le fichier makefile exemple, xaswit.mak, installe un fichier de commutation de chargement 32 bits dans `/var/mqm/exits`.

Si vous disposez de gestionnaires de files d'attente 64 bits, le fichier makefile exemple, xaswit.mak, installe un fichier de commutation de chargement 32 bits dans `/var/mqm/exits` et un fichier de commutation de chargement 64 bits dans `/var/mqm/exits64`.

Sécurité de fichier

Il est possible que votre système d'exploitation échoue au chargement du fichier de commutation de chargement par WebSphere MQ, pour des raisons qui échappent au contrôle de WebSphere MQ. Si cela se produit, des messages d'erreur sont écrits dans les journaux d'erreurs WebSphere MQ et l'appel MQBEGIN peut éventuellement échouer. Pour aider votre système d'exploitation à réussir le chargement du fichier de commutation de chargement, vous devez respecter les conditions suivantes :

1. Le fichier de commutation de chargement doit être disponible dans l'emplacement indiqué dans le fichier qm.ini.
2. Le fichier de commutation de chargement doit être accessible pour tous les processus devant le charger, y compris les processus de gestionnaire de files d'attente et les processus d'application.
3. Toutes les bibliothèques desquelles dépend le fichier de commutation de chargement, y compris les bibliothèques qui sont fournies par le produit de base de données, doit être présentes et accessibles.

Ajout d'informations de configuration au gestionnaire de files d'attente
Une fois que vous avez créé un fichier de commutation de chargement pour votre gestionnaire de base de données et que vous l'avez placé dans un emplacement sûr, vous devez indiquer cet emplacement à votre gestionnaire de files d'attente.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Pour spécifier l'emplacement, procédez comme suit :

- Sous Windows et Linux (plateformesx86 et x86-64), les systèmes utilisent WebSphere 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.
- Sur tous les autres systèmes, spécifiez les détails du fichier de commutation de chargement dans la strophe XAResourceManager du fichier qm.ini du gestionnaire de files d'attente.

Ajoutez une strophe XAResourceManager pour la base de données que votre gestionnaire de files d'attente va coordonner. Dans le scénario le plus courant, il existe une seule base de données et donc une seule strophe XAResourceManager. Pour des détails sur des configurations plus compliquées impliquant plusieurs bases de données, voir «Configurations à bases de données multiples», à la page 63. Les attributs de la strophe XAResourceManager sont les suivants :

Name=nom
Chaîne choisie par l'utilisateur identifiant le gestionnaire de ressources. Cet attribue affecte un nom à la strophe XAResourceManager. Le nom est obligatoire et peut comprendre jusqu'à 31 caractères.

Le nom que vous choisissez doit être unique ; une seule strophe XAResourceManager avec ce nom doit figurer dans ce fichier qm.ini. Le nom doit également être significatif, car il est utilisé par le gestionnaire de files d'attente pour faire référence à ce gestionnaire de ressources dans les messages de journal des erreurs de gestionnaire de ressources et dans la sortie lorsque la commande

dspmqtrn est utilisée. (Voir «Affichage des unités d'oeuvre en attente avec la commande dspmqtrn», à la page 65 pour plus d'informations.)

Une fois que vous avez choisi un nom et démarré le gestionnaire de files d'attente, ne modifiez pas l'attribut Name. Pour plus de détails sur la modification des informations de configuration, voir «Modification des informations de configuration», à la page 68.

SwitchFile=nom
Il s'agit du nom du fichier de commutation de chargement XA que vous avez créé précédemment. Cet attribut est obligatoire. Le code du gestionnaire de files d'attente et des processus d'application WebSphere MQ tente de charger le fichier de commutation de chargement à deux reprises:

1. Au démarrage du gestionnaire de files d'attente
2. Lorsque vous effectuez le premier appel à MQBEGIN dans votre processus d'application WebSphere MQ

Les attributs de sécurité et de droits d'accès du fichier de commutation de chargement doivent autoriser ces processus à exécuter cette action.

XAOpenString=chaîne
Il s'agit d'une chaîne de données que le code WebSphere MQ transmet dans ses appels à la fonction xa_open du gestionnaire de base de données. Cet attribut est facultatif. S'il est omis, une chaîne de longueur zéro est utilisée.

Le code du gestionnaire de files d'attente et des processus d'application WebSphere MQ appelle la fonction xa_open à deux reprises:

1. Au démarrage du gestionnaire de files d'attente
2. Lorsque vous effectuez le premier appel à MQBEGIN dans votre processus d'application WebSphere MQ

Le format de cette chaîne est spécifique à chaque produit de base de données et décrit dans la documentation de ce produit. En général, la chaîne xa_open contient des informations d'authentification (nom d'utilisateur et mot de passe) pour autoriser une connexion à la base de données dans les processus de gestionnaire de files d'attente et d'application.

XACloseString=chaîne
Il s'agit d'une chaîne de données que le code WebSphere MQ transmet dans ses appels à la fonction xa_close du gestionnaire de base de données. Cet attribut est facultatif. S'il est omis, une chaîne de longueur zéro est utilisée.

Le code du gestionnaire de files d'attente et des processus d'application WebSphere MQ appelle la fonction xa_close à deux reprises:

1. Au démarrage du gestionnaire de files d'attente
2. Lorsque vous effectuez un appel à MQDISC dans votre processus d'application WebSphere MQ , après avoir précédemment effectué un appel à MQBEGIN.

Le format de cette chaîne est spécifique à chaque produit de base de données et décrit dans la documentation de ce produit. En général, la chaîne est vide et l'attribut XACloseString est souvent omis dans la strophe XAResourceManager.

ThreadOfControl=THREAD |PROCESS
La valeur de ThreadOfControl peut être THREAD ou PROCESS. Le gestionnaire de files d'attente utilise cette valeur à des fins de sérialisation. Cet attribut est facultatif. S'il est omis, la valeur PROCESS est utilisée.

Si le code du client de base de données permet aux unités d'exécution d'appeler les fonctions XA sans sérialisation, la valeur de ThreadOfControl peut être THREAD. Le gestionnaire de files d'attente suppose qu'il peut appeler les fonctions XA dans la bibliothèque partagée du client de base de données à partir de plusieurs unités d'exécution en même temps si nécessaire.

Si le code du client de base de données n'autorise pas les unités d'exécution à appeler ses fonctions XA de cette façon, la valeur de ThreadOfControl doit être PROCESS. Dans ce cas, le gestionnaire de files d'attente sérialise tous les appels à la bibliothèque partagée du client de base de données pour

qu'un seul appel à la fois soit effectué depuis un processus particulier. Vous aurez probablement aussi besoin de vérifier que votre application effectue une sérialisation similaire si elle s'exécute avec plusieurs unités d'exécution.

Notez que la capacité du produit de base de données à traiter les processus à unités d'exécution multiples de cette façon constitue un problème pour le fournisseur de ce produit. Consultez la documentation du produit de base de données pour savoir si vous pouvez définir l'attribut ThreadOfControl sur THREAD ou PROCESS. Nous vous recommandons dans la mesure du possible de définir ThreadOfControl sur THREAD. En cas de doute, l'option la *plus sûre* consiste à définir ThreadOfControl sur PROCESS, mais vous perdrez les avantages de performance potentiel apportés par l'utilisation de THREAD.

Ecriture et modification de vos applications
Comment implémenter une unité d'oeuvre globale.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les exemples de programmes d'application pour les unités de travail globales du scénario 1 qui sont fournis avec une installation WebSphere MQ sont décrits dans le manuel «Présentation des unités d'oeuvre», à la page 42.

En général, une unité d'oeuvre globale est implémentée dans une application par la méthode suivante (en pseudocode) :

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

L'instruction MQBEGIN permet d'indiquer de début d'une unité d'oeuvre globale. L'instruction MQCMIT permet de signaler la fin de l'unité d'oeuvre globale et d'exécuter cette dernière avec tous les gestionnaires de ressources participants à l'aide du protocole de validation en deux phases.

Entre MQBEGIN et MQCMIT, le gestionnaire de files d'attente n'effectue pas d'appels vers la base de données pour mettre à jour ses ressources. En d'autres termes, les tables d'une base de données peuvent uniquement être modifiées par votre code (par exemple, l'instruction SQL INSERT dans le pseudocode).

Le rôle du gestionnaire de files d'attente en ce qui concerne la base de données est d'indiquer à celle-ci quand une unité d'oeuvre globale a démarré et a terminé, et si l'unité d'oeuvre globale doit être validée ou annulée.

En ce qui concerne votre application, le gestionnaire de files d'attente exécute deux rôles : il tient lieu de gestionnaire de ressources (les ressources étant des messages dans les files d'attente) et de gestionnaire de transactions pour l'unité d'oeuvre globale.

Commencez par utiliser les exemples de programmes fournis et parcourez les différents WebSphere MQ et les appels d'API de base de données effectués dans ces programmes. Les appels d'API concernés sont intégralement documentés dans «Exemples de programmes WebSphere MQ», à la page 100, Types de données utilisés dans l'interface MQIet (dans le cas de la propre API de la base de données) dans la documentation de la base de données.

Test du système
Le seul moyen de savoir si votre application et votre système sont correctement configurés est de les exécuter lors d'un test. Vous pouvez tester la configuration du système (communication réussie entre le gestionnaire de files d'attente et la base de données) en générant et en exécutant l'un des exemples de programme fournis.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la

version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Configuration de Db2

Informations de configuration et de prise en charge de DB2 .

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les niveaux pris en charge de Db2 sont définis sur la page IBM WebSphere MQ -Configuration système requise détaillée .

Remarque : Les instances 32 bits de Db2 ne sont pas prises en charge sur les plateformes où le gestionnaire de files d'attente est 64 bits.

Procédez comme suit :

1. Vérifiez les paramètres de variable d'environnement.
2. Créez le fichier de commutation de chargement Db2 .
3. Ajoutez les informations de configuration de gestionnaire de ressources.
4. Modifiez les paramètres de configuration Db2 si nécessaire.

Lisez ces informations en tenant compte des informations générales fournies dans «Configuration de votre système pour la coordination de base de données», à la page 48.

Avertissement : Si vous exécutez `db2profile` sur des plateformes UNIX and Linux, les variables d'environnement LIBPATH et LD_LIBRARY_PATH sont définies. Il est conseillé de `unset` ces variables d'environnement, voir le guide *Guide d'initiation* approprié.

Vérification des paramètres des variables d'environnement Db2

Vérifiez que vos variables d'environnement Db2 sont définies pour les processus de gestionnaire de files d'attente ***ainsi que dans*** vos processus d'application. En particulier, vous devez toujours définir la variable d'environnement DB2INSTANCE ***avant de*** démarrer le gestionnaire de files d'attente. La variable d'environnement DB2INSTANCE identifie l'instance Db2 contenant les bases de données Db2 en cours de mise à jour. Exemple :

- Sur les systèmes UNIX and Linux , utilisez:

```
export DB2INSTANCE=db2inst1
```

- Sur les systèmes Windows , utilisez:

```
set DB2INSTANCE=DB2
```

Sous Windows avec une base de données Db2 , vous devez ajouter l'utilisateur MUSR_MQADMIN au groupe DB2USERS pour permettre au gestionnaire de files d'attente de démarrer.

Création du fichier de commutation de chargement Db2

La méthode la plus simple pour créer le fichier de commutation de chargement Db2 consiste à utiliser l'exemple de fichier xaswit.mak, fourni par WebSphere MQ pour générer les fichiers de commutation de chargement pour divers produits de base de données.

Sur les systèmes Windows , vous pouvez trouver xaswit.mak dans le répertoire *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm`. *MQ_INSTALLATION_PATH* représente le

répertoire de haut niveau dans lequel WebSphere MQ est installé. Pour créer le fichier de commutation de chargement Db2 avec Microsoft Visual C + +, utilisez:

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

Le fichier de commutation généré est placé dans `c:\Program Files\IBM\WebSphere MQ\exits`.

Vous trouverez xaswit.mak dans le répertoire *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Editez xaswit.mak pour *supprimer la mise en commentaire* des lignes appropriées à la version de Db2 que vous utilisez. Exécutez ensuite le fichier makefile à l'aide de la commande :

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

Le fichier de commutation de chargement 32 bits généré est placé dans `/var/mqm/exits`.

Le fichier de commutation de chargement 64 bits généré est placé dans `/var/mqm/exits64`.

Ajout d'informations de configuration de gestionnaire de ressources pour Db2

Vous devez modifier les informations de configuration pour que le gestionnaire de files d'attente déclare Db2 comme participant dans les unités d'oeuvre globales. Cette modification des informations de configuration est décrite plus en détails dans «Ajout d'informations de configuration au gestionnaire de files d'attente», à la page 50.

- Sur les systèmes Windows et Linux (plateformesx86 et x86-64), utilisez WebSphere 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.
- Sur tous les autres systèmes, spécifiez les détails du fichier de commutation de chargement dans la strophe XAResourceManager du fichier qm.ini du gestionnaire de files d'attente.

Figure 9, à la page 54 est un exemple UNIX illustrant une entrée XAResourceManager où la base de données à coordonner est appelée `mydbname`, ce nom étant spécifié dans `XAOpenString`:

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

Figure 9. Exemple d'entrée XAResourceManager pour Db2 sur les plateformes UNIX

Remarque :

1. `ThreadOfControl=THREAD` ne peut pas être utilisé avec des versions de Db2 antérieures à la version 8. Définissez `ThreadOfControl` et le paramètre XAOpenString `toc` sur l'une des combinaisons suivantes :
   - `ThreadOfControl=THREAD` et `toc=t`
   - `ThreadOfControl=PROCESS` et `toc=p`

   Si vous utilisez le fichier de commutation de chargement XA jdbcdb2 pour permettre la coordination JDBC/JTA, vous devez utiliser `ThreadOfControl=PROCESS` et `toc=p`.

Modification des paramètres de configuration Db2

Pour chaque base de données Db2 coordonnée par le gestionnaire de files d'attente, vous devez définir des privilèges de base de données, modifier le paramètre tp_mon_name et réinitialiser le paramètre maxappls. Pour ce faire, procédez comme suit :

Définir les privilèges d'accès à la base de données

Les processus de gestionnaire de files d'attente s'exécutent avec l'utilisateur et le groupe effectifs mqm sur des systèmes UNIX and Linux. Sur les systèmes Windows , ils s'exécutent en tant qu'utilisateur ayant démarré le gestionnaire de files d'attente. Il peut s'agir de :

1. L'utilisateur ayant émis la commande `strmqm`, ou
2. Utilisateur sous lequel le serveur IBM MQSeries Service COM s'exécute

Par défaut, cet utilisateur est appelé MUSR_MQADMIN.

Si vous n'avez pas spécifié de nom d'utilisateur et de mot de passe dans la chaîne xa_open, **l'utilisateur sous lequel le gestionnaire de files d'attente exécute** est utilisé par Db2 pour authentifier l'appel xa_open. Si cet utilisateur (par exemple, l'utilisateur mqm sur des systèmes UNIX and Linux) ne dispose pas des privilèges minimum dans la base de données, cette dernière refuse d'authentifier l'appel xa_open.

Les mêmes considérations s'appliquent à votre processus d'application. Si vous n'avez pas spécifié de nom d'utilisateur et de mot de passe sur la chaîne xa_open, l'utilisateur sous lequel votre application s'exécute est utilisé par Db2 pour authentifier l'appel xa_open effectué lors du premier appel MQBEGIN. Là encore, cet utilisateur doit disposer de privilèges minimum dans la base de données pour que cela fonctionne.

Par exemple, accordez à l'utilisateur mqm le droit de connexion dans la base de données mydbname en exécutant les commandes Db2 suivantes:

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

Pour plus d'informations sur la sécurité, voir «Sécurité», à la page 64.

Windows **Modifiez le paramètre TP_MON_NAME**

Pour les systèmes Db2 for Windows uniquement, modifiez le paramètre de configuration TP_MON_NAME pour nommer la DLL utilisée par Db2 pour appeler le gestionnaire de files d'attente pour l'enregistrement dynamique.

Utilisez la commande `db2 update dbm cfg using TP_MON_NAME mqmax` pour nommer MQMAX.DLL comme bibliothèque utilisée par Db2 pour appeler le gestionnaire de files d'attente. Elle doit être présente dans un répertoire situé dans PATH.

Redéfinir le paramètre maxappls

Vous pouvez avoir besoin de modifier le paramètre *maxappls* qui limite le nombre maximal d'applications pouvant être connectées à une base de données. Voir «Installation et configuration du produit de base de données», à la page 49.

Configuration d'Oracle

Prise en charge d'Oracle et informations de configuration

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Procédez comme suit :

1. Vérifiez les paramètres de variable d'environnement.
2. Créez le fichier de commutation de chargement Oracle.

3. Ajoutez les informations de configuration de gestionnaire de ressources.
4. Modifiez les paramètres de configuration Oracle si nécessaire.

Une liste actualisée des niveaux d'Oracle pris en charge par IBM WebSphere MQ est fournie à la page IBM WebSphere MQ -Configuration système requise détaillée.

Vérification des paramètres de variable d'environnement Oracle

Assurez-vous que vos variables d'environnement Oracle sont définies pour les processus de gestionnaire de files d'attente, ainsi que dans vos processus d'application. En particulier, définissez toujours les variables d'environnement suivantes avant de démarrer le gestionnaire de files d'attente :

ORACLE_HOME
Répertoire de base Oracle. Par exemple, sur les systèmes UNIX and Linux , utilisez:

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

Sur les systèmes Windows , utilisez:

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

ORACLE_SID
Identificateur de sécurité (SID) Oracle utilisé. Si vous utilisez Net8 pour la connectivité client-serveur, vous pouvez ne pas avoir besoin de définir cette variable d'environnement. Consultez votre documentation Oracle.

Voici un exemple de définition de cette variable d'environnement sur des systèmes UNIX and Linux :

```
export ORACLE_SID=sid1
```

Sur les systèmes Windows , l'équivalent est:

```
set ORACLE_SID=sid1
```

Remarque : La variable d'environnement PATH doit être définie pour inclure le répertoire des fichiers binaires (par exemple `ORACLE_INSTALL_DIR/VERSION/32BIT_NAME/bin` ou `ORACLE_INSTALL_DIR/VERSION/64BIT_NAME/bin`), sinon vous pouvez voir un message indiquant que les bibliothèques de clients Oracle sont absentes de la machine.

Si vous exécutez des gestionnaires de files d'attente sur des systèmes Windows 64 bits, les clients Oracle 64 bits et 32 bits doivent être installés. Vous devez installer les deux clients car le gestionnaire de files d'attente s'exécute en tant que processus 32 bits qui utilisent un fichier de commutation de chargement 32 bits, qui doit à son tour démarrer une dll client Oracle 32 bits.

Le fichier de commutation de chargement, chargé par des gestionnaires de files d'attente 64 bits, doit accéder aux bibliothèques client Oracle 64 bits. Les gestionnaires de files d'attente 32 bits doivent accéder au client Oracle 32 bits lorsque IBM WebSphere MQ s'exécute sur un système Windows 64 bits.

Création du fichier de commutation de chargement Oracle

Pour créer le fichier de commutation de chargement Oracle, utilisez le fichier exemple `xaswit.mak`, fourni par IBM WebSphere MQ pour générer les fichiers de commutation de chargement pour divers produits de base de données. Sur les systèmes Windows , `xaswit.mak` se trouve dans le répertoire `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. Pour créer le fichier de commutation de chargement Oracle avec Microsoft Visual C + +, utilisez: `nmake /f xaswit.mak oraswit.dll`

Le fichier de commutation généré est placé dans *MQ_INSTALLATION_PATH*\exits.*MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.

xaswit.mak se trouve dans le répertoire *MQ_INSTALLATION_PATH*/samp/xatm. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.

Editez xaswit.mak pour supprimer la mise en commentaire des lignes correspondant à la version d'Oracle que vous utilisez. Exécutez ensuite le fichier makefile à l'aide de la commande :

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

Le fichier de commutation de chargement 32 bits généré est placé dans /var/mqm/exits.

Le fichier de commutation de chargement 64 bits généré est placé dans /var/mqm/exits64.

Ajout d'informations de configuration de gestionnaire de ressources pour Oracle

Vous devez modifier les informations de configuration pour le gestionnaire de files d'attente afin de déclarer Oracle comme participant dans des unités d'oeuvre globales. Cette modification des informations de configuration pour le gestionnaire de files d'attente est décrite plus en détails dans «Ajout d'informations de configuration au gestionnaire de files d'attente», à la page 50.

- Sur les systèmes Windows et Linux (plateformesx86 et x86-64), utilisez le IBM WebSphere 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.
- Sur tous les autres systèmes, spécifiez les détails du fichier de commutation de chargement dans la strophe XAResourceManager du fichier qm.ini du gestionnaire de files d'attente.

La Figure 10, à la page 57 est un exemple pour les systèmes UNIX and Linux illustrant une entrée XAResourceManager. Vous devez ajouter un répertoires de journal (LogDir) à la chaîne d'ouverture XA pour que toutes les informations d'erreur et de traçage soient consignées au même endroit.

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

Figure 10. Exemple d'entrée pour Oracle sur des plateformes UNIX and Linux

Remarque :

1. Dans la Figure 10, à la page 57, la chaîne xa_open a été utilisée dans quatre paramètres. Des paramètres supplémentaires peuvent être inclus, comme décrit dans la documentation Oracle.
2. Lorsque vous utilisez le paramètre IBM WebSphere MQ ThreadOfControl=THREAD, vous devez utiliser le paramètre Oracle +threads=true dans la strophe XAResourceManager.

Reportez-vous au *guide du développeur d'application serveur Oracle8* pour plus d'informations sur la chaîne xa_open.

Modification des paramètres de configuration Oracle

Pour chaque base de données Oracle coordonnée par le gestionnaire de files d'attente, vous devez vérifier le nombre maximal de sessions et définir des privilège d'accès à la base de données. Pour ce faire, procédez comme suit :

Vérification du nombre maximal de sessions
Vous pouvez avoir besoin de vérifier vos paramètres LICENSE_MAX_SESSIONS et PROCESSES pour prendre en compte les connexions supplémentaires requises par les processus appartenant au gestionnaire de files d'attente. Pour plus d'informations, voir «Installation et configuration du produit de base de données», à la page 49.

Définir les privilèges d'accès à la base de données
Le nom d'utilisateur Oracle spécifié dans la chaîne xa_open doit disposer des privilèges d'accès à la vue DBA_PENDING_TRANSACTIONS, comme décrit dans la documentation Oracle.

Les privilèges nécessaires peuvent être attribués à l'aide de l'exemple de commande suivant :

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

Configuration d' Informix
Informations de configuration et de support Informix .

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Procédez comme suit :

1. Vérifiez que vous avez installé le SDK client Informix approprié:
   - Les gestionnaires de files d'attente et les applications 32 bits requièrent un SDK client Informix 32 bits.
   - Les gestionnaires de files d'attente et les applications 64 bits requièrent un SDK client Informix 64 bits.
2. Vérifiez que les bases de données Informix sont correctement créées par .
3. Vérifiez les paramètres de variable d'environnement.
4. Générez le fichier de commutation de chargement Informix .
5. Ajoutez les informations de configuration de gestionnaire de ressources.

La liste actualisée des niveaux d' Informix pris en charge par WebSphere MQ est disponible sur la pageIBM WebSphere MQ -Configuration système requise détaillée .

Vérification de la création correcte des bases de données Informix

Chaque base de données Informix devant être coordonnée par un gestionnaire de files d'attente WebSphere MQ doit être créée en spécifiant le paramètre `log` . Exemple :

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

Les gestionnaires de files d'attente WebSphere MQ ne peuvent pas coordonner les bases de données Informix pour lesquelles le paramètre `log` n'est pas spécifié lors de la création. Si un gestionnaire de files d'attente tente de coordonner une base de données Informix pour laquelle le paramètre `log` n'est pas spécifié lors de la création, l'appel xa_open à Informix échoue et un certain nombre d'erreurs FFST sont générées.

Vérification des paramètres des variables d'environnement Informix

Vérifiez que vos variables d'environnement Informix sont définies pour les processus de gestionnaire de files d'attente ***ainsi que dans*** vos processus d'application. En particulier, définissez toujours les variables d'environnement suivantes **avant** de démarrer le gestionnaire de files d'attente :

INFORMIXDIR
Répertoire d'installation du produit Informix .

- Pour les applications UNIX and Linux 32 bits, utilisez la commande suivante :

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

- Pour les applications UNIX and Linux 64 bits, utilisez la commande suivante :

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

- Pour les applications Windows , utilisez la commande suivante:

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

Pour les systèmes dotés de gestionnaires de files d'attente 64 bits qui doivent prendre en charge les applications 32 bits et 64 bits, vous avez besoin des SDK client Informix 32 bits et 64 bits installés. Le fichier makefile exemple, `xaswit.mak` utilisé pour la création d'un fichier de commutation de chargement définit également les deux répertoires d'installation du produit.

INFORMIXSERVER
Nom du serveur Informix . Par exemple, sur les systèmes UNIX and Linux , utilisez:

```
export INFORMIXSERVER=hostname_1
```

Sur les systèmes Windows , utilisez:

```
set INFORMIXSERVER=hostname_1
```

ONCONFIG
Nom du fichier de configuration du serveur Informix . Par exemple, sur les systèmes UNIX and Linux , utilisez:

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

Sur les systèmes Windows , utilisez:

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

Création du fichier de commutation de chargement Informix

Pour créer le fichier de commutation de chargement Informix , utilisez l'exemple de fichier xaswit.mak, fourni par WebSphere MQ pour générer les fichiers de commutation de chargement pour divers produits de base de données. Sur les systèmes Windows , vous pouvez trouver xaswit.mak dans le répertoire *MQ_INSTALLATION_PATH*`\tools\c\samples\xatm`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé. Pour créer le fichier de commutation de chargement Informix avec Microsoft Visual C + +, utilisez:

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

Le fichier de commutation généré est placé dans `c:\Program Files\IBM\WebSphere MQ\exits`.

Vous trouverez xaswit.mak dans le répertoire *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Editez xaswit.mak pour *supprimer la mise en commentaire* des lignes appropriées à la version d' Informix que vous utilisez. Exécutez ensuite le fichier makefile à l'aide de la commande :

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

Le fichier de commutation de chargement 32 bits généré est placé dans `/var/mqm/exits`.

Le fichier de commutation de chargement 64 bits généré est placé dans `/var/mqm/exits64`.

Ajout d'informations de configuration de gestionnaire de ressources pour Informix

Vous devez modifier les informations de configuration pour que le gestionnaire de files d'attente déclare Informix en tant que participant dans des unités d'oeuvre globales. Cette modification des informations de configuration pour le gestionnaire de files d'attente est décrite plus en détails dans «Ajout d'informations de configuration au gestionnaire de files d'attente», à la page 50.

- Sur les systèmes Windows et Linux (plateformesx86 et x86-64), utilisez WebSphere 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.
- Sur tous les autres systèmes, spécifiez les détails du fichier de commutation de chargement dans la strophe XAResourceManager du fichier qm.ini du gestionnaire de files d'attente.

Figure 11, à la page 60 est un exemple UNIX illustrant une entrée qm.ini XAResourceManager où la base de données à coordonner est appelée `mydbname`, ce nom étant spécifié dans le fichier `XAOpenString`:

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

Figure 11. Exemple d'entrée XAResourceManager pour Informix sur les plateformes UNIX

Remarque : Par défaut, l'exemple xaswit.mak sur les plateformes UNIX crée un fichier de commutation de chargement qui utilise des bibliothèques Informix à unités d'exécution. Vous devez vous assurer que le contrôle ThreadOfest défini sur THREAD lors de l'utilisation de ces bibliothèques Informix . Dans la Figure 11, à la page 60, l'attribut ThreadOfControl de la strophe XAResourceManager du fichier qm.ini est défini sur THREAD. Lorsque THREAD est spécifié, les applications doivent être générées à l'aide des bibliothèques Informix à unités d'exécution et des bibliothèques d'API à unités d'exécution WebSphere MQ .

L'attribut XAOpenString doit contenir le nom de la base de données, suivi du symbole @, puis du nom du serveur Informix .

Pour utiliser les bibliothèques Informix non à unités d'exécution, vous devez vous assurer que l'attribut de strophe XAResourceManager du fichier qm.ini ThreadOfControl est défini sur PROCESS. Vous devez également effectuer les modifications suivantes dans le fichier exemple xaswit.mak :

1. Supprimer la mise en commentaire du code de génération d'un fichier de commutation de chargement sans unités d'exécution.
2. Mettre en commentaire le code de génération du fichier de commutation de chargement avec unités d'exécution.

Configuration de Sybase

Prise en charge de Sybase et informations de configuration

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la

version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Procédez comme suit :

1. Vérifiez que vous avez installé les bibliothèques Sybase XA, par exemple, en installant l'option XA DTM.
2. Vérifiez les paramètres de variable d'environnement.
3. Activez la prise en charge de Sybase XA.
4. Créez le fichier de commutation de chargement Sybase.
5. Ajoutez les informations de configuration de gestionnaire de ressources.

La liste actualisée des niveaux de Sybase pris en charge par WebSphere MQ est disponible sur la page IBM WebSphere MQ -Configuration système requise détaillée .

Vérification des paramètres de variable d'environnement Sybase

Assurez-vous que vos variables d'environnement Sybase sont définies pour les processus de gestionnaire de files d'attente ***ainsi que dans*** vos processus d'application En particulier, définissez toujours les variables d'environnement suivantes **avant** de démarrer le gestionnaire de files d'attente :

SYBASE
Emplacement de l'installation du produit Sybase. Par exemple, sur les systèmes UNIX and Linux , utilisez:

```
export SYBASE=/sybase
```

Sur les systèmes Windows , utilisez:

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

SYBASE_OCS
Répertoire sous SYBASE dans lequel vous avez installé les fichiers client Sybase. Par exemple, sur les systèmes UNIX and Linux , utilisez:

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

Sur les systèmes Windows , utilisez:

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

Activation de la prise en charge de Sybase XA

Dans le fichier de configuration Sybase XA, `$SYBASE/$SYBASE_OCS/xa_config`, définissez un gestionnaire de ressources logiques (Logical Resource Manager, LRM) pour chaque connexion au serveur Sybase mis à jour. Un exemple du contenu de `$SYBASE/$SYBASE_OCS/xa_config` est montré dans la Figure 12, à la page 61.

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

Figure 12. Exemple du contenu de $SYBASE/$SYBASE_OCS/xa_config

Création du fichier de commutation de chargement Sybase

Pour créer le fichier de commutation de chargement Sybase , utilisez les exemples de fichier fournis avec WebSphere MQ.Sur les systèmes Windows , vous trouverez xaswit.mak dans le répertoire `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. Pour créer le fichier de commutation de chargement Sybase avec Microsoft Visual C + +, utilisez:

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

Le fichier de commutation généré est placé dans `c:\Program Files\IBM\WebSphere MQ\exits`.

Vous trouverez xaswit.mak dans le répertoire *MQ_INSTALLATION_PATH*`/samp/xatm`. *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Editez xaswit.mak pour *supprimer la mise en commentaire* des lignes correspondant à la version de Sybase que vous utilisez. Exécutez ensuite le fichier makefile à l'aide de la commande :

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

Le fichier de commutation de chargement 32 bits généré est placé dans `/var/mqm/exits`.

Le fichier de commutation de chargement 64 bits généré est placé dans `/var/mqm/exits64`.

Ajout d'informations de configuration de gestionnaire de ressources pour Sybase

Vous devez modifier les informations de configuration pour le gestionnaire de files d'attente afin de déclarer Sybase comme participant dans des unités d'oeuvre globales. Cette modification des informations de configuration est décrite plus en détails dans «Ajout d'informations de configuration au gestionnaire de files d'attente», à la page 50.

- Sur les systèmes Windows et Linux (plateformesx86 et x86-64), utilisez WebSphere 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.
- Sur tous les autres systèmes, spécifiez les détails du fichier de commutation de chargement dans la strophe XAResourceManager du fichier qm.ini du gestionnaire de files d'attente.

La Figure 13, à la page 62 montre un exemple UNIX and Linux qui utilise la base de données associée à la définition de gestionnaire de ressources logiques *lrmname* dans le fichier de configuration Sybase XA, `$SYBASE/$SYBASE_OCS/xa_config`. Incluez un nom de fichier journal si vous voulez que les appels de fonction XA soient consignés :

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

Figure 13. Exemple d'entrée pour Sybase sur des plateformes UNIX and Linux

Utilisation de programmes à unités d'exécutions multiples avec Sybase

Si vous utilisez des programmes à unités d'exécution multiples avec des unités de travail globales WebSphere MQ incorporant des mises à jour de Sybase, vous ***devez*** utiliser la valeur THREAD pour le paramètre ThreadOfControl. Veillez également à lier votre programme (et le fichier de commutation de chargement) aux bibliothèques Sybase autorisant les unités d'exécution multiples (les versions _r). L'utilisation de la valeur THREAD pour le paramètre ThreadOfControl est illustrée dans la Figure 13, à la page 62.

Configurations à bases de données multiples

Si vous souhaitez configurer le gestionnaire de files d'attente pour que les mises à jour de plusieurs bases de données puissent être incluses dans des unités d'oeuvre globales, ajoutez une strophe XAResourceManager pour chaque base de données.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Si les bases de données sont toutes gérées par le même gestionnaire de base de données, chaque strophe définit une base de données distincte. Chaque strophe spécifie le même fichier de commutation (*SwitchFile*), mais le contenu de *XAOpenString* est différent car celui-ci indique le nom de la base de données mise à jour. Par exemple, les sections présentées dans Figure 14, à la page 63 configurent le gestionnaire de files d'attente avec les bases de données Db2 *MQBankDB* et *MQFeeDB* sur les systèmes UNIX and Linux .

Important : il ne peut pas exister plusieurs strophes désignant la même base de données. Cette configuration ne fonctionne pas dans toutes les situations et elle risque d'échouer.

Vous recevrez des erreurs au format `when the MQ code makes its second xa_open call in any process in this environment, the database software fails the second xa_open with a -5 error, XAER_INVAL`.

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

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

Figure 14. Exemples d'entrées XAResourceManager pour plusieurs bases de données Db2

Si les bases de données à mettre à jour sont gérées par des gestionnaires de base de données différents, ajoutez une strophe XAResourceManager pour chacune d'entre elles. Dans ce cas, chaque strophe spécifie un fichier de commutation (*SwitchFile*) différent. Par exemple, si *MQFeeDB* est géré par Oracle au lieu de DB2, utilisez les sections suivantes sur les systèmes UNIX and Linux :

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

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

Figure 15. Exemples d'entrées XAResourceManager pour une base de données DB2 et Oracle

En principe, il n'existe aucune limite quant au nombre d'instances de base de données pouvant être configurées avec un seul gestionnaire de files d'attente.

Remarque : Pour plus d'informations sur la prise en charge de l'inclusion de bases de données Informix dans plusieurs mises à jour de base de données dans des unités d'oeuvre globales, consultez le fichier Readme du produit.

Sécurité

Remarques relatives à l'exécution de votre base de données sous le modèle XA.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les informations suivantes ne constituent que des conseils. Dans tous les cas, consultez la documentation fournie avec le gestionnaire de base de données pour déterminer les implications sur la sécurité de l'exécution de votre base de données sous le modèle XA.

Un processus d'application indique le début d'une unité d'oeuvre globale à l'aide de l'instruction `MQBEGIN`. Le premier appel `MQBEGIN` émis par une application la connecte à toutes les bases de données participantes en appelant leur code de bibliothèque client au point d'entrée xa_open. Tous les gestionnaires de base de données offrent un mécanisme pour fournir un ID utilisateur et un mot de passe à leur chaîne XAOpenString. C'est la seule fois que les informations d'authentification sont transmises.

Notez que, sur les plateformes UNIX and Linux, les applications FASTPATH doivent s'exécuter avec un ID groupe effectif mqm lorsqu'elles effectuent des appels MQI.

Considérations à prendre en compte en cas de perte du contact avec le gestionnaire de ressources XA

Le gestionnaire de files d'attente tolère l'indisponibilité des gestionnaires de base de données. Cela vous permet de démarrer et d'arrêter le gestionnaire de files d'attente indépendamment du serveur de base de données. Lorsque le contact est restauré, le gestionnaire de files d'attente et la base de données se resynchronisent. Vous pouvez également utiliser la commande rsvmqtrn pour résoudre manuellement toutes les unités d'oeuvre en attente de validation.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

En mode de fonctionnement normal, très peu de tâches d'administration sont nécessaires une fois que vous avez terminé les étapes de configuration. Le travail d'administration est facilité du fait que le gestionnaire de files d'attente tolère que des gestionnaires de base de données ne soient pas disponibles. En particulier, cela signifie que :

- Le gestionnaire de files d'attente peut démarrer à tout moment sans démarrer d'abord chaque gestionnaire de base de données.
- Le gestionnaire de files d'attente n'a pas besoin de s'arrêter et redémarrer si l'un des gestionnaires de base de données devient indisponible.

Cela vous permet de démarrer et d'arrêter le gestionnaire de files d'attente indépendamment du serveur de base de données.

Chaque fois que le contact est perdu entre le gestionnaire de files d'attente et une base de données, ceux-ci ont besoin de se resynchroniser dès qu'ils redeviennent disponibles. La resynchronisation est le processus par lequel les unités d'oeuvre en attente de validation impliquant la base de données sont terminées. En général, cela se produit automatiquement sans qu'une intervention de l'utilisateur ne soit nécessaire. Le gestionnaire de files d'attente demande à la base de données la liste des unités d'oeuvre en attente de validation. Il demande ensuite à la base de données de valider ou d'annuler chacune de ces unités d'oeuvre en attente de validation.

Lorsqu'un gestionnaire de files d'attente démarre, il se resynchronise avec chaque base de données. Lorsque une base de données individuelle devient indisponible, seule cette base de données doit être resynchronisée la prochaine fois que le gestionnaire de files d'attente constate qu'elle est à nouveau disponible.

Le gestionnaire de files d'attente rétablit automatiquement le contact avec une base de données précédemment indisponible lorsque de nouvelles unités d'oeuvre globales sont démarrées avec

MQBEGIN. Pour ce faire, il appelle la fonction xa_open dans la bibliothèque de client de base de données. Si cet appel xa_open échoue, MQBEGIN renvoie un code d'achèvement MQCC_WARNING et un code anomalie MQRC_PARTICIPANT_NOT_AVAILABLE. Vous pouvez relancer l'appel MQBEGIN ultérieurement.

Ne continuez pas à lancer une unité d'oeuvre globale qui implique des mises à jour sur une base de données qui a signalé un incident lors de l'appel MQBEGIN. Il n'aura pas de connexion à cette base de données via laquelle les mises à jour peuvent être effectuées. Votre seule option est d'arrêter le programme ou de relancer MQBEGIN périodiquement pour le cas où la base de données serait redevenue disponible.

Vous pouvez également utiliser la commande `rsvmqtrn` pour résoudre explicitement toutes les unités d'oeuvre en attente de validation.

Unités d'oeuvre en attente de validation

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Une base de données peut être laissée avec des unités d'oeuvre en attente de validation si le contact avec le gestionnaire de files d'attente est perdu après que le gestionnaire de base de données a été invité à se préparer. Jusqu'à ce que le serveur de base de données reçoive le résultat à partir du gestionnaire de files d'attente (validation ou annulation), il doit conserver les verrous de base de données associés aux mises à jour.

Comme ces verrous empêchent d'autres applications de se mettre à jour ou de lire des enregistrements de base de données, une resynchronisation doit avoir lieu dès que possible.

Si, pour une raison quelconque, vous ne peut pas attendre que le gestionnaire de files d'attente se resynchronise automatiquement avec la base de données, vous pouvez utiliser des fonctions fournies par le gestionnaire de base de données pour valider ou annuler manuellement les mise à jour de la base de données. Dans le document *X/Open Distributed Transaction Processing: The XA Specification*, cette opération est appelée décision *heuristique*. Utilisez-la uniquement comme dernier recours en raison des risques de compromission de l'intégrité des données. En effet, vous pouvez annuler par erreur les mises à jour de base de données alors que tous les autres participants ont validé leurs mises à jour.

Il est préférable de redémarrer le gestionnaire de files d'attente ou d'utiliser la commande `rsvmqtrn` lorsque la base de données a été redémarrée pour lancer la resynchronisation automatique.

Affichage des unités d'oeuvre en attente avec la commande dspmqtrn

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Pendant qu'un gestionnaire de base de données est indisponible, vous pouvez utiliser la commande **dspmqtrn** pour vérifier l'état des unités d'oeuvre globales en attente impliquant cette base de données.

La commande **dspmqtrn** affiche uniquement les unités d'oeuvre dans lesquelles un ou plusieurs participants sont en attente de validation. Les participants attendent la décision du gestionnaire de files d'attente de valider ou d'annuler les mises à jour préparées.

Pour chacune de ces unités d'oeuvre globales, l'état de chaque participant est affiché dans la sortie de **dspmqtrn**. Si l'unité d'oeuvre n'a pas mis à jour les ressources d'un gestionnaire de ressources particulier, celui-ci n'est pas affiché.

Avec ce qui concerne une unité d'oeuvre en attente de validation, un gestionnaire de ressources a l'un des états suivants :

Préparée
Le gestionnaire de ressources est préparé pour valider ses mises à jour.

Validé
Le gestionnaire de ressources a validé ses mises à jour.

Annulé
Le gestionnaire de ressources a annulé ses mises à jour.

Participé
Le gestionnaire de ressources est un participant, mais n'a pas préparé, validé ou annulé ses mises à jour.

Lorsque le gestionnaire de files d'attente est redémarré, il demande à chaque base de données ayant une strophe XAResourceManager une liste de ses unités d'oeuvre globales en attente de validation. Si la base de données n'a pas été redémarrée, ou n'est pas disponible, le gestionnaire de files d'attente ne peut pas distribuer à la base de données les résultats finaux de ces unités d'oeuvre. Le résultat des unités d'oeuvre en attente de validation est envoyé à la base de données à la première opportunité lorsque la base de données est à nouveau disponible.

Dans ce cas, le gestionnaire de base de données est signalé comme étant à l'état *préparé* jusqu'à ce que la resynchronisation ait lieu.

Chaque fois que la commande `dspmqtrn` affiche une unité d'oeuvre en attente de validation, elle commence par répertorier tous les gestionnaires de ressource possibles susceptibles de participer. Un identificateur unique, *RMId*, est attribué à ces gestionnaires de ressources au lieu de leurs *noms* lorsque leur état est indiqué pour une unité d'oeuvre en attente de validation.

Exemple de sortie de dspmqtrn montre le résultat de l'exécution de la commande suivante :

```
dspmqtrn -m MY_QMGR
```

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

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

Où *Transaction number* est l'ID de la transaction qui peut être utilisée avec la commande rsvmqtrn. Pour plus d'informations sur le message AMQ7056 , voir AMQ7000-7999: WebSphere MQ product . Les variables *XID* font partie de la *spécification X/Open XA*; pour obtenir les informations les plus récentes sur cette spécification, voir: https://publications.opengroup.org/c193.

Figure 16. Exemple de sortie de dspmqtrn

La sortie illustrée dans Exemple de sortie de dspmqtrn indique que trois gestionnaires de ressources sont associés au gestionnaire de files d'attente. Le premier est le gestionnaire de ressources `0` qui correspond au gestionnaire de files d'attente proprement dit. Les deux autres instances de gestionnaire de ressources sont les bases de données `MQBankDB` et `MQFeeDB` Db2 .

L'exemple montre une seule unité d'oeuvre en attente de validation. Un message est émis pour les trois gestionnaires de ressources, ce qui signifie que des mises à jour ont été apportées au gestionnaire de files d'attente et aux deux bases de données Db2 au sein de l'unité de travail.

Les mises à jour apportées au gestionnaire de files d'attente et au gestionnaire de ressources **0** ont été *validées*. Les mises à jour des bases de données Db2 sont à l'état *préparé* , ce qui signifie que Db2 doit être devenu indisponible avant d'être appelé pour valider les mises à jour dans les bases de données *MQBankDB* et *MQFeeDB* .

L'unité d'oeuvre en attente de validation a un identificateur externe appelé ID d'échange ou XID (*ID transaction*). Il s'agit d'un élément de données donné à Db2 par le gestionnaire de files d'attente pour identifier sa partie de l'unité d'oeuvre globale.

Résolution des unités d'oeuvre en attente avec la commande rsvmqtrn
Les unités de travail en attente sont terminées lorsque le gestionnaire de files d'attente et DB2 sont resynchronisés.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

La sortie illustrée dans la Figure 16, à la page 66 montre une seule unité de travail en attente de validation dans laquelle la décision de validation n'a pas encore été distribuée aux deux bases de données DB2 .

Pour exécuter cette unité de travail, le gestionnaire de files d'attente et DB2 doivent être resynchronisés lorsque DB2 devient disponible. Le gestionnaire de files d'attente utilise le démarrage de nouvelles unités de travail comme une opportunité de reprendre contact avec DB2. Vous pouvez également demander au gestionnaire de files d'attente de se resynchroniser explicitement à l'aide de la commande **rsvmqtrn**.

Effectuez cette opération peu après le redémarrage de DB2 afin que les verrous de base de données associés à l'unité d'oeuvre en attente de validation soient libérés aussi rapidement que possible. Utilisez l'option -a qui demande au gestionnaire de files d'attente de résoudre toutes les unités d'oeuvre en attente de validation. Dans l'exemple suivant, DB2 a redémarré, de sorte que le gestionnaire de files d'attente peut résoudre l'unité d'oeuvre en attente de validation:

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

Résultats mixtes et erreurs
Bien que le gestionnaire de files d'attente utilise un protocole de validation en deux phases, cela n'élimine pas complètement la possibilité que certaines unités d'oeuvre s'achèvent avec des résultats mixtes. C'est le cas lorsque certains participants valident des mises à jour et d'autre les annulent.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Le fait que des unités d'oeuvre s'achèvent avec des résultats mixtes a des conséquences graves, car des ressources partagées qui auraient dû être mises à jour en tant qu'unité d'oeuvre unique ne sont plus à un état cohérent.

Des résultats mixtes surviennent principalement lorsque des décisions heuristiques sont effectuées sur des unités d'oeuvre au lieu de permettre au gestionnaire de files d'attente de résoudre lui-même les unités d'oeuvre en attente de validation. De telles décisions sont en dehors du contrôle du gestionnaire de files d'attente.

Chaque fois que le gestionnaire de files d'attente détecte un résultat mixte, il génère des informations FFST et documente l'échec dans ses journaux d'erreurs, avec l'un des deux messages suivants:

- Si un gestionnaire de base de données annule une transaction au lieu de la valider :

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

- Si un gestionnaire de base de données valide une transaction au lieu de l'annuler :

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

D'autres messages identifient les bases de données qui ont été endommagées en raison d'un traitement heuristique. Il vous incombe alors de restaurer localement la cohérence des bases de données concernées. Il s'agit d'une procédure complexe dans laquelle vous devez d'abord isoler la mise à jour de base de données qui a été validée ou annulée par erreur, puis annuler ou rétablir manuellement cette modification.

Modification des informations de configuration
Une fois que le gestionnaire de files d'attente a été correctement démarré pour coordonner des unités d'oeuvre globales, ne modifiez pas les informations de configuration du gestionnaire de ressources.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Si vous devez modifier les informations de configuration, vous pouvez le faire à tout moment, mais les modifications ne prendront effet qu'après le redémarrage du gestionnaire de files d'attente.

Si vous supprimez les informations de configuration du gestionnaire de ressources pour une base de données, vous empêchez le gestionnaire de files d'attente de contacter ce gestionnaire de base de données.

Il ne faut ***jamais*** modifier l'attribut de nom (*Name*) dans des informations de configuration du gestionnaire de ressources. Cet attribut identifie de manière unique cette instance de gestionnaire de base de données auprès du gestionnaire de files d'attente. Si vous modifiez cet identificateur unique, le gestionnaire de files d'attente suppose que la base de données a été supprimée et qu'une instance entièrement nouvelle a été ajoutée. Le gestionnaire de files d'attente continue d'associer les unités d'oeuvre en attente à l'ancien *nom*, ce qui risque de laisser la base de données à l'état en attente de validation.

Suppression d'instances de gestionnaire de base de données
Si vous devez supprimer définitivement une base de données de votre configuration, vérifiez que celle-ci n'est pas en attente de validation avant de redémarrer le gestionnaire de files d'attente.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les produits de base de données fournissent des commandes pour afficher les transactions en attente de validation. Si des transactions sont en attente de validation, permettez d'abord au gestionnaire de files d'attente de se resynchroniser avec la base de données. Pour ce faire, démarrez le gestionnaire de files d'attente. Vous pouvez vérifier que la resynchronisation a eu lieu en utilisant la commande **rsvmqtrn** ou la commande propre à la base de données pour afficher les unités d'oeuvre en attente de validation. Une fois que vous vous êtes assuré que la resynchronisation a eu lieu, arrêtez le gestionnaire de files d'attente et supprimez les informations de configuration de la base de données.

Si vous ne respectez pas cette procédure, le gestionnaire de files d'attente se souvient de toutes les unités d'oeuvre en attente de validation impliquant cette base de données. Un message d'avertissement AMQ7623 est émis chaque fois le gestionnaire de files d'attente est redémarré. Si vous n'allez jamais reconfigurer cette base de données avec le gestionnaire de files d'attente, utilisez l'option -r de la commande **rsvmqtrn** pour demander au gestionnaire de files d'attente d'oublier la participation de la base de données aux transactions en attente de validation. Le gestionnaire de files d'attente ne peut oublier ces transactions que lorsque les transactions en attente de validation ont été terminées avec tous les participants.

Vous devrez parfois supprimer temporairement certaines informations de configuration du gestionnaire de ressources. Sur les systèmes UNIX and Linux, la meilleur façon est de mettre en commentaire la strophe correspondante pour pouvoir la rétablir facilement par la suite. Vous pouvez décider de le faire si des erreurs se produisent chaque fois que le gestionnaire de files d'attente contacte une base de données ou un gestionnaire de base de données particulier. Le fait de supprimer temporairement les informations de configuration du gestionnaire de ressources concerné permet au gestionnaire de files d'attente de

démarrer des unités d'oeuvre globales impliquant tous les autres participants. Voici un exemple de strophe XAResourceManager mise en commentaire :

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

Figure 17. Strophe XAResourceManager mise en commentaire sous UNIX and Linux

Sur les systèmes Windows , utilisez WebSphere MQ Explorer pour supprimer les informations relatives à l'instance du gestionnaire de base de données. Veillez tout particulièrement à entrer le nom correct dans la zone *Name* lorsque vous rétablissez cette instance. Si vous orthographiez mal le nom, vous pouvez être confronté à des problèmes de transaction en attente de validation, comme décrit dans «Modification des informations de configuration», à la page 68.

Enregistrement dynamique XA

La spécification XA fournit un moyen de réduire le nombre d'appels xa_* effectués par un gestionnaire de transactions vers un gestionnaire de ressources. Cette optimisation est également appelée *enregistrement dynamique*.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

L'enregistrement dynamique est pris en charge par DB2. D'autres bases de données peuvent prendre en charge cette fonction (consultez la documentation de votre produit de base de données pour plus de détails).

Pourquoi l'optimisation par enregistrement dynamique est-elle utile ? Dans votre application, certaines unités d'oeuvre globales peuvent contenir des mises à jour apportées à des tables de base de données et d'autres peuvent ne pas les contenir. Lorsqu'aucune mise à jour persistante n'a été apportée à une table de base de données, il n'est pas nécessaire d'inclure cette base de données dans le protocole de validation exécuté lors du traitement de MQCMIT.

Que votre base de données prenne ou non en charge l'enregistrement dynamique, votre application appelle xa_open lors du premier appel MQBEGIN sur une connexion WebSphere MQ . Elle appelle également xa_close sur l'appel MQDISC suivant. Le modèle des appels XA suivants dépend de si la base de données prend en charge l'enregistrement dynamique :

Si votre base de données ne prend pas en charge l'enregistrement dynamique...
Chaque unité de travail globale implique plusieurs appels de fonction XA effectués par le code WebSphere MQ dans la bibliothèque du client de base de données, que vous ayez ou non effectué une mise à jour permanente des tables de cette base de données dans votre unité de travail. Ces gestionnaires sont les suivants :

- xa_start et xa_end depuis le processus d'application. Ces appels sont utilisés pour déclarer le début et la fin d'une unité d'oeuvre globale.
- xa_prepare, xa_commit et xa_rollback depuis le processus d'agent du gestionnaire de files d'attente, amqzlaa0. Ces appels sont utilisés pour distribuer le résultat de l'unité d'oeuvre globale (la décision de validation ou d'annulation).

En outre, le processus d'agent du gestionnaire de files d'attente appelle également xa_open lors du premier appel MQBEGIN.

Si votre base de données prend en charge l'enregistrement dynamique...
Le code WebSphere MQ effectue uniquement les appels de fonction XA nécessaires. Pour une unité d'oeuvre globale qui **n'a pas** impliqué de mises à jour persistantes sur des ressources de base de

données, **aucun** appel XA à la base de données n'est exécuté. Pour une unité d'oeuvre globale qui **a** impliqué de telles mises à jour persistantes, les appels sont effectués vers :

- `xa_end` depuis le processus d'application pour déclarer la fin de l'unité d'oeuvre globale.
- `xa_prepare`, `xa_commit` et `xa_rollback` depuis le processus d'agent du gestionnaire de files d'attente, amqzlaa0. Ces appels sont utilisés pour distribuer le résultat de l'unité d'oeuvre globale (la décision de validation ou d'annulation).

Pour que l'enregistrement dynamique fonctionne, il est essentiel que la base de données puisse indiquer à WebSphere MQ qu'elle a effectué une mise à jour permanente qu'elle souhaite inclure dans l'unité d'oeuvre globale en cours. WebSphere MQ fournit la fonction `ax_reg` à cette fin.

Le code client de la base de données qui s'exécute dans votre processus d'application trouve la fonction `ax_reg` et l'appelle pour *enregistrer dynamiquement* le fait qu'un travail persistant a été effectué dans l'unité d'oeuvre globale en cours. En réponse à cet appel `ax_reg` , WebSphere MQ enregistre la participation de la base de données. S'il s'agit du premier appel `ax_reg` sur cette connexion WebSphere MQ , le processus d'agent de gestionnaire de files d'attente appelle `xa_open`.

Le code client de la base de données effectue cet appel `ax_reg` lorsqu'il est exécuté dans votre processus, par exemple, lors d'un appel SQL UPDATE ou de tout appel émanant de l'API client de la base de données.

Cas d'erreur
Dans un enregistrement dynamique XA, il existe un risque de problème de confusion dans le gestionnaire de files d'attente.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Pour citer un exemple courant, si vous oublié de définir correctement les variables d'environnement de base de données avant de démarrer votre gestionnaire de files d'attente, les appels `xa_open` du gestionnaire de files d'attente échouent. Aucune unité d'oeuvre globale ne peut être utilisée.

Pour éviter ceci, vérifiez que vous avez défini les variables d'environnement appropriées avant de démarrer le gestionnaire de files d'attente. Consultez la documentation de votre produit de base de données et les conseils fournis dans «Configuration de Db2», à la page 53, «Configuration d'Oracle», à la page 55 et «Configuration de Sybase», à la page 60.

Avec tous les produits de base de données, le gestionnaire de files d'attente appelle `xa_open` une fois lors de son démarrage, dans le cadre d'une session de reprise (comme expliqué dans «Considérations à prendre en compte en cas de perte du contact avec le gestionnaire de ressources XA», à la page 64). Cet appel `xa_open` échoue si vous définissez vos variables d'environnement de base de données de manière incorrecte, mais cela n'entraîne pas l'échec du démarrage du gestionnaire de files d'attente. En effet, le même code d'erreur `xa_open` est utilisé par le client de base de données pour indiquer que le serveur de base de données n'est pas disponible. WebSphere MQ ne traite pas cette erreur comme une erreur grave, car le gestionnaire de files d'attente doit pouvoir commencer à traiter les données en dehors des unités d'oeuvre globales impliquant cette base de données.

Les appels suivants à `xa_open` sont effectués à partir du gestionnaire de files d'attente lors de la première commande MQBEGIN sur une connexion WebSphere MQ (si l'enregistrement dynamique n'est pas utilisé) ou lors d'un appel par le code du client de base de données à la fonction WebSphere MQ-fourni `ax_reg` (si l'enregistrement dynamique est utilisé).

La **temporisation** des conditions d'erreur (ou, parfois, des rapports FFST) varie selon que vous utilisez l'enregistrement dynamique:

- Si vous utilisez l'enregistrement dynamique, votre appel MQBEGIN peut aboutir, mais votre appel de base de données SQL UPDATE (ou un appel similaire) échouera.
- Si vous n'utilisez pas l'enregistrement dynamique, votre appel MQBEGIN échouera.

Vérifiez que vos variables d'environnement sont définies correctement dans vos processus d'application et de gestionnaire de files d'attente.

Récapitulatif des appels XA
Voici une liste des appels qui sont effectués vers les fonctions XA dans une bibliothèque de client de base de données comme un résultat des divers appels MQI qui contrôlent les unités d'oeuvre globales. Elle ne fournit pas une description complète du protocole décrit dans la spécification XA, mais juste une brève présentation.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Notez que les appels `xa_start` et `xa_end` sont toujours appelés par le code WebSphere MQ dans le processus d'application, tandis que les appels `xa_prepare`, `xa_commit`et `xa_rollback` sont toujours appelés à partir du processus d'agent de gestionnaire de files d'attente, amqzlaa0.

Les appels `xa_open` et `xa_close` indiqués dans ce tableau sont tous exécutés depuis le processus d'application. Le processus d'agent de gestionnaire de files d'attente appelle `xa_open` dans les situations décrites dans «Cas d'erreur», à la page 70.

Tableau 7. Récapitulatif des appels de fonction XA

Appel MQI	**Appels XA effectués avec l'enregistrement dynamique**	**Appels XA effectués sans l'enregistrement dynamique**
Premier appel MQBEGIN	`xa_open`	`xa_open` `xa_start`
Appels MQBEGIN suivants	Pas d'appel XA	`xa_start`

MQCMIT (**sans** `ax_reg` appelé pendant l'unité d'oeuvre globale en cours)	Pas d'appel XA	`xa_end`
		`xa_prepare`
		`xa_commit`
		`xa_rollback`

MQCMIT (**avec** `ax_reg` appelé lors de l'unité d'oeuvre globale en cours)	`xa_end`
	`xa_prepare`
	`xa_commit`
	`xa_rollback`
		Non applicable. Aucun appel n'est effectué à ax_reg en mode non dynamique.

MQBACK (**sans** `ax_reg` appelé pendant l'unité de travail globale en cours)	Pas d'appel XA	`xa_end` `xa_rollback`
MQBACK (**avec** `ax_reg` appelé pendant l'unité de travail globale en cours)	`xa_end` `xa_rollback`	Non applicable. Aucun appel n'est effectué à ax_reg en mode non dynamique.
MQDISC, dans le cas où MQCMIT ou MQBACK a été appelé en premier. Si ce n'est pas la cas, le traitement de MQCMIT est le premier effectué pendant MQDISC.	`xa_close`	`xa_close`

Tableau 7. Récapitulatif des appels de fonction XA (suite)		
Appel MQI	**Appels XA effectués avec l'enregistrement dynamique**	**Appels XA effectués sans l'enregistrement dynamique**
Remarques : 1. Pour MQCMIT, `xa_commit` est appelé si `xa_prepare` aboutit. Sinon, `xa_rollback` est appelé.		

Scénario 2 : Un autre logiciel assure la coordination

Dans le scénario 2, un gestionnaire de transactions externe coordonne les unités d'oeuvre globales, et les démarre et les valide sous le contrôle de l'API du gestionnaire de transactions. Les instructions MQBEGIN, MQCMIT et MQBACK ne sont pas disponibles.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

La présente section décrit ce scénario, notamment :

- «Coordination de point de synchronisation externe», à la page 72
- «Utilisation de CICS», à la page 75
- «Utilisation de Microsoft Transaction Server (COM +)», à la page 80

Le client IBM WebSphere MQ for HP Integrity NonStop Server peut utiliser la fonction de gestion des transactions (TMF) HP NonStop pour coordonner des unités d'oeuvre globales. Pour plus d'informations, voir Utilisation de HP NonStop TMF.

Coordination de point de synchronisation externe

Une unité d'oeuvre globale peut également être coordonnés par un gestionnaire de transactions compatible X/Open XA externe. Ici, le gestionnaire de files d'attente WebSphere MQ participe à l'unité de travail, mais ne la coordonne pas.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Le flux de contrôle dans une unité d'oeuvre globale coordonnée par un gestionnaire de transactions externe se présente comme suit :

1. Une application indique au coordinateur de point de synchronisation externe (par exemple, TXSeries) qu'elle souhaite démarrer une transaction.
2. Le coordinateur de point de synchronisation informe les gestionnaires de ressources connus, tels que WebSphere MQ, sur la transaction en cours.
3. L'application émet des appels vers les gestionnaires de ressources associés à la transaction en cours. Par exemple, l'application peut émettre des appels MQGET à WebSphere MQ.
4. L'application envoie une demande de validation ou d'annulation au coordinateur de point de synchronisation externe.
5. Le coordinateur de point de synchronisation termine la transaction en émettant les appels appropriés pour chacun des gestionnaires de ressources, généralement, à l'aide de protocoles de validation en deux phases.

Les niveaux pris en charge des coordinateurs de point de synchronisation externe qui peuvent fournir un processus de validation en deux phases pour les transactions auxquelles WebSphere MQ participe sont définis à l'adresse IBM WebSphere MQ -Configuration système requise détaillée.

Le reste de cette section explique comment activer des unités d'oeuvre externes.

Structure de commutation IBM WebSphere MQ XA
Chaque gestionnaire de ressources qui participe à une unité d'oeuvre coordonnée en externe doit fournir une structure de commutateur XA. Cette structure définit les capacités du gestionnaire de ressources et les fonctions qui doivent être appelées par le coordinateur de point de synchronisation.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

IBM WebSphere MQ fournit deux versions de cette structure :

- *MQRMIXASwitch* pour la gestion des ressources XA statiques
- *MQRMIXASwitchDynamic* pour la gestion des ressources XA dynamiques

Consultez la documentation de votre gestionnaire de transactions pour déterminer si vous devez utiliser l'interface de gestion de ressources statiques ou dynamiques. Lorsqu'un gestionnaire de transactions prend en charge cette fonction, nous vous recommandons d'utiliser la gestion des ressources XA dynamiques.

Certains gestionnaires de transaction 64 bits traitent le type *long* dans les spécifications XA en mode 64 bits et d'autres, en mode 32 bits. WebSphere MQ prend en charge les deux modèles:

- Si votre gestionnaire de transactions est en mode 32 bits, ou qu'il est en mode 64 bits mais qu'il traite le type *long* en mode 32 bits, utilisez le fichier de commutation de chargement répertorié dans le Tableau 8, à la page 73.
- Si votre gestionnaire de transactions est en mode 64 bits et traite le type *long* en mode 64 bits, utilisez le fichier de commutation de chargement répertorié dans le Tableau 9, à la page 74.

Une liste des gestionnaires de transactions 64 bits connus qui traitent le type *long* en mode 64 bits est fournie dans le Tableau 10, à la page 74. Consultez la documentation de votre gestionnaire de transactions si vous n'êtes pas sûr du modèle qu'il utilise.

Tableau 8. Noms de fichier de chargement de commutateur XA

Plateforme	**Nom du fichier de commutation de chargement (serveur)**	**Nom du fichier de commutation de chargement (client transactionnel étendu)**
Windows	*mqmxa.dll*	*mqcxa.dll*
AIX (sans unités d'exécution)	*libmqmxa.a*	*libmqcxa.a*
AIX (avec unités d'exécution)	*libmqmxa_r.a*	*libmqcxa_r.a*
HP-UX (sans unités d'exécution)	*libmqmxa.so*	*libmqcxa.so*
HP-UX (avec unités d'exécution)	*libmqmxa_r.so*	*libmqcxa_r.so*
Linux (sans unités d'exécution)	*libmqmxa.so*	*libmqcxa.so*
Linux (avec unités d'exécution)	*libmqmxa_r.so*	*libmqcxa_r.so*
Solaris	*libmqmxa.so*	*libmqcxa.so*

Tableau 9. Noms des autres fichiers de chargement de commutateur XA 64 bits		
Plateforme	**Nom du fichier de commutation de chargement (serveur)**	**Nom du fichier de commutation de chargement (client transactionnel étendu)**
AIX (sans unités d'exécution)	*libmqmxa64.a*	*libmqcxa64.a*
AIX (avec unités d'exécution)	*libmqmxa64_r.a*	*libmqcxa64_r.a*
HP-UX (sans unités d'exécution)	*libmqmxa64.so*	*libmqcxa64.so*
HP-UX (avec unités d'exécution)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Linux (sans unités d'exécution)	*libmqmxa64.so*	*libmqcxa64.so*
Linux (avec unités d'exécution)	*libmqmxa64_r.so*	*libmqcxa64_r.so*
Solaris	*libmqmxa64.so*	*libmqcxa64.so*

Tableau 10. Gestionnaire de transactions 64 bits nécessitant le fichier de commutation de chargement 64 bits complémentaire
Gestionnaire de transactions
Tuxedo

Certains coordinateurs de point de synchronisation externes (non CICS) requièrent que chaque gestionnaire de ressources participant à une unité de travail fournisse son nom dans la zone de nom de la structure de commutateur XA. Le nom du gestionnaire de ressources WebSphere MQ est MQSeries_XA_RMI.

Le coordinateur de point de synchronisation définit comment la structure du commutateur WebSphere MQ XA est liée à ce dernier. Des informations sur la liaison de la structure de commutateur WebSphere MQ XA à CICS sont fournies dans «Utilisation de CICS», à la page 75. Pour plus d'informations sur la liaison de la structure de commutateur XA WebSphere MQ avec d'autres coordinateurs de point de synchronisation compatibles XA, consultez la documentation fournie avec ces produits.

Les remarques suivantes s'appliquent à l'utilisation de WebSphere MQ avec tous les coordinateurs de point de synchronisation compatibles XA:

- La structure xa_info transmise à tout appel xa_open par le coordinateur de point de synchronisation inclut le nom d'un gestionnaire de files d'attente WebSphere MQ . Ce nom est le même que le nom de gestionnaire de files d'attente transmis à l'appel `MQCONN`. Si le nom transmis à l'appel xa_open est vide, le gestionnaire de files d'attente par défaut est utilisé.

  Sinon, la structure xa_info peut contenir des valeurs pour les paramètres *TPM* et *AXLIB*. Le paramètre *TPM* indique le gestionnaire de transactions utilisé. Les valeurs admises sont CICS, TUXEDO et ENCINA. Le paramètre *AXLIB* indique le nom de la bibliothèque qui contient les fonctions ax_reg et ax_unreg du gestionnaire de transactions. Pour plus d'informations sur ces paramètres, voir Configuration d'un client transactionnel étendu. Si la structure xa_info contient l'un de ces paramètres, le gestionnaire de files d'attente est spécifié dans le paramètre *QMNAME*, sauf si le gestionnaire de files d'attente par défaut est utilisé.
- Un seul gestionnaire de files d'attente à la fois peut participer à une transaction coordonnée par une instance d'un coordinateur de point de synchronisation externe. Le coordinateur de point de synchronisation est effectivement connecté au gestionnaire de files d'attente et est soumis à la règle stipulant qu'une seule connexion à la fois est prise en charge.

- Toutes les applications qui incluent des appels à un coordinateur de point de synchronisation externe ne peuvent se connecter qu'au gestionnaire de files d'attente qui participe à la transaction gérée par ce coordinateur (car elles sont effectivement déjà connectées à ce gestionnaire). Toutefois, de telles applications doivent exécuter un appel `MQCONN` pour obtenir un descripteur de connexion et un appel `MQDISC` avant de quitter.
- Un gestionnaire de files d'attente avec des mises à jour de ressource coordonnées par un coordinateur de point de synchronisation externe doit être démarré avant ce coordinateur. De même, le coordinateur de point de synchronisation doit s'arrêter avant le gestionnaire de files d'attente.
- Si votre coordinateur de point de synchronisation externe s'arrête de façon anormale, arrêtez et redémarrez votre gestionnaire de files d'attente ***avant*** de redémarrer le coordinateur pour que les opérations de messagerie non validées au moment de la défaillance soient correctement résolues.

Utilisation de CICS

CICS est l'un des éléments de TXSeries.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les versions de TXSeries compatibles XA (et qui utilisent un processus de validation en deux phases) sont définies à l'adresse suivante: IBM WebSphere MQ -Configuration système requise détaillée

WebSphere MQ prend également en charge d'autres gestionnaires de transactions. Vous trouverez la liste actualisée des logiciels pris en charge dans IBM WebSphere MQ -Configuration système requise détaillée.

Exigences du processus de validation en deux phases

Exigences du processus de validation en deux phases lorsque vous utilisez le processus de validation en deux phases CICS avec WebSphere MQ. Ces exigences ne s'appliquent pas à z/OS.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Prenez connaissance des exigences suivantes :

- WebSphere MQ et CICS doivent résider sur la même machine physique.
- WebSphere MQ ne prend pas en charge CICS sur un client WebSphere MQ MQI.
- Vous devez démarrer le gestionnaire de files d'attente avec son nom spécifié dans la section de définition de ressource XAD, ***avant*** de tenter de démarrer CICS. Si vous n'y parvenez pas, vous ne pourrez pas démarrer CICS si vous avez ajouté une strophe de définition de ressource XAD pour WebSphere MQ à la région CICS .
- Un seul gestionnaire de files d'attente WebSphere MQ à la fois est accessible à partir d'une seule région CICS .
- Une transaction CICS doit émettre une demande `MQCONN` pour pouvoir accéder aux ressources WebSphere MQ . L'appel `MQCONN` doit indiquer le nom du gestionnaire de files d'attente WebSphere MQ spécifié dans l'entrée XAOpen de la section de définition de ressource XAD pour la région CICS . Si cette entrée est vide, la demande `MQCONN` doit spécifier le gestionnaire de files d'attente par défaut.
- Une transaction CICS qui accède aux ressources WebSphere MQ doit émettre un appel `MQDISC` à partir de la transaction avant de revenir à CICS. Si vous ne le faites pas, cela peut signifier que le serveur d'applications CICS est toujours connecté, laissant les files d'attente ouvertes. De plus, si vous n'installez pas d'exit de fin de tâche (voir «Exemple d'exit de fin de tâche», à la page 79), le serveur d'applications CICS risque de s'arrêter de manière anormale ultérieurement, peut-être lors d'une transaction ultérieure.
- Vous devez vous assurer que l'ID utilisateur CICS (cics) est membre du groupe mqm, de sorte que le code CICS ait le droit d'appeler WebSphere MQ.

Pour les transactions s'exécutant dans un environnement CICS , le gestionnaire de files d'attente adapte ses méthodes d'autorisation et de détermination du contexte comme suit:

- Le gestionnaire de files d'attente interroge l'ID utilisateur sous lequel CICS exécute la transaction. Il s'agit de l'ID utilisateur vérifié par le gestionnaire des droits d'accès aux objets et qui est utilisé pour les informations contextuelles.
- Dans le contexte de message, le type d'application est MQAT_CICS.
- Le nom de l'application dans le contexte est copié à partir du nom de transaction CICS .

Prise en charge de General XA
Le général XA n'est pas pris en charge sous IBM i. Un module de chargement de commutateur XA est fourni pour vous permettre de lier CICS à WebSphere MQ sur les systèmes UNIX and Linux . En outre, des exemples de fichier de code source sont fournis pour vous permettre de développer les commutateurs XA pour d'autres messages de transaction.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les noms des modules de chargement de commutateur fournis sont les suivants :

Tableau 11. Code essentiel pour les applications CICS : routine d'initialisation XA

C (source)	**C (exec) - ajoutez l'un des éléments suivants à votre section XAD.Stanza**
amqzscix.c	amqzsc - TXSeries pour AIXversion 5.1, amqzsc - TXSeries for HP-UX, version 5.1 amqzsc - TXSeries for Sun Solaris, version 5.1
amqzscin.c	mqmc4swi - TXSeries pour Windowsversion 5.1

Génération de bibliothèques à utiliser avec TXSeries for Multiplatforms
Utilisez ces informations lors de la génération de bibliothèques à utiliser avec TXSeries for Multiplatforms.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Les *fichiers de commutation de chargement préconfigurés* sont des bibliothèques partagées (appelées *DLL* sur le système Windows) que vous pouvez utiliser avec les programmes CICS, qui nécessitent une transaction de validation en deux phases à l'aide du protocole XA. Les noms de ces bibliothèques préconfigurées se trouvent dans la table Code essentiel pour les applications CICS : routine d'initialisation XA. Un exemple de code source est également fourni dans les répertoires suivants :

Tableau 12. Répertoires d'installation sous Windows, systèmes d'exploitation UNIX and Linux

Plateforme	**Répertoire**	**Fichier source**
UNIX and Linux	*MQ_INSTALLATION_PATH*/samp/	amqzscix.c

Tableau 12. Répertoires d'installation sous Windows, systèmes d'exploitation UNIX and Linux (suite)		
Plateforme	**Répertoire**	**Fichier source**
Windows	*MQ_INSTALLATION_PATH*\Tools \c \ Exemples	amqzscin.c

où*MQ_INSTALLATION_PATH* est le répertoire dans lequel vous avez installé IBM WebSphere MQ.

Pour générer le fichier de commutation de chargement à partir de l'exemple de source, suivez les instructions appropriées pour votre système d'exploitation :

AIX

Entrez la commande suivante :

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

Solaris

Entrez la commande suivante :

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

HP-UX

Entrez la commande suivante :

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

Plateformes Linux

Entrez la commande suivante :

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

Windows

Procédez comme suit :

1. Utilisez la commande cl pour générer amqzscin.obj en compilant au moins les variables suivantes :

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

2. Créez un fichier de définition de module nommé `mqmc1415.def`, qui contient les lignes suivantes :

   ```
   LIBRARY MQMC4SWI
   EXPORTS
   CICS_XA_Init
   ```

3. Utilisez la commande **lib** afin de générer un fichier d'exportation et une bibliothèque d'importation en utilisant au moins l'option suivante :

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

   Si la commande lib aboutit, un fichier `mqmc4swi.exp` est également généré.

4. Utilisez la commande link pour générer `mqmc4swi.dll` en utilisant au moins l'option suivante :

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

Prise en charge d'IBM WebSphere MQXA et Tuxedo
IBM WebSphere MQ sous Windows, les systèmesUNIX and Linux peuvent bloquer indéfiniment les applications XA coordonnées par Tuxedo dans xa_start.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Cela ne peut se produire que lorsque plusieurs processus coordonnés par Tuxedo dans une transaction globale unique tentent d'accéder à IBM WebSphere MQ à l'aide du même ID de branche de transaction (XID). Cela ne peut pas avoir lieu si Tuxedo attribue à chaque processus de la transaction globale un ID transaction différent à utiliser avec IBM WebSphere MQ.

Pour éviter ce problème, configurez chaque application de Tuxedo qui accède à IBM WebSphere MQ sous un identificateur de transaction globale (gtrid) unique dans son groupe de serveurs Tuxedo. Les processus d'un même groupe de serveurs utilisent le même ID transaction lorsqu'ils accèdent à des gestionnaires de ressources pour le compte d'un identificateur de transaction globale unique et sont donc susceptibles de se bloquer dans xa_start dans IBM WebSphere MQ. Les processus de groupes de serveurs différents utilisent des ID transaction distincts lorsqu'ils accèdent à des gestionnaires de ressources et n'ont donc pas besoin de sérialiser leur travail de transaction dans IBM WebSphere MQ.

Activation du processus de validation en deux phases CICS
Pour permettre à CICS d'utiliser un processus de validation en deux phases pour coordonner les transactions qui incluent des appels MQI, ajoutez une entrée de strophe de définition de ressource XAD CICS à la région CICS . Notez que cette rubrique ne s'applique pas à z/OS.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Voici un exemple d'ajout d'une entrée de strophe XAD pour WebSphere MQ for Windows, où <Drive> est l'unité sur laquelle WebSphere MQ est installé (par exemple, D:).

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

Pour des clients transactionnels étendus, utilisez le fichier de commutation de chargement mqcc4swi.dll.

Voici un exemple d'ajout d'une entrée de strophe XAD pour les systèmes WebSphere MQ for UNIX and Linux , où *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé:

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

Pour des clients transactionnels étendus, utilisez le fichier de commutation de chargement amqczsc.

Pour plus d'informations sur l'utilisation de la commande **cicsadd** , voir *CICS Administration Reference*ou *CICS Administration Guide* pour votre plateforme.

Les appels à WebSphere MQ peuvent être inclus dans une transaction CICS et les ressources WebSphere MQ seront validées ou annulées comme indiqué par CICS. Cette prise en charge n'est pas disponible pour les applications client.

Vous ***devez*** émettre un `MQCONN` à partir de votre transaction CICS afin d'accéder aux ressources WebSphere MQ , suivies d'un `MQDISC` correspondant à la sortie.

Activation des exits utilisateur CICS
Un point d'exit utilisateur CICS (généralement appelé *exit utilisateur*) est un emplacement dans un module CICS où CICS peut transférer le contrôle à un programme que vous avez écrit (un programme *d'exit utilisateur*) et où CICS peut reprendre le contrôle lorsque votre programme d'exit a terminé son travail.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Avant d'utiliser un exit utilisateur CICS , lisez le manuel *CICS -Guide d'administration* correspondant à votre plateforme.

Exemple d'exit de fin de tâche
WebSphere MQ fournit un exemple de code source pour un exit d'arrêt de tâche CICS .

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

L'exemple de code source figure dans les répertoires suivants :

Tableau 13. Exits de fin de tâche CICS

Plateforme	**Répertoire**	**Fichier source**
Systèmes UNIX and Linux	*MQ_INSTALLATION_PATH*/samp	amqzscgx.c
Windows	*MQ_INSTALLATION_PATH*\Tools \c \ Exemples	amqzscgn.c

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Les instructions de génération de l'exemple d'exit de fin de tâche sont contenues dans les commentaires situés en haut de chaque fichier source.

Cet exit est appelé par CICS lors de l'arrêt normal et anormal de la tâche (après la prise d'un point de synchronisation). Aucun travail récupérable n'est autorisé dans le programme d'exit.

Ces fonctions sont uniquement utilisées dans un contexte WebSphere MQ et CICS dans lequel la version CICS prend en charge l'interface XA. CICS fait référence à ces bibliothèques en tant que "programmes" ou "exits utilisateur".

CICS comporte un certain nombre d'exits utilisateur et amqzscgx, s'il est utilisé, est défini et activé sur CICS en tant qu' "exit utilisateur de fin de tâche (UE014015)", c'est-à-dire l'exit numéro 15.

Lorsque l'exit de fin de tâche est appelé par CICS, CICS a déjà informé WebSphere MQ de l'état de fin de la tâche et WebSphere MQ a effectué l'action appropriée (validation ou annulation). Tout ce que fait l'exit est d'exécuter une commande MQDISC pour effectuer un nettoyage.

L'un des objectifs de l'installation et de la configuration de votre système CICS pour utiliser un exit d'arrêt de tâche est de protéger votre système contre certaines des conséquences d'un code d'application défectueux. Par exemple, si votre transaction CICS se termine de manière anormale sans avoir d'abord appelé MQDISC et qu'aucun exit d'arrêt de tâche n'est installé, vous pouvez voir (dans un délai d'environ 10 secondes) une défaillance irrémédiable ultérieure de la région CICS . En effet, l'unité d'exécution de santé de WebSphere MQ, qui s'exécute dans le processus cicsas, n'aura pas été publiée et n'aura pas eu le temps de nettoyer et de renvoyer. Les symptômes peuvent être que le processus cicsas se termine immédiatement, après avoir écrit des rapports FFST dans /var/mqm/errors ou l'emplacement équivalent sous Windows.

Utilisation de Microsoft Transaction Server (COM +)

COM + (Microsoft Transaction Server) est conçu pour aider les utilisateurs à exécuter des applications de logique métier dans un serveur intermédiaire standard.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Pour plus d'informations, voir Fonctions disponibles uniquement avec l'installation principale sous Windows .

COM + divise le travail en *activités*, qui sont généralement de courts blocs indépendants de la logique métier, tels que *Transférer des fonds d'un compte à un autre*. COM + est très orienté objet, et, en particulier, sur COM ; une activité COM+ est librement représentée par un objet COM (business).

COM+ fait partie intégrante du système d'exploitation. Pour utiliser COM + sous Windows 2000 et Windows XP, vous avez besoin du correctif logiciel Q313582 (également appelé package de cumul COM + 19.1).

COM+ fournit à l'administrateur d'objet métier les trois services suivants, afin d'éliminer une grande partie des soucis du programmeur d'objet métier :

- Gestion des transactions
- Sécurité
- Regroupement de ressources

Vous utilisez généralement COM + avec du code frontal qui est un client COM pour les objets détenus dans COM +, et des services de back end tels qu'une base de données, avec un pontage WebSphere MQ entre l'objet métier COM + et le back-end.

Le code frontal peut être un programme autonome ou un ASP (Active Server Page) hébergé par Microsoft Internet Information Server (IIS). Le code frontal peut se trouver sur le même ordinateur que COM + et ses objets métier, avec une connexion via COM. Alternativement, le code frontal peut être sur un autre ordinateur, avec une connexion via DCOM. Vous pouvez utiliser des clients différents pour accéder au même objet métier COM+ dans des situations différentes.

Le code dorsal peut se trouver sur le même ordinateur que COM + et ses objets métier, ou sur un autre ordinateur avec une connexion via l'un des protocoles pris en charge par WebSphere MQ .

Expiration des unités d'oeuvre globales

Le gestionnaire de files d'attente peut être configuré de sorte que les unités d'oeuvre globales arrivent à expiration à l'issue d'un délai d'inactivité prédéfini.

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

Pour activer ce comportement, définissez les variables d'environnement suivantes :

- *AMQ_TRANSACTION_EXPIRY_RESCAN*= < intervalle de réexamen en millisecondes >

- *AMQ_XA_TRANSACTION_EXPIRATION*= < intervalle de délai d'attente en millisecondes >

Avertissement : Les variables d'environnement affectent uniquement les transactions qui sont à l'état *Idle* dans le tableau 6-4 de la spécification XA. En d'autres termes, il s'agit des transactions qui ne sont pas associées dans une unité d'exécution d'application, mais pour lesquelles le logiciel de gestionnaire de transactions externe n'a pas encore exécuté l'appel de fonction **xa_prepare**.

Les gestionnaires de transactions externes conservent uniquement un journal des transactions qui sont préparées, validées ou annulées. Si le gestionnaire de transactions externe tombe en panne pour une raison quelconque, lorsqu'il est rétabli, il termine l'exécution des transactions préparées, validées et annulées, mais les transactions actives n'ayant pas encore été préparées sont orphelines. Pour éviter ce problème, définissez *AMQ_XA_TRANSACTION_EXPIRY* de manière à tenir compte de l'intervalle entre l'émission des appels API transactionnels MQI et la fin de la transaction, pour une application ayant effectué une tâche transactionnelle sur d'autres gestionnaires de ressources.

Pour garantir un nettoyage rapide à l'issue de l'expiration de la variable *AMQ_XA_TRANSACTION_EXPIRY*, définissez la valeur de *AMQ_TRANSACTION_EXPIRY_RESCAN* sur une valeur inférieure à celle de l'intervalle *AMQ_XA_TRANSACTION_EXPIRY*, idéalement de sorte que la nouvelle analyse survienne plus d'une fois dans l'intervalle *AMQ_XA_TRANSACTION_EXPIRY*.

Disposition unité de récupération

WebSphere MQ for z/OS fournit des dispositions d'unité de récupération. Cette fonction permet de configurer si la deuxième phase des transactions de validation à 2 phases peut être gérée, par exemple, lors de la reprise, lorsque vous êtes connecté à un autre gestionnaire de files d'attente du même groupe de partage de files d'attente (QSG).

Remarque : Cette rubrique est également disponible dans IBM MQ Version 8.0 et les versions ultérieures. Toutefois, vous ne pouvez pas passer à une version ultérieure à l'aide de la zone de liste "Modifier la version" . Pour accéder à la rubrique dans une version ultérieure, éditez le numéro de version dans la zone URL de votre navigateur.

WebSphere MQ for z/OS V7.0.1 et versions ultérieures prend en charge la disposition de l'unité de récupération.

Disposition unité de récupération

La disposition d'unité de récupération est liée à la connexion de l'application ainsi qu'à toutes les transactions que celle-ci démarre. Il existe deux dispositions d'unité de récupération possibles.

- Une disposition d'unité de récupération GROUP identifie qu'une application transactionnelle se connecte logiquement au groupe de partage de files d'attente et n'a aucune affinité avec un gestionnaire de files d'attente particulier. Toute transaction de validation de phase 2 que celle-ci démarre et ayant terminé la phase 1 du processus de validation, c'est-à-dire en attente de validation, peut être analysée et résolue lorsqu'elle est connectée à un gestionnaire de files d'attente au sein du groupe de partage de files d'attente. Dans un scénario de reprise, cela signifie que le coordinateur de transactions n'a pas à se reconnecter au même gestionnaire de files d'attente, qui peut être indisponible.
- Une disposition d'unité de récupération QMGR (gestionnaire de files d'attente) identifie qu'une application a une affinité directe avec le gestionnaire de files d'attente auquel elle est connectée, et toutes les transactions qu'elle démarre ont aussi cette disposition.

  Dans un scénario de reprise, le coordinateur de transaction doit se reconnecter au même gestionnaire de files d'attente pour interroger et résoudre les transactions en attente de validation, que le gestionnaire appartienne à un groupe de partage de files d'attente ou non.

Choix du langage de programmation à utiliser

Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

IBM WebSphere MQ prend en charge les langages de procédure de programmation suivants:

- C
- Visual Basic (systèmes Windows uniquement)
- COBOL

Ces langues 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 la prise en charge de ces langues, voir «Utilisation des langages de procédure avec WebSphere MQ», à la page 82.

IBM WebSphere MQ fournit une prise en charge pour:

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

Ces langages utilisent le modèle d'objet IBM WebSphere MQ , qui fournit des classes fournissant les mêmes fonctionnalités que les appels et les structures WebSphere MQ , mais qui constituent une méthode plus naturelle de programmation dans un environnement orienté objet. Certains des langages qui utilisent le modèle d'objet IBM WebSphere MQ fournissent des fonctions supplémentaires qui ne sont pas disponibles dans l'interface de file d'attente de messages (MQI). Pour plus d'informations sur la prise en charge de ces langues, voir «Programmation orientée objet avec WebSphere MQ», à la page 82.

Utilisation des langages de procédure avec WebSphere MQ

Pour plus d'informations sur l'écriture de vos applications dans la langue de votre choix, voir les liens suivants:

- «Codage en C», à la page 86
- «Codage dans Visual Basic», à la page 90
- «Codage en COBOL», à la page 89

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.

WebSphere 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 WebSphere MQ», à la page 84.

Si vous pouvez choisir la langue dans laquelle coder vos programmes, tenez compte de la longueur maximale des messages traités par vos programmes. Si vos programmes ne traitent que des messages dont la longueur maximale est connue, vous pouvez les coder dans n'importe quel langage de programmation pris en charge. Mais si vous ne connaissez pas la longueur maximale des messages que les programmes doivent traiter, le langage 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 quel langage pris en charge par CICS. L'interface EXEC CICS fournit les appels pour la gestion de la mémoire, si nécessaire.

Programmation orientée objet avec WebSphere MQ

Certains des langages et des infrastructures de programmation qui utilisent le modèle d'objet IBM WebSphere MQ fournissent des fonctions supplémentaires qui ne sont pas disponibles dans l'interface

de file d'attente de messages (MQI). Pour plus de détails sur les classes, les méthodes et les propriétés fournies par le modèle d'objet IBM WebSphere MQ , voir «Modèle d'objet IBM WebSphere MQ», à la page 91.

.NET

Pour plus d'informations sur le codage des programmes .NET à l'aide des classes WebSphere MQ .NET, voir Utilisation de .NET . Les Message Service Clients for C/C++ and .NET fournissent une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que le service JMS (Java Message Service) API.

C++

IBM WebSphere MQ fournit des classes C++ équivalentes aux objets WebSphere 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 de programmes à l'aide du modèle d'objet WebSphere MQ en C + +. Message Service Clients for C/C++ and .NET fournit une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que le service JMS (Java Message Service) API.

Java

Pour plus d'informations sur le codage des programmes à l'aide du modèle d'objet WebSphere MQ en Java, voir Utilisation de Java . Pour plus d'informations sur les différences entre les classes IBM WebSphere MQ pour Java et les classes IBM WebSphere MQ afin de vous aider à décider lesquelles utiliser, voir «Dois-je utiliser des classes IBM WebSphere MQ pour Java ou des classes IBM WebSphere MQ pour JMS?», à la page 93.

JMS

Websphere MQ fournit également des classes qui implémentent la spécification JMS (Java Message Service). Pour plus de détails sur les classes Websphere MQ pour JMS, voir Utilisation de Java. Pour plus d'informations sur les différences entre les classes IBM WebSphere MQ pour Java et les classes IBM WebSphere MQ afin de vous aider à décider lesquelles utiliser, voir «Dois-je utiliser des classes IBM WebSphere MQ pour Java ou des classes IBM WebSphere MQ pour JMS?», à la page 93.

Les Message Service Clients for C/C++ and .NET fournissent une interface de programme d'application (API) appelée XMS qui possède le même ensemble d'interfaces que le service JMS (Java Message Service) API.

ActiveX

WebSphere MQ ActiveX est communément appelé MQAX. Le MQAX est inclus dans WebSphere MQ for Windows. La prise en charge d' ActiveX a été stabilisée au niveau WebSphere MQ Version 6.0 . Pour exploiter les fonctions introduites dans WebSphere MQ à partir de la version 6.0, envisagez d'utiliser .NET à la place. Pour plus d'informations sur le codage des programmes à l'aide du modèle d'objet WebSphere MQ dans ActiveX, voir Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX) .

Concepts associés

Présentation technique

«Développement d'applications», à la page 7
IBM WebSphere MQ fournit plusieurs façons de développer des applications pour envoyer et recevoir les 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.

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

Référence associée

Référence de développement d'application

Fichiers de définition de données IBM WebSphere MQ

IBM WebSphere 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 permettant d'écrire des exits de canal sont décrits dans WebSphere 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 installables WebSphere MQ», à la page 390.

Pour les fichiers de définition de données pris en charge en C + +, voir Utilisation de C++.

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

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 WebSphere MQ , sous les rubriques suivantes:

- «Fichiers d'inclusion du langage C», à la page 84
- «Fichiers du module Visual Basic», à la page 85
- «Fichiers de copie COBOL», à la page 85

Fichiers d'inclusion du langage C

Les fichiers d'inclusion WebSphere MQ C sont répertoriés dans les fichiers d'en-tête C. Ils sont installés dans les répertoires ou les bibliothèques suivants:

Plateforme	Répertoire d'installation ou bibliothèque
Plateformes UNIX	*MQ_INSTALLATION_PATH*/inc/
Systèmes Windows	*MQ_INSTALLATION_PATH*\Tools\c\include

où *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Remarque : Pour les plateformes UNIX , 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

WebSphere 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, WebSphere 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).

Les fichiers de copie COBOL WebSphere MQ sont répertoriés dans Fichiers COBOL COPY. Ils sont installés dans les répertoires suivants:

Plateforme	Répertoire d'installation ou bibliothèque
Autres plateformes UNIX	*MQ_INSTALLATION_PATH*/inc/
Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (pour Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (pour IBM VisualAge COBOL)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere 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.

Codage en C

Notez les informations des sections suivantes lors du codage des programmes WebSphere 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. Par exemple, le symbole ¬ représente un caractère blanc:

  ```
  #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
```

Codage en COBOL

Notez les informations de la section suivante lors du codage des programmes WebSphere 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 WebSphere 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 WebSphere MQ ensemble de sorte qu'elles soient toutes alignées sur les limites.

Codage dans pTAL

Notez les informations de la section suivante lors du codage des programmes IBM WebSphere MQ dans pTAL.

HP Integrity NonStop Server

Définition et initialisation de structures IBM WebSphere MQ

Les définitions de structure pTAL pour les structures IBM WebSphere MQ sont fournies avec des noms qui se terminent par ^DEF. Par exemple, les déclarations pTAL suivantes seraient codées pour créer une structure IBM WebSphere MQ Message Descriptor (MQMD) et une structure IBM WebSphere MQ Put Message Options (MQPMO).

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

IBM WebSphere MQ fournit pTAL DEFINE avec des noms qui se terminent par ^DEFAULT pour initialiser les structures IBM WebSphere MQ avec des valeurs par défaut. Les instructions pTAL suivantes sont codées pour affecter des valeurs par défaut aux structures MQMD et MQPMO déclarées:

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

Vous pouvez déclarer et initialiser d'autres structures IBM WebSphere MQ à l'aide d'un code similaire.

pTAL et CRE

Les programmes pTAL ne peuvent pas initialiser l'environnement d'exécution commun et doivent donc être utilisés avec une routine principale en langage C ou COBOL.

Les exemples pTAL fournis avec IBM WebSphere MQ utilisent une routine principale en langage C appelée AMQSPTM0.C

Paramètres avec type de données MQCHAR

Les procédures MQGET, MQPUT et MQPUT1 comportent chacune un paramètre **Buffer** dont le type de données est MQCHAR .EXT . Ce paramètre est utilisé pour envoyer et recevoir les données de message de l'application.

Les paramètres de ce tri sont affichés dans les exemples pTAL sous forme de tableaux de chaînes. 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 procédure est déclaré en tant que MQCHAR .EXT, mais l'adresse de toutes les données peut être spécifiée en tant que paramètre sur l'appel de procédure.

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.

Codage dans Visual Basic

Notez les informations de la section suivante lors du codage des programmes WebSphere MQ dans Visual Basic.

Remarque : En dehors de l'environnement .NET, la prise en charge de Visual Basic (VB) dans WebSphere MQ a été stabilisée au niveau V6.0 . La plupart des nouvelles fonctions ajoutées à WebSphere MQ 7.0 ou version ultérieure ne sont pas disponibles pour les applications VB. Si vous programmez dans VB.NET, utilisez les classes WebSphere MQ .NET. Pour plus d'informations, voir Utilisation de .NET.

Visual Basic est pris en charge uniquement sous Windows.

Pour éviter une traduction non intentionnelle des données binaires transmises entre Visual Basic et WebSphere 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

WebSphere 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 WebSphere 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. WebSphere 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.

Modèle d'objet IBM WebSphere MQ

Le modèle d'objet IBM WebSphere MQ se compose de classes, de méthodes et de propriétés. Utilisez ces informations pour en savoir plus sur chacun de ces concepts.

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

- *Classes* représentant des concepts WebSphere 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 WebSphere MQ .

Lorsque vous créez une application WebSphere MQ à l'aide du modèle d'objet WebSphere MQ , vous créez des instances de ces classes dans le programme. 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).

Ces rubriques décrivent en détail chacun des modèles d'objet WebSphere MQ :

- «Classes», à la page 91
- «Références d'objet», à la page 92
- «Codes retour», à la page 92

Classes

Le modèle d'objet WebSphere 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 WebSphere MQ qui utilise l'interface MQI, WebSphere 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 WebSphere MQ ultérieurs. Avec le modèle d'objet WebSphere 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.

Dois-je utiliser des classes IBM WebSphere MQ pour Java ou des classes IBM WebSphere MQ pour JMS?

Une application Java peut utiliser des classes IBM WebSphere MQ pour Java ou des classes IBM WebSphere MQ pour JMS pour accéder aux ressources IBM WebSphere MQ . Chaque approche a ses avantages.

IBM WebSphere MQ classes for Java encapsule l'interface MQI (Message Queue Interface), l'API IBM WebSphere MQ native, et utilise le même modèle d'objet que les autres interfaces orientées objet, tandis que IBM WebSphere MQ classes for Java Message Service implémente les interfaces JMS (Java Message Service) de Sun.

Si vous connaissez bien IBM WebSphere MQ dans des environnements autres que Java, en utilisant des langages procéduraux ou orientés objet, vous pouvez transférer vos connaissances existantes vers l'environnement Java à l'aide des classes IBM WebSphere MQ pour Java. Vous pouvez également exploiter toute la gamme des fonctions de IBM WebSphere MQ, qui ne sont pas toutes disponibles dans IBM WebSphere MQ classes for JMS.

Si vous n'êtes pas familiarisé avec IBM WebSphere MQou que vous possédez déjà une expérience JMS, il peut être plus facile d'utiliser l'API JMS familière pour accéder aux ressources IBM WebSphere MQ à l'aide de IBM WebSphere MQ classes for JMS. JMS fait également partie intégrante de la plateforme Java Platform, Enterprise Edition (Java EE). Les applications Java EE peuvent utiliser des beans gérés par message (MDB) pour traiter les messages de manière asynchrone, et les MDB ne peuvent traiter que les messages JMS. JMS est également le mécanisme standard permettant à Java EE d'interagir avec des systèmes de messagerie asynchrones tels que IBM WebSphere MQ. Chaque serveur d'applications compatible avec Java EE doit inclure un fournisseur JMS. Par conséquent, vous pouvez utiliser JMS pour communiquer entre les différents serveurs d'applications ou vous pouvez transférer une application d'un fournisseur JMS à un autre sans modifier l'application.

Conception d'applications IBM WebSphere 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 WebSphere MQ.

Lors de la conception d'une application IBM WebSphere 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
- Client
- 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 WebSphere MQ. Pour plus d'informations, voir Introduction à WebSphere MQ Administration Interface (MQAI) et Automatisation des tâches d'administration .

Langage de programmation

IBM WebSphere MQ prend en charge un certain nombre de langages de programmation procéduraux et orientés objet pour l'écriture d'applications. Pour plus d'informations, voir «Choix du langage de programmation à utiliser», à la page 81.

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.

Si vous utilisez C, codez 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 .

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. Pour plus d'informations, voir Clusters de gestionnaires de files d'attente .

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 «Conception de vos messages», à la page 96.

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 WebSphere MQ à une destination standard gérée par IBM WebSphere MQ publish? subscribe, et permet à IBM WebSphere 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 Introduction à la messagerie de publication / abonnement IBM WebSphere MQ .

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 WebSphere 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 WebSphere MQ *déclenchement* , voir «Démarrage des applications IBM WebSphere MQ à l'aide de déclencheurs», à la page 342). 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 WebSphere MQ Evénements d'instrumentation , comme décrit dans Evénements d'instrumentation).

Exécution de votre application sur un client IBM WebSphere MQ

L'interface MQI complète est prise en charge dans l'environnement client, ce qui permet à presque toutes les applications IBM WebSphere MQ d'être réactivées pour s'exécuter sur un client IBM

WebSphere MQ MQI. Liez l'application sur le client IBM WebSphere MQ MQI à la bibliothèque MQIC plutôt qu'à la bibliothèque MQI.

Remarque : Une application exécutée sur un client IBM WebSphere 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 client IBM WebSphere MQ MQI», à la page 374.

Performances de l'application
Les décisions de conception peuvent avoir un impact sur les performances de vos applications. Pour des suggestions d'amélioration des performances des applications IBM WebSphere MQ , voir «Conception et performances des applications», à la page 97.

Techniques IBM WebSphere MQ avancées
Pour les applications plus avancées, vous pouvez utiliser des techniques IBM WebSphere MQ avancées telles que la corrélation des réponses et la génération et l'envoi d'informations de contexte IBM WebSphere MQ . Pour plus d'informations, voir «Techniques IBM WebSphere MQ avancées», à la page 98.

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 WebSphere 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 335 pour plus de détails). Vous pouvez utiliser la fonction de *persistance* des messages IBM WebSphere MQ pour assurer la distribution des messages importants.

Test des applications IBM WebSphere MQ

L'environnement de développement d'application pour les programmes IBM WebSphere 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 WebSphere MQ .

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

IBM WebSphere MQ

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Ecriture d'une application de mise en file d'attente», à la page 202
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 client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Utilisation de .NET», à la page 581
Les classes WebSphere MQ pour .NET permettent à un programme écrit dans l'infrastructure de programmation .NET de se connecter à WebSphere MQ en tant que client WebSphere MQ MQI ou de se connecter directement à un serveur WebSphere MQ .

«Utilisation de C++», à la page 652

WebSphere MQ fournit des classes C++ équivalentes aux objets WebSphere 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 des classes WebSphere MQ pour JMS», à la page 741
WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) est le fournisseur JMS fourni avec WebSphere MQ. Outre l'implémentation des interfaces définies dans le package javax.jms , WebSphere MQ classes for JMS fournit deux ensembles d'extensions à l'API JMS.

«Utilisation des classes WebSphere MQ pour Java», à la page 677
Les classes WebSphere MQ for Java vous permettent d'utiliser WebSphere MQ dans un environnement Java. Une application Java peut utiliser des classes WebSphere MQ pour Java ou des classes WebSphere MQ pour JMS pour accéder aux ressources WebSphere MQ .

«Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX)», à la page 1065
Les WebSphere MQ Automation Classes for ActiveX (MQAX) sont des composants ActiveX qui fournissent des classes que vous pouvez utiliser dans votre application pour accéder à WebSphere MQ.

Conception de vos messages

Tenez compte des aspects indiqués dans ces informations pour vous aider à concevoir des messages.

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 WebSphere 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. Lors de l'écriture d'une application pour WebSphere MQ for Windows ou WebSphere MQ sur les systèmes UNIX and Linux , assurez-vous que vous savez comment votre système a été configuré en ce qui

concerne l'allocation de fichiers journaux afin de réduire le risque de concevoir une application qui s'exécutera dans les limites des fichiers journaux.

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
Si un message doit être inséré dans plusieurs files d'attente, vous pouvez utiliser une liste de distributionou effectuer une publication dans une rubrique.

Conception et 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 qui appellent WebSphere 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.

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

Pour WebSphere 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.

Techniques IBM WebSphere MQ avancées

Pour une application IBM WebSphere MQ simple, vous devez choisir les objets WebSphere 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 279).
- Etablissement d'un exit de rappel à utiliser lorsqu'un message arrive ; voir «Consommation asynchrone de messages IBM WebSphere MQ», à la page 34.
- 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 WebSphere 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 39.

Démarrage automatique des programmes WebSphere MQ

Utilisez WebSphere MQ *déclenchement* pour démarrer un programme automatiquement 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 WebSphere MQ à l'aide de déclencheurs», à la page 342. 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 nonWebSphere MQ .

WebSphere MQ peut définir des objets de service pour démarrer les programmes WebSphere MQ lorsque le gestionnaire de files d'attente démarre ; voir Objets de service.

Génération de rapports WebSphere 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 12.

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 .

Exemples de programmes WebSphere MQ

Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

- «Exemples de programmes pour les plateformes réparties», à la page 100

Concepts associés

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Ecriture d'une application de mise en file d'attente», à la page 202
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 client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Utilisation des services Web dans WebSphere MQ», à la page 977
Vous pouvez développer des applications IBM WebSphere MQ pour les services Web à l'aide du transport IBM WebSphere MQ pour SOAP ou du pont IBM WebSphere MQ pour HTTP.

«Ecriture d'applications de publication / abonnement», à la page 290
Commencez à écrire des applications de publication / abonnement WebSphere MQ .

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

«Traitement des erreurs de programme», à la page 569
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.

Exemples de programmes pour les plateformes réparties

Cette rubrique décrit les exemples de programmes fournis avec IBM WebSphere MQ, écrits en C et COBOL. Les exemples illustrent des utilisations typiques de l'interface MQI (Message Queue Interface).

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. Toutefois, ces exemples peuvent être utilisés comme base pour vos propres programmes de mise en file d'attente de messages.

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.

Exemples de programmes C++: Voir Utilisation de C++ pour une description des exemples de programmes disponibles en C + +.

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. Pour configurer le gestionnaire de files d'attente afin qu'il accepte les connexions client, voir «Préparation et exécution des exemples de programmes», à la page 115 pour plus de détails.

Utilisez les liens suivants pour en savoir plus sur les exemples de programmes:

- «Fonctions démontrées dans les exemples de programme», à la page 102
- «Exemples de programmes de publication / abonnement», à la page 141
- «Exemples de programmes Put», à la page 146
- «Exemple de programme Liste de distribution», à la page 133
- «Exemples de programmes de navigation», à la page 121
- «Exemple de programme de navigateur», à la page 122
- «Exemples de programmes Get», à la page 135
- «Exemples de programmes de message de référence», à la page 147
- «Exemples de programmes de demande», à la page 153
- «Exemples de programmes Inquire», à la page 140
- «Exemple de programme d'interrogation des propriétés d'un descripteur de message», à la page 141
- «Exemples de programme Set», à la page 157
- «Exemples de programmes Echo», à la page 134
- «Exemple de programme de conversion de données», à la page 125
- «Exemples de programmes de déclenchement», à la page 161
- «Exemple de programme d'insertion asynchrone», à la page 120
- «Exemples de coordination de base de données», à la page 125
- «Exemple de transaction CICS», à la page 123
- «Exemples TUXEDO», à la page 162
- «Exemple de gestionnaire de files d'attente de rebut», à la page 133
- «Exemple de programme Connect», à la page 124
- «Exemple de programme d'exit API», à la page 118
- «Utilisation de l'exit de sécurité SSPI sur les systèmes Windows», à la page 176
- «Exécution des exemples à l'aide de files d'attente distantes», à la page 177

- «Exemple de programme de surveillance de file d'attente de cluster (AMQSCLM)», à la page 177
- «Exemple de programme de recherche de noeud final de connexion (CEPL)», à la page 187

Fonctions démontrées dans les exemples de programme

Collection de tables présentant les techniques démontrées par les exemples de programme WebSphere 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.

Exemples pour les systèmes UNIX and Linux

Cette rubrique présente les techniques démontrées par les exemples de programme pour WebSphere MQ sur les systèmes UNIX and Linux .

Voir «Préparation et exécution d'exemples de programmes sur les systèmes UNIX», à la page 117 pour savoir où sont stockés les exemples de programmes pour WebSphere MQ sur les systèmes UNIX et Linux .

Tableau 14, à la page 102 Le tableau répertorie les fichiers source C et COBOL qui sont fournis et indique si un exécutable serveur ou client est inclus.

Tableau 14. WebSphere MQ sur des exemples de programmes UNIX and Linux démontrant l'utilisation de l'interface MQI (C et COBOL)

Technique	**C (source) («1», à la page 105)**	**COBOL (source) («2», à la page 105)**	**Serveur (exécutable C)**	**Client (exécutable C) («3», à la page 105)**
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	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Insertion de messages dans une liste de distribution («4», à la page 105)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Réponse à un message de demande	amqsinqa	amqminqx amqiinqx	amqsinq	aucun échantillon
Obtention de messages (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 («4», à la page 105)	amqsprma	aucun échantillon	amqsprm	amqsprmc

Tableau 14. WebSphere MQ sur des exemples de programmes UNIX and Linux démontrant l'utilisation de l'interface MQI (C et COBOL) (suite)

Technique	**C (source) («1», à la page 105)**	**COBOL (source) («2», à la page 105)**	**Serveur (exécutable C)**	**Client (exécutable C) («3», à la page 105)**
Obtention de messages de référence à partir d'une file d'attente («4», à la page 105)	amqsgrma	aucun échantillon	amqsgrm	amqsgrmc
Exit de canal de message de référence («4», à la page 105)	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
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	(«5», à la page 105)	aucun échantillon	aucun échantillon	aucun échantillon
WebSphere 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
WebSphere 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 («6», à la page 105)	amqscic0.ccs	aucun échantillon	amqscic0	aucun échantillon
Transaction Encina («4», à la page 105)	amqsxae0	aucun échantillon	amqsxae0	aucun échantillon

Tableau 14. WebSphere MQ sur des exemples de programmes UNIX and Linux démontrant l'utilisation de l'interface MQI (C et COBOL) (suite)

Technique	**C (source) («1», à la page 105)**	**COBOL (source) («2», à la page 105)**	**Serveur (exécutable C)**	**Client (exécutable C) («3», à la page 105)**
Transaction TUXEDO permettant d'insérer des messages («7», à la page 105)	amqstxpx	aucun échantillon	aucun échantillon	aucun échantillon
Transaction TUXEDO pour obtenir des messages («7», à la page 105)	amqstxgx	aucun échantillon	aucun échantillon	aucun échantillon
Serveur pour TUXEDO («7», à la page 105)	amqstxsx	aucun échantillon	aucun échantillon	aucun échantillon
gestionnaire de files d'attente de rebuts	Répertoire `./tools/c/Samples/dlq` («8», à la page 105)	aucun échantillon	amqsdlq	aucun échantillon
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 SSL/TLS sur MQCONNX	amqssslc	aucun échantillon	non applicable	amqssslc

Tableau 14. WebSphere MQ sur des exemples de programmes UNIX and Linux démontrant l'utilisation de l'interface MQI (C et COBOL) (suite)

Technique	C (source) («1», à la page 105)	COBOL (source) («2», à la page 105)	Serveur (exécutable C)	Client (exécutable C) («3», à la page 105)

Remarques :

1. La version exécutable des exemples de client WebSphere MQ MQI partage la même source que les exemples exécutés dans un environnement 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. Les versions exécutables des exemples de client WebSphere MQ MQI ne sont pas disponibles sur WebSphere MQ for HP-UX.
4. Pris en charge sur WebSphere MQ for AIX, WebSphere MQ for HP-UXet WebSphere MQ for Solaris uniquement.
5. Sous WebSphere MQ for AIX, WebSphere MQ for HP-UXet WebSphere MQ for Solaris, ce programme est appelé `amqsvfc0.c`
6. CICS est pris en charge par WebSphere MQ for AIX et WebSphere MQ for HP-UX uniquement.
7. TUXEDO n'est pas pris en charge par WebSphere MQ for Linux on System p.
8. La source du gestionnaire de files d'attente de rebut se compose de plusieurs fichiers et est fournie dans un répertoire distinct.

Des informations détaillées sur la prise en charge des systèmes UNIX and Linux sont disponibles sur la page de la configuration système requise pour WebSphere MQ à l'adresse Configuration système requise pour IBM WebSphere MQ.

Exemples pour le client IBM WebSphere MQ pour HP Integrity NonStop Server

Cette rubrique présente les techniques mises en oeuvre par les exemples de programme pour le client IBM WebSphere MQ sur les systèmes HP Integrity NonStop Server .

Tableau 15, à la page 105Le tableau répertorie les exemples de programme source C, COBOL et pTAL fournis.

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL

Technique	C				COBOL		pTAL	
	OSS (Source)	OSS (Exécutable)	Gardien (Source)	Tuteur (Exécutable)	OSS (Source)	Gardien (Source)	OSS (Source)	Gardien (Source)
Utilisation de l'interface de publication / abonnement	amqspuba.c amqssbxa.c amqssuba.c amqspse0.c	amqspubc amqssbxc amqssubc	MQSPUBC MQSSBXC MQSSUBC	AMQSPUBC AMQSSBXC AMQSSUBC	amq0pub0.cbl amq0sub0.cbl	MQSPUBL MQSSUBL	amqtpub0.tal amqtsub0.tal	MQSPUBT MQSSUBT

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	**C**				**COBOL**		**pTAL**	
Insertion de messages à l'aide de l'appel MQPUT	amqsput0.c	amqsputc	MQSPUTC	amqsputc	amq0put0.cbl	MQSPUTL	amqtput0.tal	MQSPUTT
Insertion d'un message unique à l'aide de l'appel MQPUT1	amqsecha.c	amqsechc	MQSECHC	AMQSECHC			amqtech0.tal	MQSECHT
Insertion de messages dans une liste de distribution	amqsptl0.c	amqsptlc	MQSPTLC	AMQSPTLC	amq0ptl0.cbl	MQSPTLL		
Réponse à un message de demande	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Obtention de messages (pas d'attente)	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Obtention de messages (attente avec une limite de temps)	amqsget0.c	amqsgetc	MQSGETC	AMQSGETC	amq0get0.cbl	MQSGETL	amqtget0.tal	MQSGETT
Obtention de messages (attente illimitée)	amqstrg0.c	amqstrgc	MQSTRGC	Commande AMQSTRGC				

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	**C**				**COBOL**		**pTAL**	
Obtention de messages (avec conversion de données)	amqsecha.c	amqsechc	MQSECHC	AMQSECHC				
Insertion de messages de référence dans une file d'attente	amqsprma.c	amqsprmc	MQSPRMC	AMQSPRMC				
Obtention de messages de référence à partir d'une file d'attente	amqsgrma.c	amqsgrmc	MQSGRMC	AMQSGRMC				
Exit de canal de message de référence	amqsqrma.c amqsxrma.c		MQSQRMC MQSXRMC					
Navigation dans les 20 premiers caractères d'un message	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Exploration des messages complets	amqsbcg0.c	amqsbcgc	MQSBCGC	AMQSBCGC				
Utilisation d'une file d'entrée partagée	amqsinqa.c	amqsinqc	MQSINQC	MQSINQC				

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	C				COBOL		pTAL	
Utilisation d'une file d'attente en entrée exclusive	amqstrg0.c	amqstrgc	MQSTRGC	Commande AMQSTRGC				
Utilisation de l'appel MQINQ	amqsinqa.c	amqsinqc	MQSINQC	AMQSINQC				
Utilisation de l'appel MQSET	amqsseta.c	amqssetc	MQSSETC	AMQSSETC				
Utilisation d'une file d'attente de réponse	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
Demande d'exceptions de message	amqsreq0.c	amqsreqc	MQSREQC	AMQSREQC	amq0req0.cbl	MQSREQL		
Acceptation d'un message tronqué	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Utilisation d'un nom de file d'attente résolue	amqsgbr0.c	amqsgbrc	MQSGBRC	AMQSGBRC	amq0gbr0.cbl	MQSGBRL		
Déclenchement d'un processus	amqstrg0.c	amqstrgc	MQSTRGC	Commande AMQSTRGC				
Utilisation de la conversion de données	amqsvfc0.c							

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	**C**				**COBOL**		**pTAL**	
Gestionnaire de files d'attente de rebut (1)	Répertoire ./samp/dlq							
Connexion à un gestionnaire de files d'attente à l'aide de MQCONNX	amqscnxc.c	amqscnxc	MQSCNXC					
Utilisation des exits API	amqsaxe0.c amqsaem0.c							
Exit d'équilibrage de charge de cluster	amqswlm0.c		MQSWLMC					
Moniteur de file d'attente de cluster	amqsclma.c							
Insertion de messages de manière asynchrone et obtention du statut à l'aide de l'appel MQSTAT	amqsapt0.c	amqsaptc	MQSAPTC	MQSAPTC				

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	**C**				**COBOL**		**pTAL**	
Clients reconnectables	amqsghac.c amqsmhac.c amqsphac.c	amqsghac amqsmhac amqsphac	MQSGHAC MQSMHAC MQSPHAC MQSFHAC	AMQSGHAC AMQSMHAC AMQSPHAC AMQSFHAC				
Utilisation des consommateurs de messages pour consommer de manière asynchrone des messages provenant de plusieurs files d'attente	amqscbf0.c	amqscbfc						
Spécification des informations de connexion SSL/TLS sur MQCONNX	amqssslc.c	amqssslc	MQSSSLC	AMQSSSLC				
Trace d'activité	amqsact0.c	amqsactc	MQSACTC	AMQSACTC				
Propriétés des messages	amqsiqma.c amqsstma.c	amqsiqmc amqsstmc	MQSIQMC MQSSTMC	AMQSIQMC AMQSSTMC				
Serveur de commandes	amqsstop.c		MQSSTOC					

Tableau 15. Exemples de programmes IBM WebSphere MQ on HP Integrity NonStop Server illustrant l'utilisation de C, COBOL et pTAL (suite)

Technique	C				COBOL		pTAL	
Evénements du journal	amqslog0.c	amqslogc	MQSLGC	AMQSLOGC				
Comptabilité	amqsmon0.c	amqsmonc	MQSMONC	AMQSMONC				
Interface d'administration	amqsaicq.c amqsaiem.c amqsailq.c							
Exemple de fonction principale de langage C pour l'appel de pTAL			MQSPTMC					

Remarques :

1. La source du gestionnaire de files d'attente de rebut se compose de plusieurs fichiers et est fournie dans un répertoire distinct.
2. Pour plus d'informations sur le développement d'applications pour votre client IBM WebSphere MQ sur la plateforme HP Integrity NonStop Server, voir :
   - «Génération de votre application sur HP Integrity NonStop Server», à la page 452
     - «Préparation des programmes C dans HP Integrity NonStop Server», à la page 455
     - «Préparation de programmes COBOL», à la page 456
     - «Préparation des programmes pTAL», à la page 457

Exemples pour IBM WebSphere MQ for Windows

Cette rubrique présente les techniques démontrées par les exemples de programme pour IBM WebSphere MQ for Windows.

Tableau 16, à la page 111Le tableau répertorie les fichiers source C et COBOL qui sont fournis et indique si un exécutable serveur ou client est inclus.

Tableau 16. Exemples de programmes IBM WebSphere 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

Tableau 16. Exemples de programmes IBM WebSphere 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)
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
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

Tableau 16. Exemples de programmes IBM WebSphere 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)**
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
WebSphere 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
WebSphere 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 114)	aucun échantillon	amqsdlq	aucun échantillon
A partir d'un client WebSphere MQ MQI, insertion d'un message	aucun échantillon	aucun échantillon	aucun échantillon	amqsputc
A partir d'un client WebSphere MQ 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
Equilibrage de charge de cluster	amqswlm0	aucun échantillon	amqswlm	aucun échantillon
Routines de sécurité SSPI	amqsspin	aucun échantillon	amqrspin.dll	amqrspin.dll

Tableau 16. Exemples de programmes IBM WebSphere 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)
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 SSL/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.

Exemples Visual Basic pour IBM WebSphere MQ for Windows

Cette rubrique présente les techniques mises en oeuvre par les exemples de programmes Visual Basic pour IBM WebSphere MQ for Windows.

Le Tableau 17, à la page 114 présente les techniques mises en évidence par les exemples de programmes IBM WebSphere 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 WebSphere MQ . Pour savoir comment modifier les exemples de projets afin de les utiliser avec les clients IBM WebSphere MQ , voir «Préparation des programmes Visual Basic sous Windows», à la page 481.

Tableau 17. Exemples de programmes IBM WebSphere 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
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

Tableau 17. Exemples de programmes IBM WebSphere MQ for Windows illustrant l'utilisation de MQI (Visual Basic) (suite)

Technique	Nom du fichier de projet
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

Préparation et exécution des exemples de programmes

Configurez votre gestionnaire de files d'attente pour qu'il accepte en toute sécurité les demandes de connexion entrantes provenant des applications exécutées 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 comme suit:

```
DISPLAY QMGR CHLAUTH
```

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», à la page 115 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 115.

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:

   qmgr est le nom de votre gestionnaire de files d'attente
   nnnn est le numéro de port que vous avez choisi

   a) Pour les systèmes UNIX et Windows :

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

   ```
   DEFINE CHANNEL('channel-name') CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Channel for use by sample programs')
   ```

   nom_canal est le nom du canal.
4. Créez une règle d'authentification de canal autorisant uniquement l'adresse IP de votre système client à utiliser le canal en émettant la commande MQSC:

```
SET CHLAUTH('channel-name') TYPE(ADDRESSMAP) ADDRESS('client-machine-IP-address') +
MCAUSER('non-privileged-user-id')
```

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 115

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:

```
DEFINE QLOCAL('queue-name') DESCR('Queue for use by sample programs')
```

nom_file_attente est le nom de la file d'attente.

6. Accordez l'accès pour vous connecter au gestionnaire de files d'attente et l'interroger:
   a) Pour les systèmes UNIX etWindows , émettez les commandes MQSC:

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

non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 115

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:
   a) Pour les systèmes UNIX et Windows , émettez les commandes MQSC:

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

nom_file_attente est le nom de la file d'attente.

non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 115

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:
   a) Pour les systèmes UNIX et Windows , émettez les commandes MQSC:

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

non-privileged-user-id est l'ID utilisateur que vous avez obtenu à l'étape «1», à la page 115

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.

Tâches associées

Octroi de l'accès à un objet WebSphere MQ sur les systèmes UNIX ou Linux et Windows

Référence associée

SET CHLAUTH

De la définition d'un canal

DEFINE QLOCAL
SET AUTHREC

Préparation et exécution d'exemples de programmes sur les systèmes UNIX

Tableau 18. Où trouver les exemples pour WebSphere MQ sur les systèmes UNIX 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 WebSphere MQ est installé.	

Les exemples de fichiers WebSphere MQ sur les systèmes UNIX and Linux se trouvent dans les répertoires répertoriés dans Tableau 18, à la page 117 si les valeurs par défaut ont été utilisées lors de l'installation. 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. Pour plus d'informations sur la procédure à suivre, voir «Exécution des exemples de programme», à la page 118.

Préparation et exécution d'exemples de programmes sur les systèmes Windows

Tableau 19. Où trouver les exemples pour WebSphere 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 WebSphere MQ est installé.

Remarque :

1. Des versions 64 bits de certains exemples de fichiers exécutables C sont disponibles.

Les exemples de fichiers WebSphere MQ for Windows se trouvent dans les répertoires répertoriés dans Tableau 19, à la page 117 si les valeurs par défaut ont été utilisées lors de l'installation ; les valeurs par défaut de l'unité d'installation sont < c: >. Pour exécuter les exemples, utilisez les versions exécutables fournies ou compilez les versions source comme vous le feriez pour les autres applications WebSphere MQ for Windows . Pour plus d'informations sur la procédure à suivre, voir «Exécution des exemples de programme», à la page 118.

Exécution des exemples de programme

Utilisez cette rubrique lorsque vous exécutez des exemples de programme sur différentes plateformes.

Avant de pouvoir exécuter l'un des exemples de programme, créez un gestionnaire de files d'attente et configurez les définitions par défaut. Ceci est expliqué dans Administration.

Sur les plateformes Windows, UNIX et Linux

Les exemples ont besoin d'un ensemble de files d'attente à utiliser. Utilisez vos propres files d'attente ou exécutez l'exemple de fichier MQSC amqscos0.tst pour créer un ensemble.

Pour ce faire sur les systèmes UNIX 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.

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.

Vous pouvez maintenant exécuter les modèles d'application. Entrez le nom de l'exemple d'application 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`.

Consultez la description des exemples individuels pour plus d'informations sur les paramètres attendus par chacun d'entre eux.

Longueur du nom de la file d'attente

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.

Exemples d'interrogation, de définition et d'écho

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, les systèmes UNIX and Linux le font en éditant le fichier `amqscos0.tst` et en remplaçant les noms des fichiers exécutables C par les noms des fichiers exécutables COBOL avant d'utiliser la commande **`runmqsc`** , comme indiqué précédemment.

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», à la page 405.

Source
amqsaxe0.c

Binaire
amqsaxe

Configuration pour l'exemple d'exit

1. Ajoutez ce qui suit au 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 WebSphere 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 WebSphere MQ.

2. Définir la variable d'environnement

   ```
   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 exécutable client et serveur binaire sur les plateformes Windows, UNIX and Linux .

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 : Le *type de reconnexion* 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 WebSphere 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 de l'unité d'exécution et du stockage est effectuée par WebSphere 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 programme», à la page 102 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 programme», à la page 102 .

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.

Systèmes UNIX, Linux et Windows

Utilisez cette rubrique lorsque vous apprenez à parcourir les exemples de programme sur les systèmes UNIX, Linux et 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.

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 programme», à la page 102 .

Ce programme utilise les paramètres suivants:

1. Nom de la file d'attente source
2. Nom du gestionnaire de files d'attente
3. Paramètre facultatif pour les propriétés.

Les deux premiers paramètres d'entrée de ce programme sont obligatoires. Par exemple, démarrez le programme de l'une des manières 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)

Les valeurs admises pour le paramètre de propriété sont les suivantes:

Valeur	Comportement
0	Comportement par défaut, tel qu'il était pour V6. 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. Par 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 peuvent être renvoyées selon qu'une propriété de version 6 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 *msg tronqué* si un message plus long est lu.

Voir Administration pour un exemple de la sortie de cet utilitaire.

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 «Génération d'une application IBM WebSphere MQ», à la page 446 .

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. Les échecs sont envoyés à la file d'attente SYSTEM.SAMPLE.CICS.DLQ.

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]] [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

SvrconnChannelNom
Nom du canal de connexion serveur

QMgrName
Nom du gestionnaire de files d'attente cible

Si vous ne spécifiez pas le nom de connexion TCP/IP, MQCONNX est émis avec *ClientConnPtr* défini sur NULL. 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. 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 programme», à la page 102 .

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 expliquer comment WebSphere MQ peut coordonner les mises à jour WebSphere MQ et les mises à jour de base de données au sein d'une même unité de travail.

Ces exemples sont les suivants:

1. AMQSXAS0 (en C) ou AMQ0XAS0 (en COBOL), qui met à jour une base de données unique dans une unité de travail WebSphere 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 dans une unité de travail WebSphere MQ , indiquant comment accéder à plusieurs bases de données. Ces exemples sont fournis pour illustrer l'utilisation de l'appel MQBEGIN, d'appels SQL mixtes et WebSphere MQ , ainsi que l'emplacement et le moment de la connexion à une base de données.

Figure 18, à la page 126 montre comment les exemples fournis sont utilisés pour mettre à jour les bases de données:

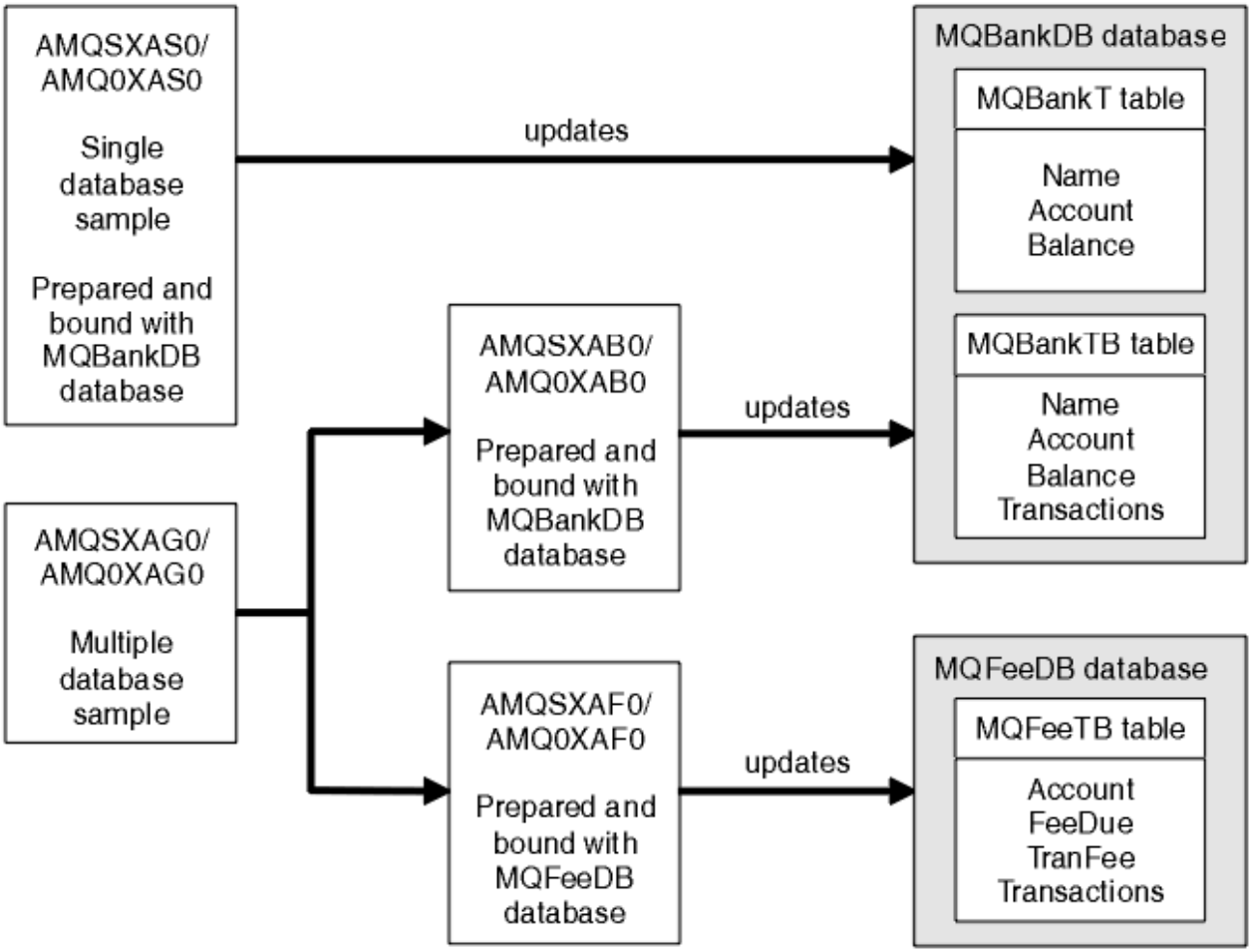


Figure 18. Exemples de coordination de base de données

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 WebSphere 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 WebSphere MQ est installé.

- Sous AIX, le chemin de répertoire est:

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

- Sous HP-UX et Solaris, le chemin de répertoire est le suivant:

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

- 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:

```
c:\sqllib
```

Remarque : Avant d'exécuter la commande link sur les systèmes Windows , vérifiez que la variable d'environnement LIB contient des chemins d'accès aux bibliothèques DB2 et WebSphere MQ .

Copiez les fichiers suivants dans un répertoire temporaire:

- Le fichier `amqsxag0.c` de votre installation WebSphere MQ

  Remarque : Ce fichier se trouve dans les répertoires suivants:

  – Sur les systèmes UNIX and Linux :

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – 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

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

- HP-UX

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

- Solaris

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

- Systèmes Windows

```
cl /c /IMQ_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

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

- HP-UX Révision 11i

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

- Solaris

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

- 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

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

- HP-UX Révision 11i

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

- Solaris

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

- Systèmes Windows

```
link util.obj amqsxas0.obj mqm.lib db2api.lib /out:amqsxas0.exe
```

Informations supplémentaires

Si vous travaillez sous AIX ou HP-UX 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 savoir comment procéder, voir «Scénario 1 : Le gestionnaire de files d'attente effectue la coordination», à la page 44.

Les titres suivants fournissent des informations sur la façon d'exécuter des exemples en C et COBOL:

- «Echantillons C», à la page 131
- «Exemples COBOL», à la page 132

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_
Définir sur YES ou sur 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 WebSphere MQ .

Le manuel Administration ou *System Management Guide* de votre plateforme explique comment fonctionne le gestionnaire de rebut et comment vous l'exécutez.

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 programme», à la page 102 .

Les programmes sont destinés à s'exécuter en tant que programmes déclenchés.

Sur les systèmes UNIX, Linux et 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.

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, 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, 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.

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 programme», à la page 102 .

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 deux paramètres:

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 une variable d'environnement ou au fichier de définition de canal du client.

Pour exécuter ces programmes, entrez l'un des éléments suivants:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

où `myqueue` est le nom de la file d'attente à partir de laquelle le programme va extraire des messages et `qmanagername` est le gestionnaire de files d'attente propriétaire de `myqueue`.

Si vous omettez `qmanagername`, les programmes prennent la valeur par défaut ou, dans le cas du client MQI, le gestionnaire de files d'attente identifié par une variable d'environnement ou le fichier de définition de canal du client.

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 que l'exemple reste simple et qu'il soit facile à configurer, les exemples de programmes qui se reconnectent à un gestionnaire de files d'attente à instance unique qui est démarré, arrêté puis redémarré sont affichés ; voir «Configurer et contrôler le gestionnaire de files d'attente», à la page 138.

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 WebSphere 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*** **[*qMgrNom*]**

- **amqsghac** est une application IBM WebSphere 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 WebSphere 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 WebSphere MQ MQI client . Il vérifie qu'un gestionnaire de files d'attente multi-instance IBM WebSphere 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 *QueueManagerNom*. Il se reconnecte automatiquement en cas de basculement du gestionnaire de files d'attente.
- Elle place *InTransactionCount***RepeatCount* messages persistants dans *QueueName* pendant lequel vous faites basculer le gestionnaire de files d'attente autant de fois que nécessaire. **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 messages *InTransactionNombre* 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 *SideQueueNom* 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.
- Faites varier 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. Voir Vérification d'une installation 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 19, à la page 137.

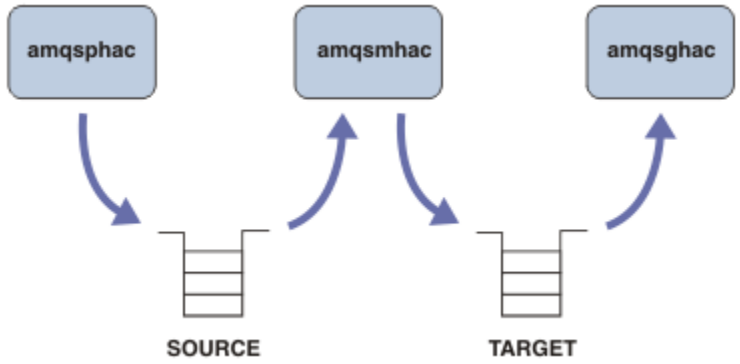


Figure 19. Exemples de client reconnectable

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, entrez **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 139 dans «Configurer et contrôler le gestionnaire de files d'attente», à la page 138.)
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.

Configurer et contrôler le gestionnaire de files d'attente

1. Créez le gestionnaire de files d'attente.

```
C:\>crtmqm QM1
WebSphere MQ queue manager created.
Directory 'C:\IBM\MQ\MQ7\Data\qmgrs\QM1' created.
Creating or replacing default objects for QM1.
Default objects statistics : 67 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

   Mémorisez le répertoire de données pour définir la variable **MQCHLLIB** ultérieurement.

2. Démarrez le gestionnaire de files d'attente.

```
C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

3. 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: WebSphere MQ queue created.
     2 : DEFINE QLOCAL(TARGET) REPLACE
AMQ8006: WebSphere MQ queue created.
     3 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) MCAUSER(MUSR_MQADMIN)
REPLACE
AMQ8014: WebSphere MQ channel created.
     4 : DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) CONNAME('LOCALHOST(2345)')
QMNAME(QM1) REPLACE
AMQ8014: WebSphere MQ channel created.
     5 : ALTER LISTENER(SYSTEM.DEFAULT.LISTENER.TCP) TRPTYPE(TCP) PORT(2345)
AMQ8623: WebSphere MQ listener changed.
     6 : START LISTENER(SYSTEM.DEFAULT.LISTENER.TCP)
AMQ8021: Request to start WebSphere MQ Listener accepted.
     7 : START CHANNEL(CHANNEL1)
AMQ8018: Start WebSphere MQ channel accepted.
7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4. 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 138et 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.
WebSphere MQ queue manager 'QM1' ending.
WebSphere MQ queue manager 'QM1' ended.

C:\>strmqm QM1
WebSphere MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
WebSphere MQ queue manager 'QM1' started.
```

amqsphac

```
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnectedmessage
<Message 3>
message <Message 4>
message <Message 5>
```

amqsmhac

```
Sample AMQSMHA0 start
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
No more messages.
Sample AMQSMHA0 end
C:\>
```

amqsghac

```
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
16:25:22 : EVENT : Connection Reconnecting (Delay: 0ms)
16:25:45 : EVENT : Connection Reconnecting (Delay: 0ms)
16:26:02 : EVENT : Connection Reconnected
message <Message 3>
message <Message 4>
message <Message 5>
```

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 programme», à la page 102 .

Ces programmes étant destinés à être exécutés en tant que programmes déclenchés, leur seule entrée est une structure MQTMC2 (message de déclenchement) pour les systèmes IBM i, Windowset UNIX and Linux . 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. 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:

- Pour les systèmes UNIX, Linux et Windows , démarrez le programme **runmqtrm** dans une session, puis le programme `amqsreq` dans une autre.

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.

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*) contenue 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\WebSphere 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 / abonnement dans WebSphere MQ.

Il existe trois exemples de programmes en langage C qui illustrent comment programmer l'interface de publication / abonnement WebSphere 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 WebSphere 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 «Préparation et exécution des exemples de programmes», à la page 115 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` .

Six types d'exemple sont répertoriés dans le Tableau 20, à la page 142.

Tableau 20. 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 .
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.

Java

L'exemple Java `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 «Préparation et exécution des exemples de programmes», à la page 115 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 WebSphere MQ Explorer Eclipse pour 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 Java MQPubSubApiSample», à la page 143 montrent comment exécuter l'exemple à partir de l'espace de travail Eclipse .

Exécution de l'exemple Java MQPubSubApiSample

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 «Préparation et exécution des exemples de programmes», à la page 115 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 Java client WebSphere MQ MQI. 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 en fonction de différentes configurations WebSphere MQ . L'exemple est illustré sous Windows. Les chemins d'accès aux fichiers diffèrent sur les autres plateformes.

Procédure

1. Importer les exemples de programmes 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** > **projet Java**.
   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 WebSphere 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 WebSphere MQ .

k) Dans le panneau **Importer** , Figure 20, à la page 144, 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**.

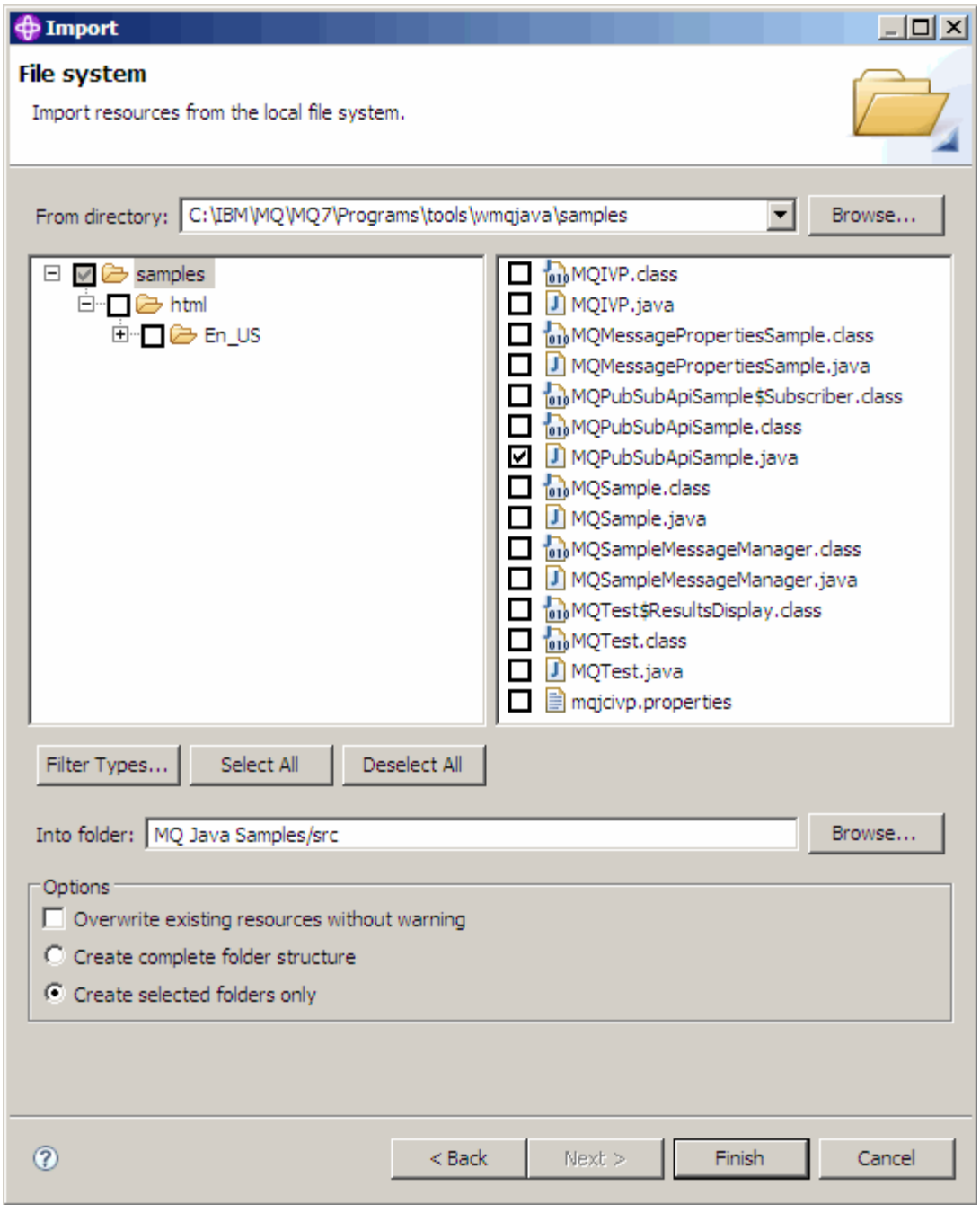


Figure 20. Importation du système de fichiers

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.javaRun-as** > **1. Application Java**

- 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 `-D` JVM ou en fournissant une valeur par défaut pour la propriété système en 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 les propriétés système, codez une valeur par défaut dans l'ID d'accès, 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 Application Java 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:
   - Sur les systèmes UNIX 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 WebSphere MQ est installé. Sous Windows, définissez les attributs équivalents dans le registre.
2. Assurez-vous que le module est accessible à WebSphere 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 :
   - Sur les systèmes UNIX and Linux , vérifiez que le répertoire`/var/mqm/trace` existe et exportez la variable d'environnement suivante:

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Sous Windows, vérifiez que le répertoire `C:\temp` existe et définissez la variable d'environnement suivante:

     ```
     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 programme», à la page 102 .

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

Ces programmes prennent chacun 2 paramètres:

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 une variable d'environnement ou au fichier de définition de canal du client. Pour exécuter ces programmes, entrez l'un des éléments suivants:

- `amqsput myqueue qmanagername`
- `amqsputc 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`.

Exécution de l'exemple amq0put

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.

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 de stocker l'objet dans des files d'attente WebSphere 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, en fonction de notre exemple, amqsxrm. Les sections suivantes décrivent les exemples de programme de message de référence.

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 la réception si le récepteur s'exécute sous un produit WebSphere MQ à l'exception de WebSphere MQ for IBM i.

Exécution des exemples de message de référence

Utilisez ces informations pour apprendre à exécuter des exemples de programmes de message de référence.

Les exemples de message de référence s'exécutent comme suit:

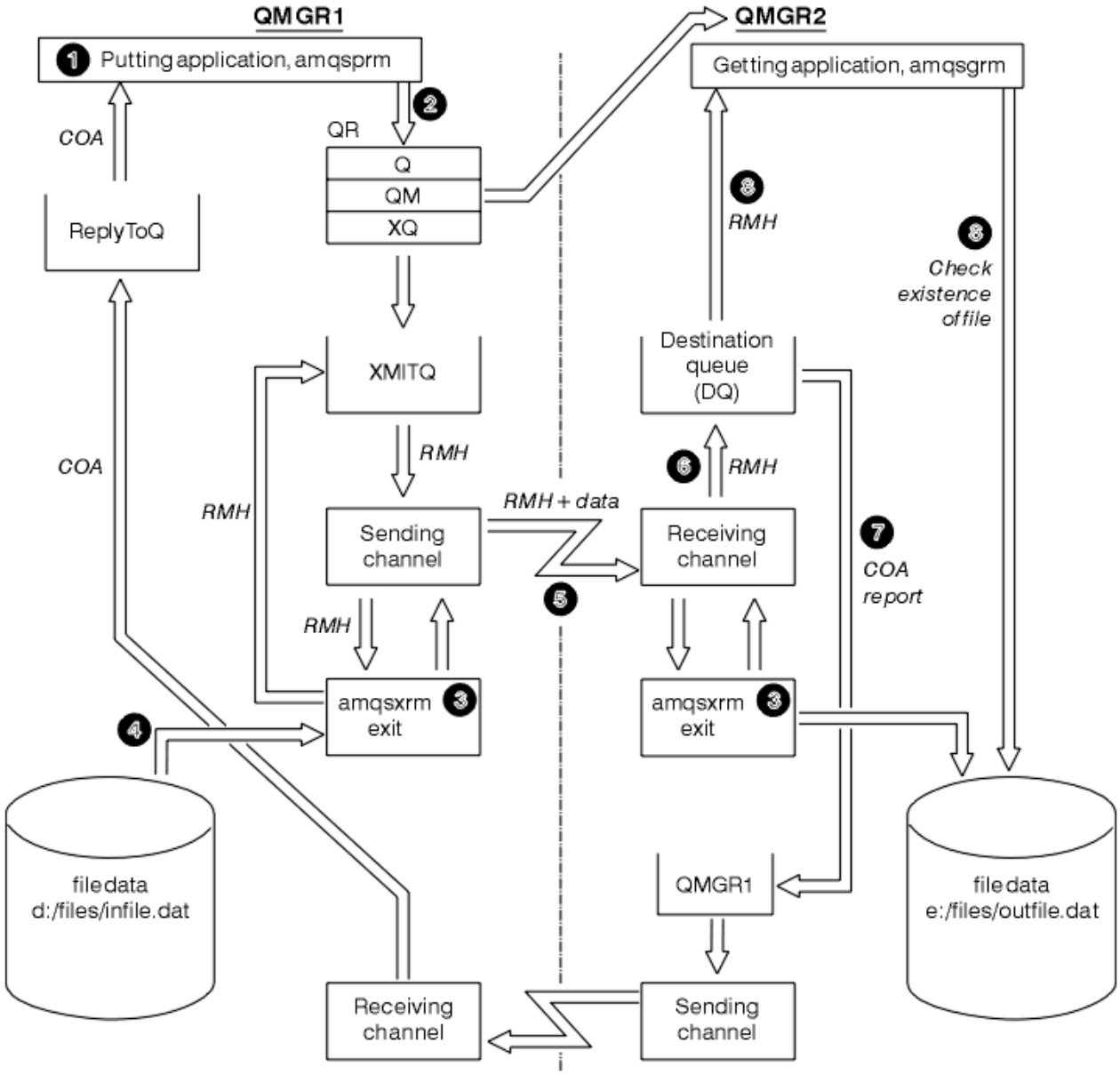


Figure 21. Exécution des exemples de message de référence

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 de l'exemple de message de référence, il s'agit de la machine émettrice sous la forme MACHINE1 avec un gestionnaire de files d'attente appelé QMGR1 et de la machine réceptrice sous la forme 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 pour 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) 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 WebSphere MQ .

Les définitions MQSC ont utilisé un style AIX pour définir les exits. 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
```

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)
```

2. Une fois les objets WebSphere 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 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. Par défaut, il s'agit du 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
```

Remarque : Pour les plateformes UNIX 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. Il est défini sur WebSphere MQ for Windows comme suit:

```
msgexit('pathname\amqsxrm.dll(MsgExit)')
```

Défini sur WebSphere MQ for AIX, WebSphere MQ for HP-UXet WebSphere MQ for Solaris, 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, dans WebSphere MQ for Windows, dans le chemin indiqué 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. Sous WebSphere MQ for AIX, WebSphere MQ for HP-UXet WebSphere MQ for Solaris, 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 WebSphere 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. A titre d'exemple, l'exemple fourni avec WebSphere 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 UNIX 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

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Pour compiler amqsxrma, utilisez les commandes suivantes:

Sous AIX

```
xlc_r -q64 -e MsgExit -bE:amqsxrm.exp -bM:SRE -o amqsxrm_64_r
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -lmqm_r amqsqrma.c
```

sous HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o amqsxrma.o amqsqrma.c -IMQ_INSTALLATION_PATH/inc
$ ld -b amqsxrma.o -o /var/mqm/exits64/amqsxrma -LMQ_INSTALLATION_PATH/lib64
-L/usr/lib/pa20_64 -lmqm_r -lpthread
```

ActivéLinux

```
$ gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
```

Sous Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm

 -lsocket
-lnsl -ldl
```

Sous Windows

WebSphere MQ fournit désormais la bibliothèque mqm avec des packages client ainsi que des packages serveur. L'exemple suivant utilise donc `mqm.lib` au lieu de `mqmvx.lib`:

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Pour des informations générales sur l'écriture et la compilation des exits de canal, voir «Ecriture de programmes d'exit de canal», à la page 416

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 programme», à la page 102 .

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.

Systèmes UNIX, Linux et Windows

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 22, à la page 155 montre comment utiliser les exemples Request et Inquire ensemble.

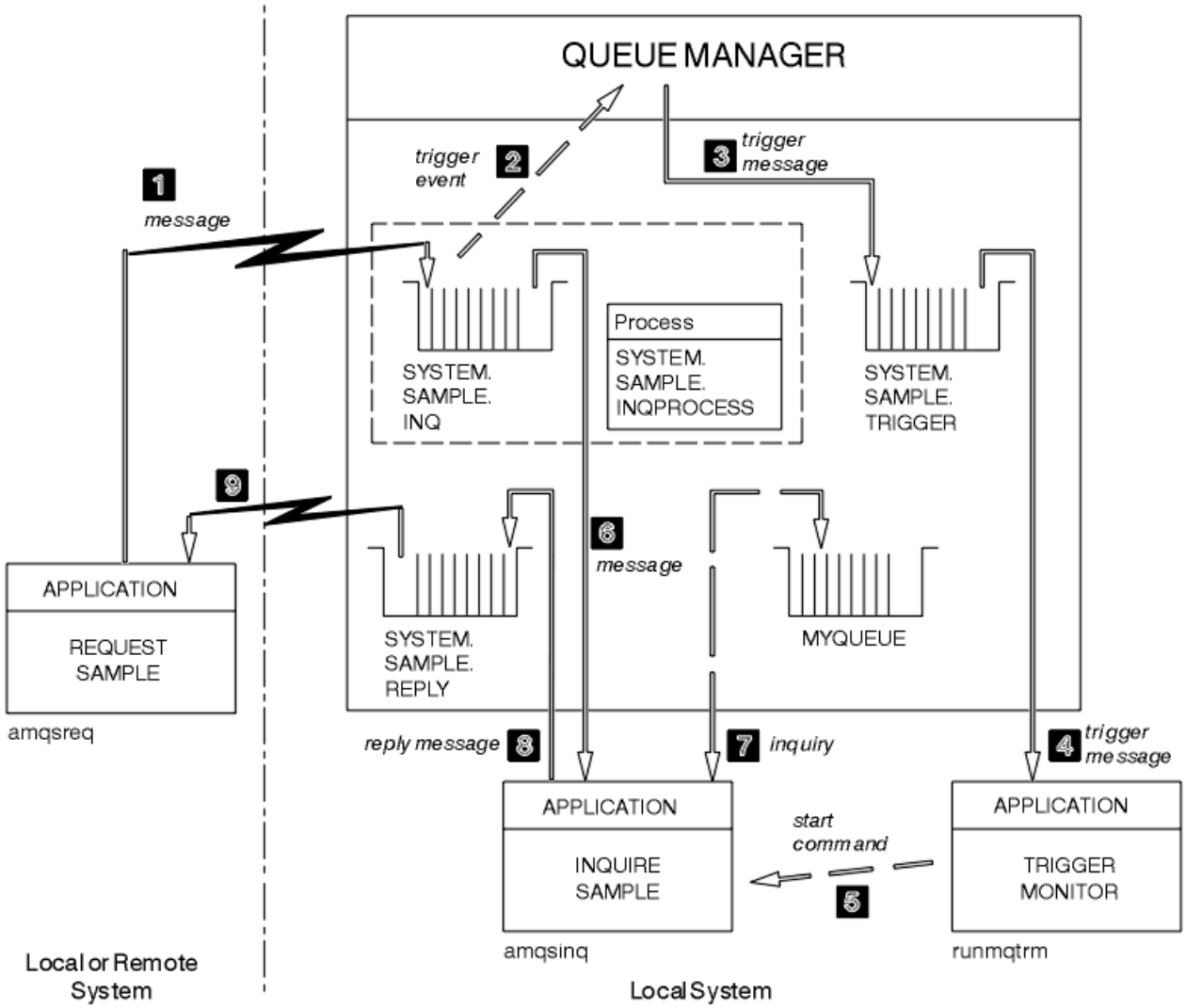


Figure 22. Demander et demander des exemples à l'aide du déclenchement

Dans Figure 22, à la page 155 , 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 22, à la page 155 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 156.

Pour plus d'informations sur le déclenchement, voir «Démarrage des applications IBM WebSphere MQ à l'aide de déclencheurs», à la page 342.

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.

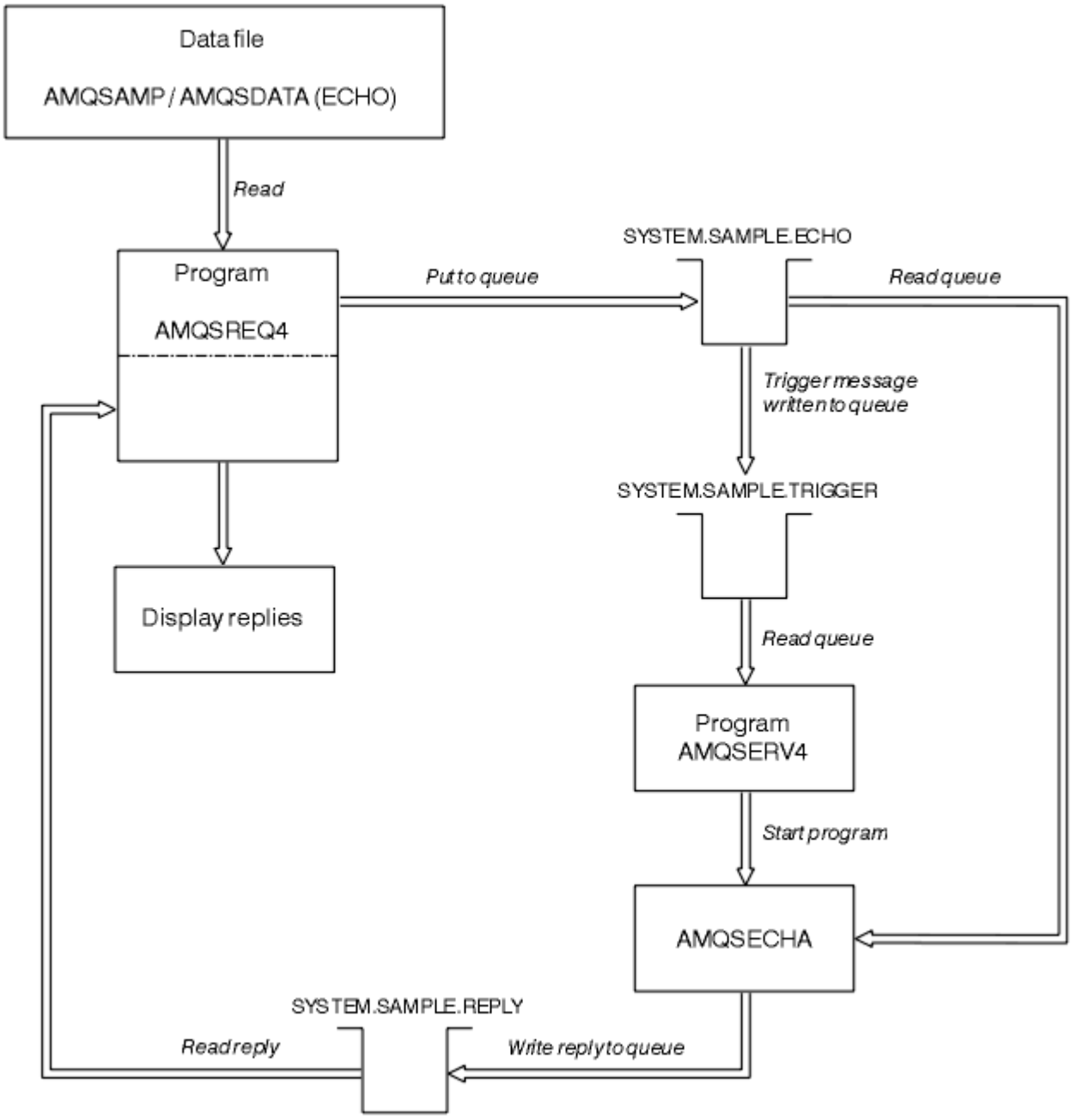


Figure 23. Exemple de diagramme de programme client-serveur IBM i (Echo)

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 programme», à la page 102 .

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:

- Pour les systèmes UNIX, Linux et Windows , démarrez le programme **runmqtrm** dans une session, puis démarrez le programme amqsreq dans une autre.
- 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*) contenue 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 SSL/TLS

AMQSSLC est un exemple de programme C qui montre comment utiliser les structures MQCNO et MQSCO pour fournir des informations de connexion client SSL/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 SSL/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. Ensuite, il se connecte simplement à un gestionnaire de files d'attente local et ignore les informations de connexion client

AMQSSLC 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 SSL/TLS:

-k KeyReposTige
Nom de radical du fichier de référentiel de clés. Il s'agit du chemin d'accès complet au fichier sans le suffixe .kdb. Exemple :

```
/home/user/client
C:\User\client
```

-s CipherSpec
Chaîne CipherSpec du canal SSL/TLS correspondant à SSLCIPH dans 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 strophe 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.

Paramètre de révocation de certificat OCSP:

-o URL
URL du répondeur OCSP

Exécution de l'exemple de programme SSL/TLS

Pour exécuter l'exemple de programme SSL/TLS, vous devez d'abord configurer votre environnement SSL ou 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 Configuration d'un référentiel de clés sur les systèmes UNIX, Linux, and Windows.
3. Créez un référentiel de clés pour le client. Appelez-le *clientkey.kdb*.
4. Créez un certificat personnel pour le gestionnaire de files d'attente. Pour plus d'informations, voir Création d'un certificat personnel autosigné sur les systèmes UNIX, 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 autosigné à partir d'un référentiel de clés sur les systèmes UNIX, Linux et Windows et Ajout d'un certificat de l'autorité de certification (ou de la partie publique d'un certificat autosigné) dans un référentiel de clés sur les systèmes UNIX, Linux ou 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(NULL_SHA)
   ```

   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\WebSphere MQ\clientkey" -s NULL_SHA
    -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 SSL CipherSpec NULL_SHA
Using SSL key repository stem c:\Program Files\IBM\WebSphere MQ\clientkey
Using OCSP responder URL http://dummy.OCSP.responder
Connection established to queue manager QM1

Sample AMQSSSLC end
```

Si 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 la liste des codes raison, voir Codes raison 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 programme», à la page 102 .

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 UNIX, Linuxet 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). Sur les systèmes UNIX, Linux et Windows , la zone *EnvData* est utilisée comme extension de la chaîne de commande appelante.

Enfin, le programme ferme la file d'attente d'initialisation.

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** .

Conception du serveur de déclenchement

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
- 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.

Exemples TUXEDO

Découvrez les exemples de programmes Put et Get pour TUXEDO et générez l'environnement de serveur dans TUXEDO.

Avant d'exécuter ces exemples, vous devez générer l'environnement de serveur.

Remarque : Dans cette rubrique, le caractère barre oblique inversée (\) est utilisé pour fractionner les commandes longues sur plusieurs lignes. N'entrez pas ce caractère. Entrez chaque commande sur une seule ligne.

Génération de l'environnement de serveur

Informations sur la génération de l'environnement de serveur pour WebSphere MQ pour différentes plateformes.

Il est supposé que vous disposez d'un environnement TUXEDO fonctionnel.

Génération de l'environnement de serveur pour WebSphere MQ for AIX (32 bits)

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 WebSphere 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 ce qui suit 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
```

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 WebSphere MQ for AIX (64 bits)

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 WebSphere 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 ce qui suit 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
```

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 WebSphere MQ for Solaris (32 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

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:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. Ajoutez ce qui suit au fichier TUXEDO udataobj/RM.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a
```

4. Exécutez les commandes suivantes :

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs  -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so  \

$ buildclient -o dogets  -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. 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 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
```

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 WebSphere MQ for Solaris (64 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

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:

```
$ export CFLAGS="-I /<APPDIR>"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR□lib:MQ_INSTALLATION_PATH□lib:□lib64
$ export LD_LIBRARY_PATH=$TUXDIR□lib64:MQ_INSTALLATION_PATH□lib64:□lib64
```

3. Ajoutez ce qui suit au fichier TUXEDO udataobj/RM.

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a
```

4. Exécutez les commandes suivantes :

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs  -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so  \

$ buildclient -o dogets  -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. 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 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
```

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 WebSphere MQ for HP-UX (32 bits)

Remarque : L'environnement de serveur TUXEDO 32 bits ne peut être construit que sur la plateforme Itanium.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

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:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Ajoutez ce qui suit au fichier TUXEDO udataobj/RM

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.so MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

4. Exécutez les commandes suivantes :

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   Après l'exécution des commandes mkfldhdr et viewc, le fichier d'en-tête amqstxvx.h est créé dans le répertoire d'application TUXEDO. Copiez ce fichier depuis le répertoire d'application TUXEDO dans le répertoire d'inclusion TUXEDO, puis exécutez les commandes suivantes.

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
```

5. 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
```

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 WebSphere MQ for HP-UX (64 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

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:

```
$ export CFLAGS="-Aa -D_HPUX_SOURCE"
$ export FIELDTBLS=MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ export VIEWFILES=$APPDIR/amqstxvx.V
$ export TUXCONFIG=$APPDIR/tuxconfig
$ export PATH=$TUXDIR/bin:/usr/bin:/sbin:MQ_INSTALLATION_PATH/bin:$PATH
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib64:/lib64
$ export FLDTBLDIR=$APPDIR:$TUXDIR/udataobj
```

3. Ajoutez ce qui suit au fichier TUXEDO udataobj/RM

   Sur la plateforme HP-UX IA64 (IPF):

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.so MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib/libtux.sl
```

   Remarque : Les bibliothèques WebSphere MQ fournies sur la plateforme HP-UX IA64 (IPF) ont une extension de nom de fichier .so.

4. Exécutez les commandes suivantes :

```
$ mkfldhdr    MQ_INSTALLATION_PATH/samp/amqstxvx.flds
$ viewc       MQ_INSTALLATION_PATH/samp/amqstxvx.v
```

   Après l'exécution des commandes mkfldhdr et viewc, le fichier d'en-tête amqstxvx.h est créé dans le répertoire d'application TUXEDO. Copiez ce fichier depuis le répertoire d'application TUXEDO dans le répertoire d'inclusion TUXEDO, puis exécutez les commandes suivantes.

```
$ buildtms    -o MQXA -r MQSeries_XA_RMI
```

   Sur la plateforme HP-UX IA64 (IPF):

```
$ buildserver -o MQSERV1 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
$ buildserver -o MQSERV2 -f MQ_INSTALLATION_PATH/samp/amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSeries_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
$ buildclient -o doputs  -f MQ_INSTALLATION_PATH/samp/amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
$ buildclient -o dogets  -f MQ_INSTALLATION_PATH/samp/amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
```

5. 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
```

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 WebSphere MQ for Windows (32 bits)

Remarque : Remplacez les zones identifiées par <> dans les zones suivantes par les chemins de répertoire:

< MQMDIR>	le chemin de répertoire spécifié lors de l'installation de WebSphere MQ , par exemple `g:\Program Files\IBM\WebSphere MQ`
< REP_TUX>	le chemin de répertoire spécifié lors de l'installation de TUXEDO, par exemple `f:\tuxedo`
< IRAP>	le chemin de répertoire à utiliser pour l'exemple d'application, par exemple `f:\tuxedo\apps\mqapp`

Pour générer l'environnement de serveur et les exemples:

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 WebSphere MQ vers le répertoire de l'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 24, à la page 170) 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
```

où < MQMDIR > est remplacé comme indiqué dans l'exemple précédent. Bien qu'elle soit représentée ici sur deux lignes, la nouvelle entrée doit correspondre à une ligne du 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
   ```

   où <*APPDIR*> est remplacé.

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.

```
*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 24. Exemple de fichier ubbstxcn.cfg pour WebSphere MQ for Windows

Remarque : Modifiez les noms de répertoire et les chemins de répertoire pour qu'ils correspondent à 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. Les autres informations que vous devez ajouter sont identifiées par des caractères <> .

L'exemple de fichier ubbconfig pour WebSphere MQ for Windows est répertorié dans Figure 24, à la page 170. Il est fourni sous la forme ubbstxcn.cfg dans le répertoire des exemples WebSphere MQ .

L'exemple de fichier makefile (voir Figure 25, à la page 171) fourni pour WebSphere MQ for Windows s'appelle ubbstxmn.maket se trouve dans le répertoire des exemples WebSphere MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSeries_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figure 25. Exemple de fichier makefile TUXEDO pour WebSphere MQ for Windows

Génération de l'environnement de serveur pour WebSphere MQ for Windows (64 bits)

Remarque : Remplacez les zones identifiées par <> dans les zones suivantes par les chemins de répertoire:

< MQMDIR>	le chemin de répertoire spécifié lors de l'installation de WebSphere MQ , par exemple `g:\Program Files\IBM\WebSphere MQ`
< REP_TUX>	le chemin de répertoire spécifié lors de l'installation de TUXEDO, par exemple `f:\tuxedo`
< IRAP>	le chemin de répertoire à utiliser pour l'exemple d'application, par exemple `f:\tuxedo\apps\mqapp`

Pour générer l'environnement de serveur et les exemples:

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 WebSphere MQ vers le répertoire de l'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 26, à la page 173) 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\lib64\mqmxa64.lib <MQMDIR>\tools\lib64\mqm.lib
   ```

   où < MQMDIR > est remplacé. Bien qu'elle soit représentée ici sur deux lignes, la nouvelle entrée doit correspondre à une ligne du 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
   ```

   où <*APPDIR*> est remplacé comme illustré dans l'exemple précédent.
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.

```
*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 26. Exemple de fichier ubbstxcn.cfg pour WebSphere MQ for Windows

Remarque : Modifiez les noms de répertoire et les chemins de répertoire pour qu'ils correspondent à 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. Les autres informations que vous devez ajouter sont identifiées par des caractères <> .

L'exemple de fichier ubbconfig pour WebSphere MQ for Windows est répertorié dans Figure 26, à la page 173. Il est fourni sous la forme ubbstxcn.cfg dans le répertoire des exemples WebSphere MQ .

L'exemple de fichier makefile (voir Figure 27, à la page 174) fourni pour WebSphere MQ for Windows s'appelle ubbstxmn.maket se trouve dans le répertoire des exemples WebSphere MQ .

```
TUXDIR  = f:\tuxedo
MQMDIR  = g:\Program Files\IBM\WebSphere MQ
APPDIR  = f:\tuxedo\apps\mqapp
MQMLIB  = $(MQMDIR)\tools\lib64
MQMINC  = $(MQMDIR)\tools\c\include
MQMSAMP = $(MQMDIR)\tools\c\samples
INC = -f "-I$(MQMINC) -I$(APPDIR)"
DBG = -f "/Zi"

amqstx.exe:
 $(TUXDIR)\bin\mkfldhdr    -d$(APPDIR) $(MQMSAMP)\amqstxvx.fld
 $(TUXDIR)\bin\viewc       -d$(APPDIR) $(MQMSAMP)\amqstxvx.v
 $(TUXDIR)\bin\buildtms    -o MQXA -r MQSeries_XA_RMI
 $(TUXDIR)\bin\buildserver -o MQSERV1 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT1:MPUT -s MGET1:MGET
 $(TUXDIR)\bin\buildserver -o MQSERV2 -f $(MQMSAMP)\amqstxsx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG) \
                           -r MQSeries_XA_RMI \
                           -s MPUT2:MPUT -s MGET2:MGET
 $(TUXDIR)\bin\buildclient -o doputs -f $(MQMSAMP)\amqstxpx.c \
                           -f $(MQMLIB)\mqm.lib -v $(INC) $(DBG)
 $(TUXDIR)\bin\buildclient -o dogets -f $(MQMSAMP)\amqstxgx.c \
                           -f $(MQMLIB)\mqm.lib $(INC) -v $(DBG)
 $(TUXDIR)\bin\tmloadcf    -y $(APPDIR)\ubbstxcn.cfg
```

Figure 27. Exemple de fichier makefile TUXEDO pour WebSphere MQ for Windows

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.

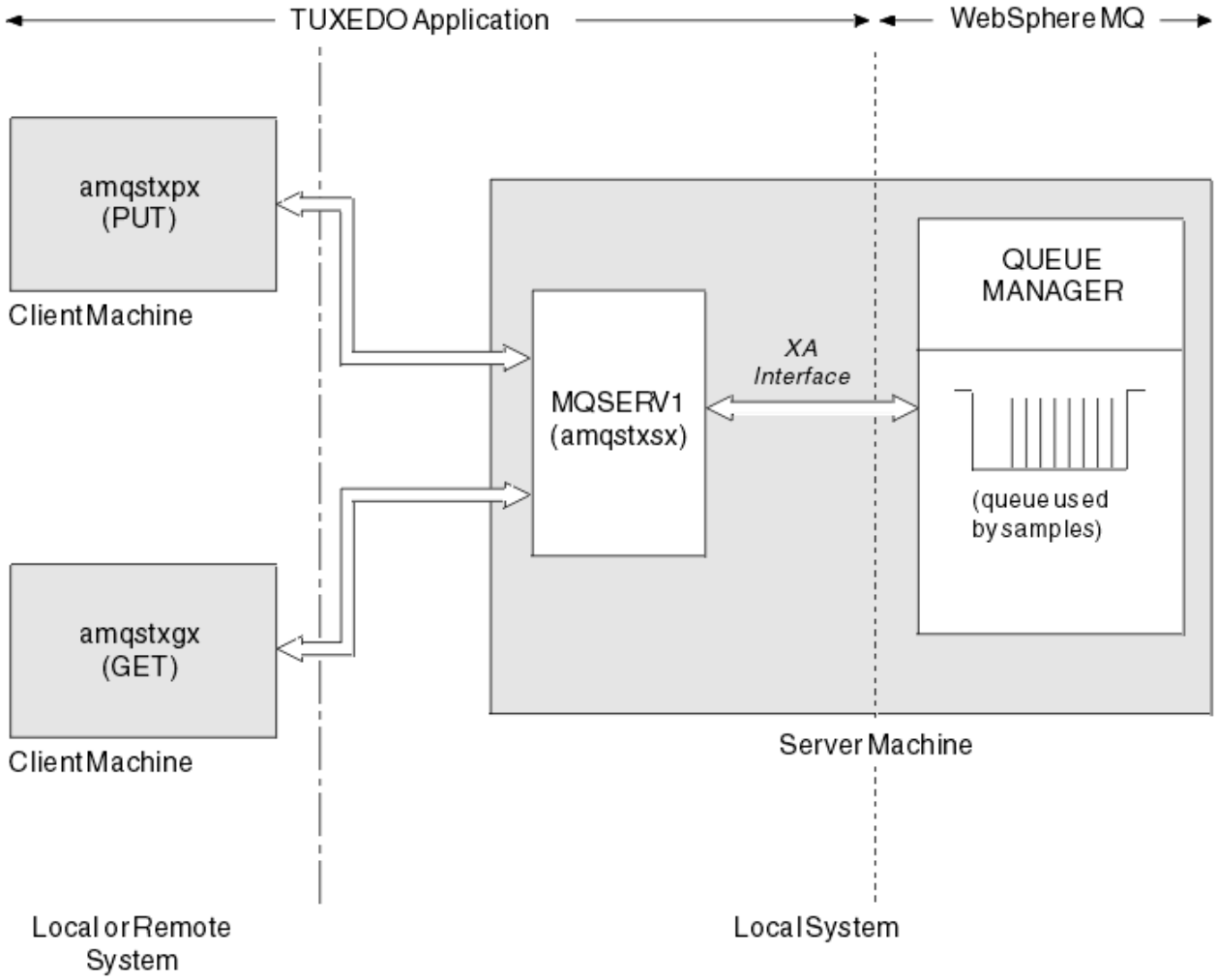


Figure 28. Comment les exemples TUXEDO fonctionnent ensemble

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.

Obtenir un exemple pour TUXEDO

Cet exemple permet d'extraire des messages d'une file d'attente par lots.

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:

• `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.

Utilisation de l'exit de sécurité SSPI sur les systèmes 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 WebSphere MQ for Windows dans le dossier *MQ_INSTALLATION_PATH*`/exits/`*INSTALLATION_NAME* . Par exemple, `C:\Program Files\IBM\WebSphere 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 nom servicePrincipalpour 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\WebSphere 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, ainsi que la bibliothèque Visual C++ et le dossier `include` sont disponibles dans votre chemin. Exemple :

```
cl /VERBOSE /LD /MT /I<path_to_Microsoft_platform_SDK\include>
/I<path_to_WebSphere MQ\tools\c\include> amqsspin.c /DSECURITY_WIN32
-link  /DLL /EXPORT:SCY_KERBEROS /EXPORT:SCY_NTLM STACK:8192
```

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 WebSphere 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 WebSphere 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 de cluster avec plusieurs instances dans un cluster, WebSphere 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 les messages doivent être envoyés. WebSphere MQ envoie des messages aux instances disponibles d'une file d'attente avec la valeur **CLWLPRTY** 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 WebSphere MQ propage la propriété **CLWLPRTY** mise à jour d'une file d'attente de cluster à tous les gestionnaires de files d'attente concernés du cluster. Par 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 en plaçant des messages (à l'aide de **MQPUT**) dans la même file d'attente de cluster. Ce transfert permet à la gestion de charge de travail de cluster WebSphere 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 WebSphere 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 WebSphere MQ requises. Par 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 WebSphere MQ envoie par défaut des messages aux instances de files d'attente en 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 WebSphere 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

Pour exécuter l'exemple de surveillance de file d'attente de cluster, vous devez configurer le gestionnaire de files d'attente afin qu'il accepte en toute sécurité les demandes de connexion entrantes provenant des applications exécutées en mode client.

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**

   ```
   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 WebSphere 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 **RUNMQSC** suivant pour définir l'exemple de surveillance de file d'attente de cluster en tant que service IBM WebSphere MQ .

```
define service(AMQSCLM) +
    descr('Active Cluster Queue Message Distribution Monitor - AMQSCLM') +
    control(qmgr) +
    servtype(server) +
    startcmd('<Install Root>\tools\c\samples\Bin\AMQSCLM.exe') +
 startarg('-m +QMNAME+ -c CLUSTER1 -q ABC* -r AMQSCLM.CONTROL.QUEUE -l c:\AMQSCLM\rpts') +
 stdout('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stdout.log') +
 stderr('C:\AMQSCLM\rpts\+QMNAME+.TSTCLUS.stderr.log')
```

où < Install Root > est l'emplacement de votre installation.

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 supplémentaires
-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 la liste des noms de file d'attente des 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 < fn>. Pour chaque combinaison de gestionnaire de files d'attente et de file d'attente, un fichier de rapport est généré qui est bridé à une certaine taille. Le consignateur écrit toujours dans le même fichier, mais il conserve également les deux versions précédentes du fichier. < /fn> 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 propriété **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.*
```

```
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 WebSphere 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 WebSphere 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 des fichiers de rapport plus anciens, 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
- A 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 instance active, ou des instances

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: Ce scénario peut être dû au fait que AMQSCLM ne s'exécute pas 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.

Exemple de programme de recherche de noeud final de connexion (CEPL)

L'exemple IBM WebSphere MQ Recherche de noeud final de connexion fournit un module d'exit simple mais puissant qui offre aux utilisateurs WebSphere MQ un moyen d'extraire des définitions de connexion d'un référentiel LDAP tel que Tivoli Directory Server.

Le client Tivoli Directory Server v6.3 doit être installé pour pouvoir utiliser CEPL.

Une connaissance pratique de l'administration de WebSphere MQ sur les plateformes prises en charge est requise pour utiliser cet exemple.

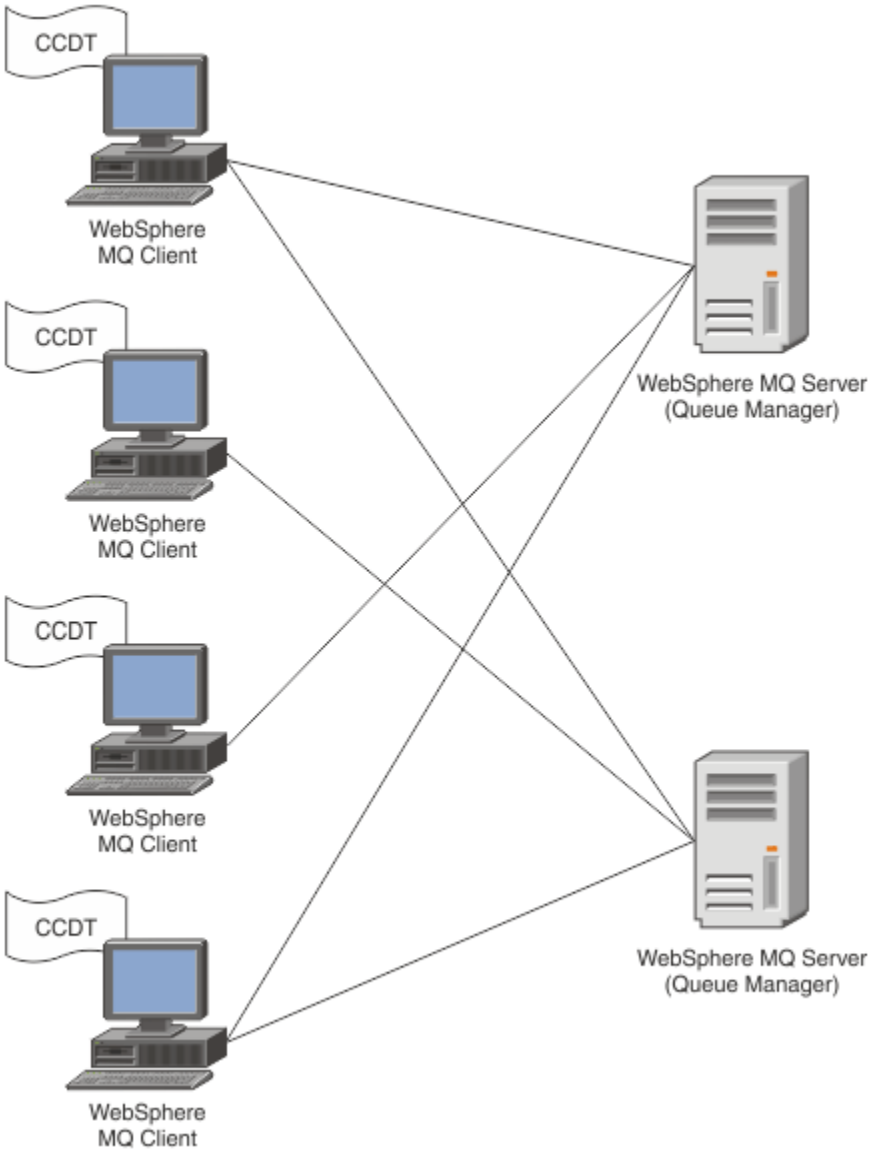
Introduction

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 WebSphere 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 MQSC WebSphere MQ 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.

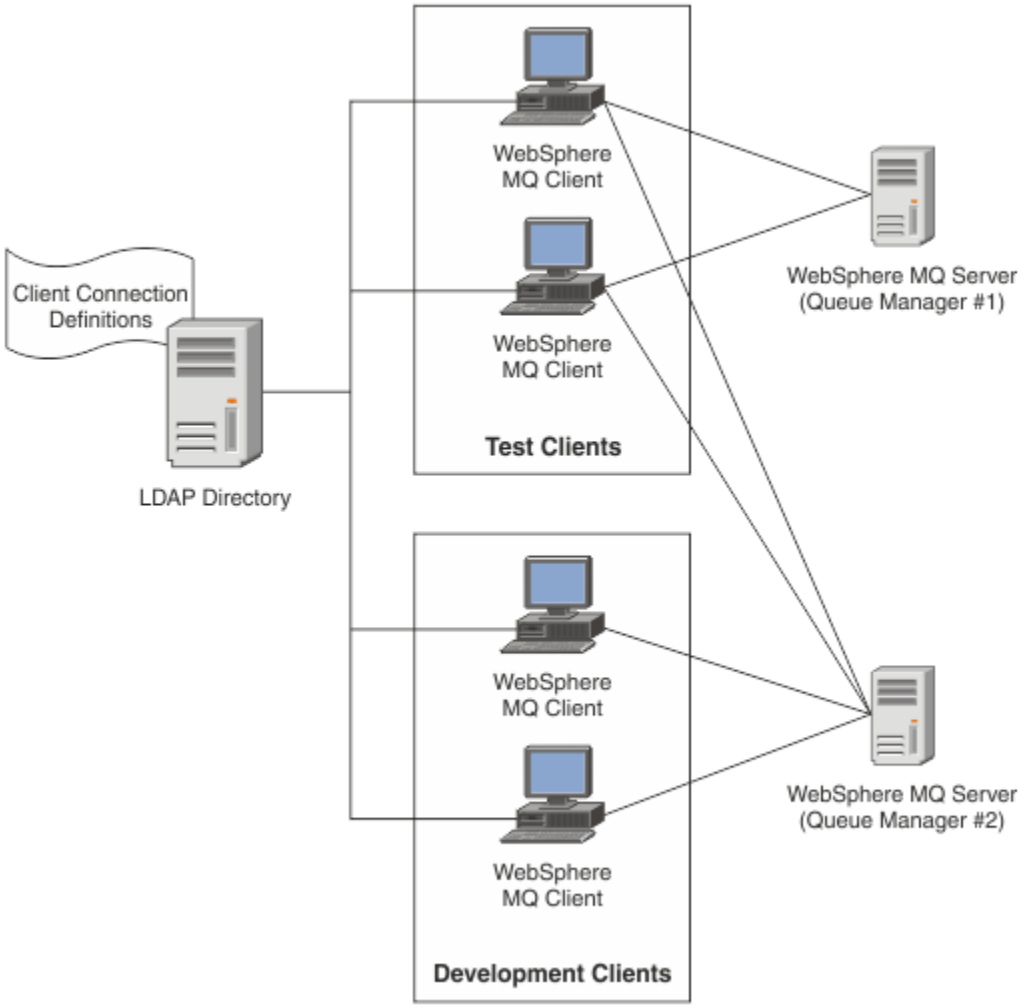


Le fichier CCDT doit être distribué à chaque client WebSphere MQ . Là où des milliers de clients peuvent exister localement ou globalement, il serait bientôt difficile de les maintenir et de les administrer. Une approche plus souple est nécessaire pour s'assurer que chaque client dispose des définitions 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 WebSphere 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 WebSphere MQ.
- Clients nouveaux ou existants qui souhaitent utiliser directement ou optimiser leur modèle de messagerie en ligne avec l'architecture métier LDAP en cours.

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 WebSphere MQ Connection requiert les logiciels suivants:

- IBM WebSphere MQ V7.0ou ultérieure
- Tivoli Directory Server V6.3 Client ou version ultérieure

Systèmes d'exploitation pris en charge:

1. Windows (XP/2003/2008)
2. Solaris (SPARC et x86-64)
3. AIX
4. Linux
   - RHEL v4 et v5 sur System p
   - SUSE v9 et v10 sur System p
   - RHEL v4 et v5 System x32 bits et x64 bits
   - SUSE v9 et v10 System x32 bits et x64 bits
5. HP IA64.

Remarque : L'exemple de programme n'est pas disponible pour les plateformes z/OS, i/5et HP PARISC.

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 WebSphere 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/<install name>/` pour pouvoir être utilisé. Pour les plateformes 64 bits, le module d'exit doit être copié dans exit64/<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 utilisé.

- Vérifiez qu' IBM Tivoli Directory Server est en cours d'exécution, puis copiez ou transférez 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 l'ID LDAP et le mot de passe LDAP sont le nom distinctif racine et le mot de passe du serveur LDAP:

  ldapadd -D "ID LDAP" -w "mot de passe LDAP" -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-amqClientConnexion

Consultez la documentation de votre serveur LDAP pour plus de détails sur l'importation du fichier de schéma.

Configuration

Une nouvelle section intitulée **PreConnect** doit être ajoutée au fichier de configuration du client, à savoir le fichier *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 en l'état *exit* ou le dossier *exit64* dans l'installation WebSphere MQ .

Fonction : nom du point d'entrée fonctionnel dans la bibliothèque qui contient le code d'exit PreConnect . La définition de fonction est conforme au prototype MQ_PRECONNECT_EXIT.

Données : 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=PreConnectExit
  Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
  Sequence=1
```

Présentation de l'exit et du schéma

Syntaxe et paramètres utilisés pour établir une connexion à un gestionnaire de files d'attente.

WebSphere MQ v7.5 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 WebSphere MQ C charge le module d'exit contenant une implémentation de la syntaxe de 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

pExitParamètres

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.

CodepComp

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.

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 */
};
```

Exemple de code pour la génération de l'exit de recherche de noeud final de connexion

Fragments de code permettant de compiler la source sous Windows et les plateformes réparties.

Compilation de la source

Vous pouvez compiler la source avec n'importe quelle bibliothèque client LDAP, par exemple IBM Tivoli Directory Server 6.3 Client. Cette documentation suppose que vous utilisez les bibliothèques client Tivoli Directory Server 6.3 .

Remarque : La bibliothèque d'exit de préconnexion a été testée 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 sous Windowset sur d'autres plateformes réparties:

Compilation de l'exit sur la plateforme Windows

Vous pouvez utiliser le fragment suivant pour compiler la source d'exit sous Windows:

```
CC=cl.exe
LL=link.exe
CCARGS=/c /I. /DWIN32 /W3 /DNDEBUG /EHsc /D_CRT_SECURE_NO_DEPRECATE /Zl

# The libraries to include
LDLIBS=Ws2_32.lib Advapi32.lib libibmldapstatic.lib libibmldapdbgstatic.lib  \
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib \
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib msvcrt.lib

OBJS=amqlcel0.obj

all: amqlcelp.dll

amqlcelp.dll: $(OBJS)
        $(LL) /OUT:amqlcelp.dll /INCREMENTAL /NOLOGO /DLL /SUBSYSTEM:WINDOWS /MACHINE: X86 /
DEF:amqlcelp.def $(OBJS) $(LDLIBS) /NODEFAULTLIB:msvcrt.lib

# The exit source
amqlcel0.obj: amqlcel0.c
              $(CC) $(CCARGS) $*.c
```

Remarque : Des avertissements peuvent s'afficher lors de la compilation des bibliothèques client IBM Tivoli Directory Server 6.3 avec le compilateur Microsoft Visual Studio 2005 ou supérieur, si vous utilisez les bibliothèques client IBM Tivoli Directory Server 6.3 compilées avec le compilateur Microsoft Visual Studio 2003.

Compilation de l'exit sur d'autres plateformes réparties

Vous pouvez utiliser le fragment suivant pour compiler la source d'exit sur d'autres plateformes réparties, par exemple, Linux. Certaines options de compilation peuvent différer sur d'autres plateformes réparties.

```
#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 une seule forme de bibliothèques peut être utilisée. 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)
```

Appel du module d'exit

Le module d'exit PreConnect peut être appelé avec trois codes anomalie différents. Cette section décrit plus en détail chaque raison d'exit.

MQXR_INIT

L'exit est appelé avec le code anomalie MQXR_INIT pour l'initialisation et l'établissement de la connexion à un serveur LDAP.

Avant l'appel *MQXR_INIT* , la zone *pExitDataPtr* de la structure MQNXP aurait été remplie avec l'attribut Data de la strophe PreConnect dans le fichier *mqclient.ini* (c'est-à-dire LDAP).

Une URL LDAP comprend au moins le protocole, le nom d'hôte, le numéro de port et le 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 *pExitUserAreaPtr* . L'échec de l'analyse syntaxique correcte de l'URL LDAP entraîne l'erreur MQCC_FAILED.

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 extraire les définitions de canal d'un serveur LDAP.

L'exit recherche sur le serveur LDAP les définitions de canal correspondant au filtre indiqué. Si le paramètre *QMgrName* contient un nom de gestionnaire de files d'attente spécifique, la recherche renvoie toutes les définitions de canal dont la valeur d'attribut LDAP *ibm-amqQueueManagerName* correspond au nom de gestionnaire de files d'attente indiqué.

Si le paramètre *QMgrName* est'*'ou' '(vide), la recherche renvoie toutes les définitions de canal dont 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. Au cours de cette opération, l'exit se déconnecte du serveur LDAP et libère toute la mémoire allouée et gérée par l'exit. Cela inclut la structure *MQNLDAPCTX* , le tableau de pointeurs et chaque MQCD qu'il référence. Toutes les autres zones sont définies sur les valeurs par défaut. Les paramètres d'exit *pQMgrName* et *ppConnectOpts* ne sont pas utilisés pendant *MQXR_TERM* et peuvent être NULL.

Schémas LDAP
Les données de connexion client sont stockées dans un référentiel global appelé annuaire LDAP (Lightweight Directory Access Protocol). Un client WebSphere MQ utilise un annuaire LDAP pour obtenir les définitions de connexion. La structure des définitions de connexion client WebSphere 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 WebSphere MQ normales sont décrites dans Attributs LDAP .

Attributs LDAP

Les attributs LDAP définis sont spécifiques à WebSphere MQ et sont mappés directement aux propriétés de connexion client.

WebSphere MQ Attributs de chaîne de répertoire de canal du client

Les attributs de chaîne de caractères avec leur mappage aux propriétés WebSphere 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 21. Attributs de chaîne de répertoire du canal du client WebSphere MQ

Attribut LDAP	Description	WebSphere 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	s a bThe LU 6.2 .	MODENAME
ibm-amqPassword	Mot de passe pouvant être utilisé.	MOT de passe
ibm-amqQueueManagerName	Nom du gestionnaire de files d'attente ou du groupe de gestionnaires de files d'attente auquel une application client WebSphere 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 SSL.	SSLCIPH
ibm-amqSslPeerName	Vérifie le nom distinctif (DN) du certificat provenant du gestionnaire de files d'attente ou du client homologue à l'autre extrémité d'un canal WebSphere 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

WebSphere 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 22. WebSphere MQ Attributs d'entier de répertoire de canal du client		
Attribut LDAP	**Description**	**WebSphere 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

WebSphere MQ Attribut booléen du canal client

Cet attribut booléen n'est mappé à aucune propriété WebSphere MQ . La syntaxe de cet attribut indique une valeur booléenne.

Tableau 23. WebSphere MQ Attribut booléen du canal client	
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.

WebSphere Attributs de liste de canaux client MQ

Les propriétés WebSphere 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 WebSphere MQ sont décrits dans le tableau suivant.

Tableau 24. WebSphere Attributs de liste de canaux client MQ		
Attribut LDAP	**Description**	**WebSphere 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

Tableau 24. WebSphere Attributs de liste de canaux client MQ (suite)		
Attribut LDAP	**Description**	**WebSphere MQ Propriété**
ibm-amqReceiveExitName	Nom du programme d'exit utilisateur à exécuter par l'exit utilisateur de réception de canal.	RCVEXIT

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 qui contient un ou plusieurs caractères génériques.

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.

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.

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.

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.

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.

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.

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 WebSphere 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.

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.

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.

ibm-amqSslCipherSpec
Cet attribut LDAP spécifie un CipherSpec unique pour une connexion SSL.

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.

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 WebSphere 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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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, les valeurs spécifiées pour **ibm-amqHeaderCompression** sont `0` , qui est mappé à `NONE`. Les valeurs qui dépassent la limite maximale autorisée seront ignorées par le client. Par exemple, la compression ibm-amqHeadercontient un maximum de 2 entiers dans la liste.

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.

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, ce qui correspond à la séquence de compression sous-jacente RLE, ZLIBFAST et ZLIBHIGH.

Toute valeur dépassant la limite maximale autorisée est ignorée par le client. Par exemple, la compression ibm-amqMessagecontient un maximum de 16 entiers dans la liste.

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.

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.

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.

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.

Ecriture d'une application de 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 :

Concepts associés

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Exemples de programmes WebSphere MQ», à la page 100
Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

«Ecriture d'applications client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Utilisation des services Web dans WebSphere MQ», à la page 977
Vous pouvez développer des applications IBM WebSphere MQ pour les services Web à l'aide du transport IBM WebSphere MQ pour SOAP ou du pont IBM WebSphere MQ pour HTTP.

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

«Traitement des erreurs de programme», à la page 569

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.

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

WebSphere MQ for Windows et WebSphere MQ sur les systèmes UNIX and Linux fournissent également:

- Appels via lesquels les programmes WebSphere MQ for Windows et WebSphere MQ sur les systèmes UNIX and Linux peuvent valider et rétablir 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 «Exemples de programmes pour les plateformes réparties», à la page 100.
- 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:

Concepts associés

Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

Appels MQI

Utilisez ces informations pour en savoir plus sur les appels dans MQI.

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. 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 WebSphere 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. 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:

Appels IBM WebSphere MQ sur les plateformes Windows, UNIX et Linux

Les produits suivants fournissent les appels MQCMIT et MQBACK:

- IBM WebSphere MQ pour Windows
- IBM WebSphere MQ sur les systèmes UNIX and Linux

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 WebSphere 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. A l'exception de WebSphere MQ for z/OS, cet appel est utilisé uniquement à partir d'un exit de conversion de données.

Voir MQXCNVC-Conversion de caractères pour la syntaxe utilisée avec l'appel MQXCNVC et «Ecriture des exits de conversion de données», à la page 432 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.

WebSphere MQ , définitions de données

Les fichiers de définition de données fournis avec WebSphere MQ contiennent:

- Définitions de toutes les constantes et codes retour WebSphere MQ
- Définitions des structures et types de données WebSphere 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 WebSphere MQ , voir «Fichiers de définition de données IBM WebSphere MQ», à la page 84.

structures

Les structures, utilisées avec les appels MQI répertoriés dans le «Appels MQI», à la page 204, sont fournies dans des fichiers de définition de données pour chacun des langages de programmation pris en charge.

Voir Récapitulatif des types de données de structure pour un récapitulatif des structures.

Programmes de remplacement IBM WebSphere 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 «Génération d'une application IBM WebSphere MQ», à la page 446. Pour plus d'informations sur la liaison à des fichiers de bibliothèque C ++, voir Utilisation de C++ *WebSphere MQ Utilisation de C++*.

IBM WebSphere MQ pour Windows

Sous IBM WebSphere 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 25. 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.lib	Prise en charge de C par le serveur TXSeries CICS (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib	Prise en charge de C par le client TXSeries CICS (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Exits des services installables pour C (32 bits)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Serveur pour 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)
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	Serveur pour IBM COBOL (64 bits)

Tableau 25. Fichiers de bibliothèque pour les applications Windows (suite)	
Fichier de bibliothèque	**Environnement**
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 WebSphere MQ est installé.

Utilisez `amqmdnet.dll` pour compiler les programmes .NET. Pour plus d'informations, voir «Compilation des programmes WebSphere MQ .NET», à la page 616 dans la section «Utilisation de .NET», à la page 581 .

Ces fichiers sont fournis à des fins de compatibilité avec les versions précédentes:

mqic32.lib
mqic32xa.lib

IBM WebSphere MQ pour AIX

Sous IBM WebSphere 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:

Tableau 26. Fichiers de bibliothèque pour les applications AIX sans unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm.a	Serveur pour C
libmqic.a & **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	WebSphere MQ pour la prise en charge de Micro Focus COBOL
libmqmcb.a	Serveur pour COBOL
libmqicb.a	Client pour COBOL

Tableau 26. Fichiers de bibliothèque pour les applications AIX sans unités d'exécution (suite)	
Fichier de bibliothèque	**Environnement**
libimqc23ia.a	Client pour C++
libimqs23ia.a	Serveur pour C++

Dans une application à unités d'exécution:

Tableau 27. Fichiers de bibliothèque pour les applications AIX à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm_r.a	Serveur pour C
libmqic_r.a & **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 pour C++
libimqs23ia_r.a	Serveur pour C++

IBM WebSphere MQ pour HP-UX

Sous IBM WebSphere MQ for HP-UX, 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.

Plateforme IA64 (IPF)

Dans une application sans unités d'exécution:

Tableau 28. Fichiers de bibliothèque pour les applications HP-UX non à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm.so	Serveur pour C
libmqic.so & **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
libimqi23ah.so	C++
libmqmcbrt.o	WebSphere MQ pour la prise en charge de Micro Focus COBOL
libmqmcb.so	Serveur pour COBOL
libmqicb.so	Client pour COBOL

Dans une application à unités d'exécution:

Tableau 29. Fichiers de bibliothèque pour les applications HP-UX à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm_r.so	Serveur pour C
libmqmzf_r.so & **libmqm_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
libimqi23ah_r.so	C++

IBM WebSphere MQ pour Linux

Sous IBM WebSphere 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:

Tableau 30. Fichiers de bibliothèque pour les applications Linux sans unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm.so	Serveur pour C
libmqic.so & **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:

Tableau 31. Fichiers de bibliothèque pour les applications Linux à unités d'exécution	
Fichier de bibliothèque	**Environnement**
libmqm_r.so	Serveur pour C
libmqic_r.so & **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

Tableau 31. Fichiers de bibliothèque pour les applications Linux à unités d'exécution (suite)	
Fichier de bibliothèque	**Environnement**
libimqc23gl_r.so	Client pour C++
libimqs23gl_r.so	Serveur pour C++

IBM WebSphere MQ Pour Solaris

Sous IBM WebSphere MQ for Solaris, 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 32. Fichiers de bibliothèque pour les applications Solaris	
Fichier de bibliothèque	**Environnement**
libmqm.so	Serveur et client pour C
libmqmzse.so	Pour C
libmqic.so	Client pour C
libmqmcs.so	Services communs 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
libimqc23as.a	Client pour C++
libimqs23as.a	Serveur pour C++

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 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 fonctionne avec un objet WebSphere 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:

- Codes anomalie pour toutes les autres plateformes WebSphere 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.

UNIX and LinuxRemarques

Remarques dont vous devez tenir compte.

Prenez note des points suivants lors du développement d'applications UNIX and Linux .

L'appel système fork dans les systèmes UNIX and Linux

Tenez compte des remarques suivantes lorsque vous utilisez un appel système fork dans des applications IBM WebSphere MQ .

Si votre application souhaite utiliser `fork`, le processus parent de cette application doit appeler `fork` avant d'effectuer des appels IBM WebSphere MQ , par exemple `MQCONN`, ou créer un objet IBM WebSphere MQ à l'aide de **`ImqQueueManager`**.

Si votre application souhaite créer un processus enfant après avoir effectué des appels IBM WebSphere 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 WebSphere MQ effectué dans le processus enfant renvoie MQRC_ENVIRONMENT_ERROR.

Traitement des signaux UNIX and Linux

Cela ne s'applique pas à WebSphere MQ for z/OS ou WebSphere MQ for Windows.

En général, les systèmes UNIX, Linux et IBM i 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.

Applications sans unités d'exécution
Non applicable sous Solaris car toutes les applications sont considérées comme des unités d'exécution même si elles n'utilisent qu'une seule unité d'exécution.

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 WebSphere MQ a géré 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.

Applications à unités d'exécution
Une unité d'exécution est considérée comme étant connectée à WebSphere MQ depuis MQCONN (ou MQCONNX) jusqu'à MQDISC.

Signaux synchrones

Des signaux synchrones apparaissent dans une unité d'exécution spécifique.

Les systèmes UNIX and Linux permettent en toute sécurité la configuration d'un gestionnaire de signaux pour ces signaux pour l'ensemble du processus. Toutefois, WebSphere MQ configure son propre gestionnaire pour les signaux suivants, dans le processus d'application, alors que toute unité d'exécution est connectée à WebSphere 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 WebSphere MQ configure ses propres gestionnaires de signaux synchrones, il sauvegarde tous les gestionnaires précédemment enregistrés pour chaque signal. Une fois que WebSphere MQ a traité l'un des signaux répertoriés, WebSphere MQ tente d'appeler

le gestionnaire de signaux qui était en vigueur lors de la première connexion WebSphere MQ au sein du processus. Les gestionnaires précédemment enregistrés sont restaurés lorsque toutes les unités d'exécution d'application se sont déconnectées de WebSphere MQ.

Etant donné que les gestionnaires de signaux sont sauvegardés et restaurés par WebSphere 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 WebSphere 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 WebSphere MQ, le gestionnaire de signaux de l'application doit appeler le gestionnaire WebSphere MQ correspondant pendant le 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 WebSphere MQ, le gestionnaire de signaux précédent doit être restauré avant que l'application ne se déconnecte de WebSphere MQ.
- Lorsqu'une application établit un gestionnaire de signaux avant de se connecter à WebSphere MQ, elle doit se déconnecter de WebSphere 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

WebSphere MQ n'utilise aucun signal asynchrone 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

WebSphere MQ gère les signaux suivants lors de l'entrée-sortie d'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)

Autres considérations
Tenez compte des remarques suivantes lorsque vous utilisez le traitement des signaux UNIX .

Applications Fastpath (dignes de confiance)

Les applications Fastpath s'exécutent dans le même processus que WebSphere MQ et sont donc exécutées dans l'environnement à unités d'exécution multiples.

Dans cet environnement, WebSphere 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 WebSphere 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 217.

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 (), WebSphere 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.

Dans l'environnement AIX , WebSphere MQ requiert le redémarrage des appels système interrompus par des signaux. 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, WebSphere 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 WebSphere MQ dans un environnement à unités d'exécution multiples ont les mêmes restrictions que pour les applications à chemins d'accès rapide. Considérez qu'ils sont connectés de manière permanente à WebSphere MQ et qu'ils n'utilisent donc pas de signaux ou d'appels de système d'exploitation non autorisant les unités d'exécution multiples.

Gestionnaires d'exit VMS

Les utilisateurs peuvent installer des gestionnaires d'exit pour une application WebSphere MQ à l'aide du service système **SYS$DCLEXH** .

Le gestionnaire d'exit reçoit le contrôle à la sortie d'une image. Un exit d'image se produit normalement lorsque vous appelez le service Exit ($EXIT) ou Forcer l'exit ($FORCEX). $FORCEX interrompt le processus cible en mode utilisateur. Ensuite, tous les gestionnaires d'exit en mode utilisateur (établis par $DCLEXH) commencent à s'exécuter dans l'ordre inverse de l'établissement. Pour plus de détails sur les gestionnaires d'exit et $FORCEX, voir le document *VMS Programming Concepts Manual* et le document *VMS System Services Manual*.

Si vous appelez une fonction MQI à partir d'un gestionnaire d'exit, le comportement de la fonction dépend de la manière dont l'image a été arrêtée. Si l'image a été arrêtée alors qu'une autre fonction MQI est active, un `MQRC_CALL_IN_PROGRESS` est renvoyé.

Il est possible d'appeler une fonction MQI à partir d'un gestionnaire d'exit si aucune autre fonction MQI n'est active et que les appels sont désactivés pour l'application WebSphere MQ . Si des appels de mise à jour sont activés pour l'application WebSphere MQ , ils échouent avec le code anomalie `MQRC_HCONN_ERROR`.

La portée d'un appel MQCONN ou MQCONNX est généralement l'unité d'exécution qui l'a émis. Si les appels de mise à jour sont activés, le gestionnaire d'exit s'exécute en tant qu'unité d'exécution distincte et les descripteurs de connexion ne peuvent pas être partagés.

Les gestionnaires d'exit sont démarrés dans le contexte interrompu du processus cible. Il incombe à l'application de s'assurer que les actions effectuées par un gestionnaire sont sûres et fiables, pour le contexte interrompu de manière asynchrone à partir duquel elles sont appelées.

Connexion et déconnexion d'un gestionnaire de files d'attente

Pour utiliser les services de programmation WebSphere 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:

z/OS par lots, WebSphere MQ for IBM i, WebSphere MQ sur les systèmes UNIX , WebSphere MQ sur les systèmes Linux et WebSphere MQ for Windows

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 Les programmes batch peuvent se connecter, consécutivement ou simultanément, à plusieurs gestionnaires de files d'attente sur le même bloc de contrôle des tâches.

IMS

La région de contrôle IMS est connectée à un ou plusieurs gestionnaires de files d'attente au démarrage. Cette connexion est contrôlée par les commandes IMS . Toutefois, les programmes d'écriture de la mise en file d'attente de messages IMS doivent utiliser l'appel MQCONN MQI 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.

Les programmes IMS peuvent se connecter, consécutivement ou simultanément, à plusieurs gestionnaires de files d'attente sur le même bloc de contrôle des tâches.

CICS Transaction Server for z/OS et CICS for MVS/ESA

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, qui estfourni avec WebSphere MQ for z/OS.

Les tâches CICS peuvent se connecter uniquement au gestionnaire de files d'attente auquel la région CICS , elle-même, est connectée.

Remarque : Les programmes CICS peuvent également utiliser les appels de connexion et de déconnexion MQI (MQCONN et MQDISC). Vous pouvez effectuer cette opération afin de pouvoir porter ces applications dans des environnements non CICS 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 pour 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 216
- «Connexion à un gestionnaire de files d'attente à l'aide de l'appel MQCONNX», à la page 217
- «Déconnexion de programmes d'un gestionnaire de files d'attente à l'aide de MQDISC», à la page 222

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Ouverture et fermeture d'objets», à la page 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
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 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

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:

- Pour IBM WebSphere 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.
- Pour les systèmes IBM WebSphere MQ for Windows, IBM i, UNIXet Linux , le gestionnaire de files d'attente par défaut est spécifié dans le fichier mqs.ini .

Sinon, dans les environnements z/OS MVS batch, 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 WebSphere MQ .

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 WebSphere MQ for z/OS est installé.

Avec l'environnement TXSeries CICS et TXSeries for 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 Sécurité.

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 224).

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 WebSphere 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 WebSphere MQ peut être établie à partir d'une unité d'exécution à la fois. Vous pouvez également envisager d'utiliser «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 221 pour autoriser l'utilisation de plusieurs connexions WebSphere MQ à partir d'une seule unité d'exécution et d'une connexion WebSphere MQ à partir de n'importe quelle unité d'exécution.[1]

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 tant qu'entrée de MQCONNX, vous pouvez fournir un nom de gestionnaire de files d'attente ou un nom de groupe de partage de files d'attente sur les systèmes de files d'attente partagées z/OS . La sortie de MQCONNX est la suivante:

- Un descripteur de connexion (Hconn)
- Un code achèvement
- Un code anomalie

Vous utilisez le descripteur de connexion lors des appels MQI suivants.

Une description de tous les paramètres de MQCONNX est fournie dans MQCONNX. La zone *Options* permet de définir STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING ou ISOLATED_BINDING pour toute version de MQCNO. Vous pouvez également établir des connexions partagées (indépendantes de l'unité d'exécution) à l'aide d'un appel MQCONNX. Pour plus d'informations à ce sujet, voir «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 221 .

[1] Lorsque vous utilisez des applications à unités d'exécution multiples avec IBM WebSphere MQ sur des systèmes UNIX and 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).

MQCNO_STANDARD_BINDING

Par défaut, MQCONNX (comme MQCONN) implique deux unités d'exécution logiques dans lesquelles l'application WebSphere MQ et l'agent du gestionnaire de files d'attente local s'exécutent dans des processus distincts. L'application WebSphere MQ demande l'opération WebSphere MQ et l'agent de gestionnaire de files d'attente local la demande. Elle est définie par l'option MQCNO_STANDARD_BINDING de l'appel MQCONNX.

Si vous spécifiez MQCNO_STANDARD_BINDING, l'appel MQCONNX utilise MQCNO_SHARED_BINDING ou MQCNO_ISOLATED_BINDING, en fonction de la valeur de l'attribut de type DefaultBinddu gestionnaire de files d'attente, qui est défini dans qm.ini ou dans le registre Windows .

Il s'agit de la valeur par défaut.

Si vous établissez une liaison à la bibliothèque `mqm` , une connexion serveur standard utilisant le type de liaison par défaut est tentée en premier. Si le chargement de la bibliothèque du serveur sous-jacent a échoué, une connexion client est tentée à la place.

- Si la variable d'environnement MQ_CONNECT_TYPE est spécifiée, l'une des options suivantes peut être fournie pour modifier le comportement de MQCONN ou MQCONNX si MQCNO_STANDARD_BINDING est spécifié. (sauf si MQCNO_FASTPATH_BINDING est spécifié avec MQ_CONNECT_TYPE défini sur LOCAL ou STANDARD pour permettre aux connexions fastpath d'être rétromigrées par l'administrateur sans modification associée de l'application:

Valeur	**Explication**
client	Une connexion client uniquement est tentée.
Fastpath	Cette valeur était prise en charge dans les éditions précédentes, mais elle est désormais ignorée si elle est spécifiée.
LOCAL	Une connexion serveur uniquement est tentée. Les connexions Fastpath sont rétromigrées vers une connexion serveur standard.
Standard	Prise en charge pour la compatibilité avec les éditions précédentes. Cette valeur est désormais traitée comme étant LOCAL.

- Si la variable d'environnement MQ_CONNECT_TYPE n'est pas définie lorsque MQCONN est appelé, une connexion serveur standard utilisant le type de liaison par défaut est tentée. Si le chargement de la bibliothèque du serveur échoue, une connexion client est tentée.

MQCNO_FASTPATH_BINDING

Les *applications sécurisées* impliquent que l'application WebSphere 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. Elle est définie par l'option MQCNO_FASTPATH_BINDING de l'appel MQCONNX.

Vous devez lier des applications sécurisées aux bibliothèques WebSphere MQ à unités d'exécution. Pour savoir comment configurer une application WebSphere MQ pour qu'elle s'exécute en tant qu'application sécurisée, voir Options MQCNO.

Cette option offre les performances les plus élevées.

Remarque : Cette option compromet l'intégrité du gestionnaire de files d'attente: il n'existe aucune protection contre l'écrasement de son stockage. Cela s'applique également si l'application contient des erreurs qui peuvent être exposées aux messages et à d'autres données du gestionnaire de files d'attente. Tenez compte de ces problèmes avant d'utiliser cette option.

MQCNO_LIEN_PARTAGE

Spécifiez cette option pour que l'application et l'agent du gestionnaire de files d'attente local s'exécutent dans des processus distincts. Cela maintient l'intégrité du gestionnaire de files d'attente, c'est-à-dire qu'il protège le gestionnaire de files d'attente des programmes errants. Toutefois, l'application et l'agent de gestionnaire de files d'attente locales partagent certaines ressources.

Cette option est intermédiaire entre MQCNO_FASTPATH_BINDING et MQCNO_ISOLATED_BINDING, tant en termes de protection de l'intégrité du gestionnaire de files d'attente qu'en termes de performances des appels MQI.

MQCNO_SHARED_BINDING est ignoré si le gestionnaire de files d'attente ne prend pas en charge ce type de liaison. Le traitement se poursuit comme si l'option n'avait pas été spécifiée.

Si une application s'est connectée au gestionnaire de files d'attente local à l'aide de MQCNO_SHARED_BINDING, le gestionnaire de files d'attente peut être arrêté alors que l'application est en cours d'exécution. Si vous redémarrez le gestionnaire de files d'attente alors que l'application est toujours en cours d'exécution, la tentative de démarrage du gestionnaire de files d'attente échoue avec l'erreur AMQ7018 car l'application est toujours en attente des ressources requises par le gestionnaire de files d'attente.

Pour démarrer le gestionnaire de files d'attente, vous devez arrêter l'application.

MQCNO_LIEN_ISOLÉ_LIAISON

Spécifiez cette option pour que l'application et l'agent du gestionnaire de files d'attente local s'exécutent dans des processus distincts, comme pour MQCNO_SHARED_BINDING. Dans ce cas, cependant, le processus d'application et l'agent de gestionnaire de files d'attente locales sont isolés les uns des autres car ils ne partagent pas de ressources.

Il s'agit de l'option la plus sûre pour protéger l'intégrité du gestionnaire de files d'attente, mais elle offre les performances les plus lentes des appels MQI.

MQCNO_ISOLATED_BINDING est ignoré si le gestionnaire de files d'attente ne prend pas en charge ce type de liaison. Le traitement se poursuit comme si l'option n'avait pas été spécifiée.

LIEN_CLIENT_MQCNO_BINDING

Spécifiez cette option pour que l'application tente une connexion client uniquement. Cette option présente les limitations suivantes:

- MQCNO_CLIENT_BINDING est rejeté sur z/OS avec MQRC_OPTIONS_ERROR.
- MQCNO_CLIENT_BINDING est rejeté avec MQRC_OPTIONS_ERROR s'il est spécifié avec une option de liaison MQCNO autre que MQCNO_STANDARD_BINDING.
- MQCNO_CLIENT_BINDING n'est pas disponible pour Java car il possède ses propres mécanismes pour choisir le type de liaison.
- V 7.5.0.7 Avant IBM WebSphere MQ Version 7.5.0, groupe de correctifs 7, MQCNO_CLIENT_BINDING n'était pas disponible pour .NET car il disposait de ses propres mécanismes pour choisir le type de liaison. Depuis la Version 7.5.0, Fix Pack 7, la restriction relative à l'utilisation de .NET pour MQCNO_CLIENT_BINDING est supprimée.
- Si la variable d'environnement MQ_CONNECT_TYPE n'est pas définie lorsque MQCONNX est appelé, une connexion serveur standard utilisant le type de liaison par défaut est tentée. Si le chargement de la bibliothèque du serveur échoue, une connexion client est tentée.

MQCNO_LIEN_LOCAL_LOCAL

Indiquez cette option pour que l'application tente de se connecter au serveur. Si MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING ou MQCNO_SHARED_BINDING est également spécifié, la connexion est de ce type et est documentée dans cette section.
Sinon, une connexion serveur standard est tentée à l'aide du type de liaison par défaut.
MQCNO_LOCAL_BINDING présente les limitations suivantes:

- MQCNO_LOCAL_BINDING est ignoré sur z/OS.

- MQCNO_LOCAL_BINDING est rejeté avec MQRC_OPTIONS_ERROR s'il est spécifié avec une option de reconnexion MQCNO autre que MQCNO_RECONNECT_AS_DEF.
- MQCNO_LOCAL_BINDING n'est pas disponible pour Java car il possède ses propres mécanismes de sélection du type de liaison.
- V 7.5.0.7 Avant IBM WebSphere MQ Version 7.5.0, groupe de correctifs 7, MQCNO_LOCAL_BINDING n'était pas disponible pour .NET car il disposait de ses propres mécanismes pour choisir le type de liaison. Depuis la Version 7.5.0, Fix Pack 7, la restriction relative à l'utilisation de .NET pour MQCNO_LOCAL_BINDING est supprimée.
- Si la variable d'environnement MQ_CONNECT_TYPE n'est pas définie lorsque MQCONNX est appelé, une connexion serveur standard utilisant le type de liaison par défaut est tentée. Si le chargement de la bibliothèque du serveur échoue, une connexion client est tentée.

Sous z/OS , ces options sont tolérées, mais seule une connexion liée standard est effectuée. MQCNO version 3, pour z/OS, offre quatre options alternatives:

MQCNO_SERIALIZE_CONN_TAG_QSG

Cela permet à une application de demander qu'une seule instance d'une application s'exécute à la fois dans un groupe de partage de files d'attente. Pour ce faire, vous devez enregistrer l'utilisation d'une balise de connexion avec une valeur spécifiée ou dérivée par l'application. La balise est une chaîne de caractères de 128 octets spécifiée dans la version 3 de MQCNO.

MQCNO_RESTRICT_CONN_TAG_QSG

Cette option est utilisée lorsqu'une application se compose de plusieurs processus (ou d'un bloc de contrôle des tâches), chacun d'eux pouvant se connecter à un gestionnaire de files d'attente. La connexion est autorisée uniquement s'il n'y a pas d'utilisation actuelle de la balise ou si l'application à l'origine de la demande se trouve dans la même portée de traitement. Il s'agit de l'espace adresse MVS dans le même groupe de partage de files d'attente que le propriétaire de l'étiquette.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Similaire à MQCNO_SERIALIZE_CONN_TAG_QSG, mais seul le gestionnaire de files d'attente local est interrogé pour déterminer si la balise demandée est déjà utilisée.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Cette opération est similaire à MQCNO_RESTRICT_CONN_TAG_QSG, mais seul le gestionnaire de files d'attente local est interrogé pour déterminer si la balise demandée est déjà utilisée.

Restrictions pour les applications sécurisées

Les restrictions suivantes s'appliquent aux applications sécurisées:

- 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.
- Sur les systèmes WebSphere MQ on UNIX 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 l'appel MQI suivant.
- Sous WebSphere MQ for HP-UX, les applications Fast Path à unités d'exécution multiples devront probablement définir une taille de pile supérieure à la taille par défaut. Utilisez une taille de 256 Ko.

- Sous WebSphere MQ for Windows , les applications 64 bits sécurisées ne sont pas prises en charge. Si vous essayez d'exécuter une application 64 bits sécurisée, elle sera rétromirée vers une connexion liée standard.
- Sous WebSphere MQ sur les systèmes UNIX and Linux , 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.

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 : Non pris en charge sur WebSphere MQ for z/OS.

Sur les plateformes WebSphere MQ autres que WebSphere MQ for z/OS, 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 221, l'unité d'exécution 1 émet une demande d'obtention qui attend les messages qui ne sont peut-être 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.

Utilisation des options d'appel MQCONNX avec MQ_CONNECT_TYPE

Utilisez ces informations pour comprendre les différentes options d'appel MQCONNX comment elles sont utilisées avec MQ_CONNECT_TYPE.

Sur les systèmes WebSphere MQ for IBM i, WebSphere MQ for Windowset WebSphere MQ sur les systèmes UNIX and Linux , 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 33. 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.

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:

- 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.
- Sous WebSphere MQ for IBM i où, 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.

A l'exception de CICS sur z/OS, une fois que MQDISC a été appelé, le descripteur de connexion (Hconn) n'est plus valide et vous ne pouvez pas émettre d'autres appels MQI tant que vous n'avez pas appelé 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.

Si vous utilisez MQCONNX pour vous connecter à WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ .

Pour effectuer l'une des opérations suivantes, vous devez d'abord *ouvrir* l'objet WebSphere 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 244).

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 215.

Il existe quatre types d'objet WebSphere 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 WebSphere MQ , voir Objets.

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 245.

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 à la suite d'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 224
- «Création de files d'attente dynamiques», à la page 233
- «Ouverture des files d'attente distantes», à la page 234
- «Fermeture d'objets à l'aide de l'appel MQCLOSE», à la page 234

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 235
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 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

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 sous 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 programmes, 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 les «Portée de MQCONN ou MQCONNX», à la page 217 et «Connexions partagées (indépendantes de l'unité d'exécution) avec MQCONNX», à la page 221. Toutefois, des considérations supplémentaires sont à prendre en compte dans certains environnements:

CICS
Dans un programme CICS , vous ne pouvez utiliser le descripteur que dans la même tâche CICS à partir de laquelle vous avez effectué l'appel MQOPEN.

IMS et z/OS par lots
Dans les environnements IMS et de traitement 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 247 sous «Listes de diffusion», à la page 245.

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 WebSphere MQ , l'appel MQOPEN exécute une fonction de résolution de nom sur le nom de file d'attente que vous spécifiez. 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. Pour une présentation du processus de résolution de nom, voir Tableau 34, à la page 226 . Le nom que vous fournissez dans *ObjectQMgrName* est résolu *avant* dans *ObjectName*.

La Tableau 34, à la page 226 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 34. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN

Entrée dans MQOD		**Noms résolus**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**File d'attente de transmission**
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 34. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)

Entrée dans MQOD		**Noms résolus**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**File d'attente de transmission**
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 34. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)

Entrée dans MQOD		**Noms résolus**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**File d'attente de transmission**
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 34. Résolution des noms de file d'attente lors de l'utilisation de MQOPEN (suite)				
Entrée dans MQOD		**Noms résolus**		
ObjectQMgrName	***ObjectName***	***ObjectQMgrName***	***ObjectName***	**File d'attente de transmission**
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. Lorsque vous utilisez des gestionnaires de files d'attente WebSphere MQ for z/OS qui font 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 34, à la page 226.

   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 ObjectQMgrindiqué, dans la file d'attente intra-groupe ou dans un canal WebSphere 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 locaux et éloignés 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 362.

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 283.

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 245.
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 supprimésmessages 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* etAttributs *DefInputOpenOption* . Pour accepter cet accès, utilisez l'option MQOO_INPUT_AS_Q_DEF. Voir Tableau 35, à la page 231 pour voir comment la définition de ces attributs affecte le type d'accès qui vous sera accordé lorsque vous utiliserez cette option.

Tableau 35. 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**	**PARTAGE**	**EXCLUSIVE**
Partageable	PARTAGE	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 35, à la page 231 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 245.

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 332).

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 39
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», à la page 242
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.

Option MQOPEN pour la mise au repos du gestionnaire de files d'attente
Dans l'environnement CICS sous z/OS, si vous utilisez l'appel MQOPEN lorsque le gestionnaire de files d'attente est à l'état de mise au repos, l'appel échoue toujours.

Dans les autres environnements z/OS , IBM i, Windows et UNIX and Linux , l'appel échoue lorsque le gestionnaire de files d'attente est mis au repos uniquement si vous utilisez l'option MQOO_FAIL_IF_QUIESCING de l'appel MQOPEN.

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 237).

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 WebSphere 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 le Tableau 34, à la page 226.

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.
- Sous WebSphere MQ for z/OS, si des demandes MQGET avec une option MQGMO_SET_SIGNAL sont 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, ou bien, pour les applications CICS sur z/OS, vous pouvez 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 236
- «Insertion de messages dans une file d'attente éloignée», à la page 241
- «Définition des propriétés d'un message», à la page 241
- «Contrôle des informations de contexte», à la page 242
- «Insertion d'un message dans une file d'attente à l'aide de l'appel MQPUT1», à la page 244
- «Listes de diffusion», à la page 245
- «Certains cas où les appels d'insertion échouent», à la page 250

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Obtention de messages à partir d'une file d'attente», à la page 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

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 236
- «Définition de messages à l'aide de la structure MQMD», à la page 237
- «Spécification d'options à l'aide de la structure MQPMO», à la page 237
- «Les données de votre message», à la page 240
- «Insertion de messages: Utilisation de descripteurs de message», à la page 241

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 WebSphere MQ», à la page 10et 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 245). 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.

En outre, le mode de fonctionnement de l'appel avec les points de synchronisation est déterminé par la plateforme. La valeur par défaut du contrôle de point de synchronisation est yes dans z/OS; pour les autres plateformes, elle est no.

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 39. 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», à la page 242.

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 245). 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 249.

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 249 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 248 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 WebSphere 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 WebSphere MQ à 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.

WebSphere MQ ajoute des informations d'en-tête aux messages dans les cas suivants:

- Lorsque vous placez un message dans une file d'attente éloignée, WebSphere 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 WebSphere 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, WebSphere 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, WebSphere 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.

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 234. 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 WebSphere MQ distributed-messaging techniques .

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 WebSphere MQ Version 7.0, qui ne peut pas traiter les propriétés de message, vous pouvez définir le paramètre PropertyControl dans la définition de canal pour spécifier le mode de traitement des propriétés.

Contrôle des informations de contexte

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.

Pour contrôler les informations de contexte, utilisez la zone *Options* dans la structure MQPMO.

Si vous ne le faites pas, le gestionnaire de files d'attente écrase les informations contextuelles qui se trouvent peut-être déjà dans le descripteur de message avec 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 WebSphere 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 WebSphere 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.

Si vous ne souhaitez pas d'informations de contexte associées à votre message, utilisez l'option MQPMO_NO_CONTEXT.

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 242
- «Transmission du contexte utilisateur», à la page 243
- «Transmission de tous les contextes», à la page 243
- «Définition du contexte d'identité», à la page 243
- «Définition du contexte utilisateur», à la page 243
- «Définition de tous les contextes», à la page 243

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 39
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 232
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 225). 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 237).

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.

Listes de diffusion

Non pris en charge sur WebSphere MQ for z/OS. Les listes de distribution vous 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.

V 7.5.0.8

Avertissement : Les listes de distribution ne prennent pas en charge les files d'attente alias qui pointent vers des objets de rubrique. Depuis la Version 7.5.0, Fix Pack 8, si une file d'attente alias pointe vers un objet de rubrique dans une liste de distribution, IBM WebSphere 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 29, à la page 246 montre le fonctionnement des listes de distribution.

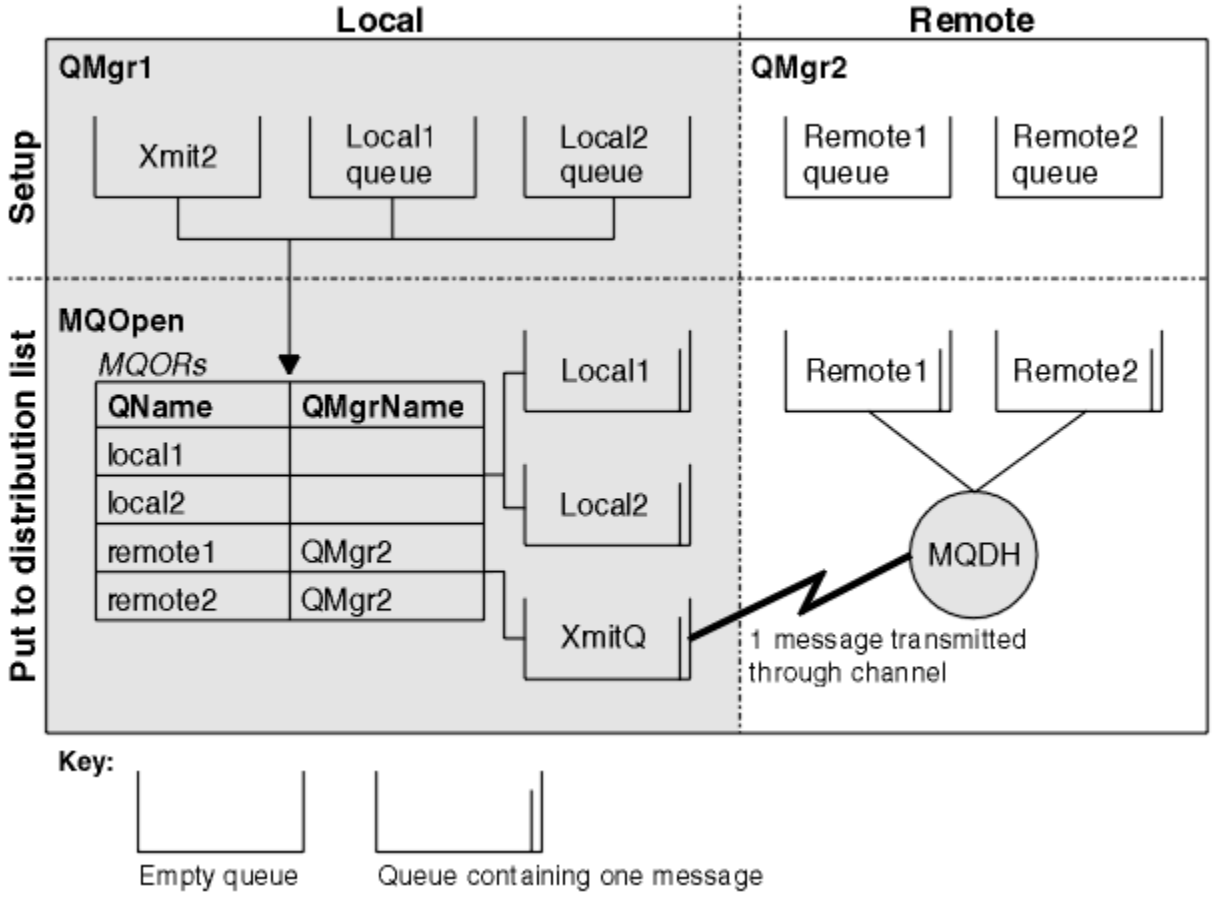


Figure 29. Fonctionnement des listes de distribution

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 235 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 30, à la page 248 montre comment ouvrir une liste de distribution en C.

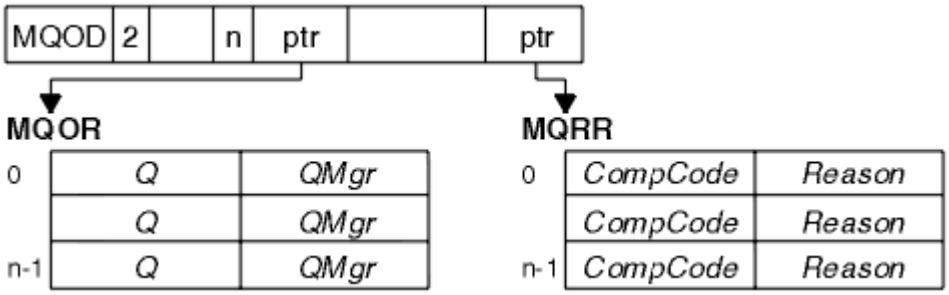


Figure 30. Ouverture d'une liste de distribution en C

La Figure 31, à la page 248 montre comment ouvrir une liste de distribution en COBOL.

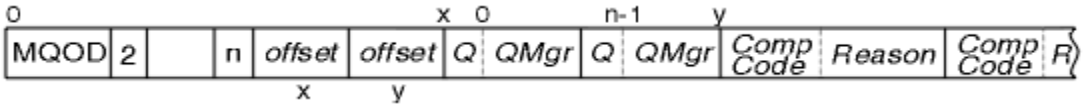


Figure 31. Ouverture d'une liste de distribution en COBOL

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 223 .

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 235 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 237 .
- 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 133 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 32, à la page 249 montre comment placer un message dans une liste de distribution dans C.

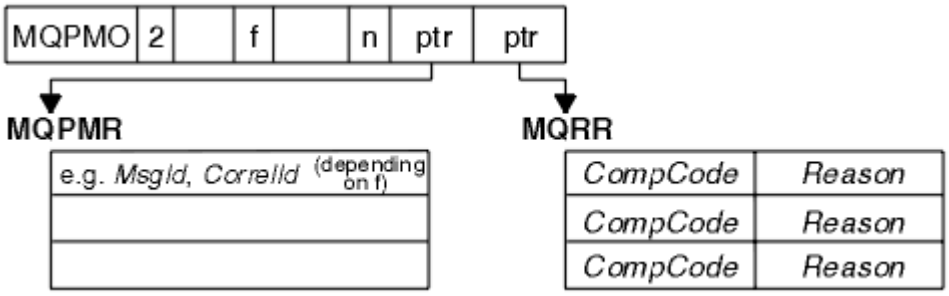


Figure 32. Insertion d'un message dans une liste de distribution en C

La Figure 33, à la page 250 montre comment placer un message dans une liste de distribution en COBOL.

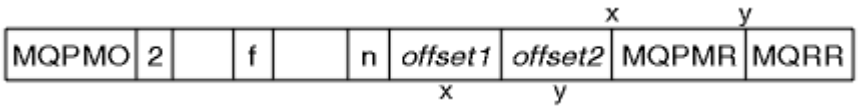


Figure 33. Insertion d'un message dans une liste de distribution en COBOL

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 215 et «Ouverture et fermeture d'objets», à la page 223.

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:

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
Utilisez ces informations pour savoir comment placer des messages dans une file d'attente.

«Interrogation et définition des attributs d'objet», à la page 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

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 252
- «Description des messages à l'aide de la structure MQMD et de l'appel MQGET», à la page 252
- «Spécification des options MQGET à l'aide de la structure MQGMO», à la page 253
- «Spécification de la taille de la zone tampon», à la page 255

Spécification des descripteurs de connexion

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 WebSphere MQ», à la page 10et 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 268.

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 256 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 279)
- 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
- Sous WebSphere MQ for z/OS, indique si le message extrait est marqué comme ignoré lors de l'annulation (voir «Annulation ignorée», à la page 280)
- 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
- Sous WebSphere MQ for z/OS, indiquez si l'appel doit être effectué. 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
- Sous WebSphere 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 282)
- Ordre dans lequel les messages et les segments (à l'exception de WebSphere MQ for z/OS) sont extraits d'une file d'attente
- A l'exception de WebSphere MQ for z/OS, indique si les messages logiques complets uniquement peuvent être extraits
- Indique si les messages d'un groupe peuvent être extraits uniquement lorsque *tous* les messages du groupe sont disponibles
- A l'exception de WebSphere MQ for z/OS, indique si les segments d'un message logique peuvent être extraits uniquement lorsque *tous* les segments du message logique sont disponibles

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. De plus, dans WebSphere 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 UNIX and Linux	Non
z/OS	Oui
Systèmes Windows	Non

- Sous WebSphere 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.

Sous WebSphere 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 sur ces options, voir «En attente de messages», à la page 279.

Signal1
Signal1 est pris en charge sur WebSphere MQ for z/OS et MQSeries for HP Integrity NonStop Server uniquement.

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 WebSphere MQ sur toutes les autres plateformes, la zone *Signal1* est réservée et sa valeur n'est pas significative.

Signal2
La zone *Signal2* est réservée sur toutes les plateformes et sa valeur n'est pas significative.

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 274.

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 18). 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 268.

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.

Ordre logique et physique

Les messages des files d'attente peuvent apparaître (dans chaque niveau de priorité) 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 36. 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 34, à la page 257:

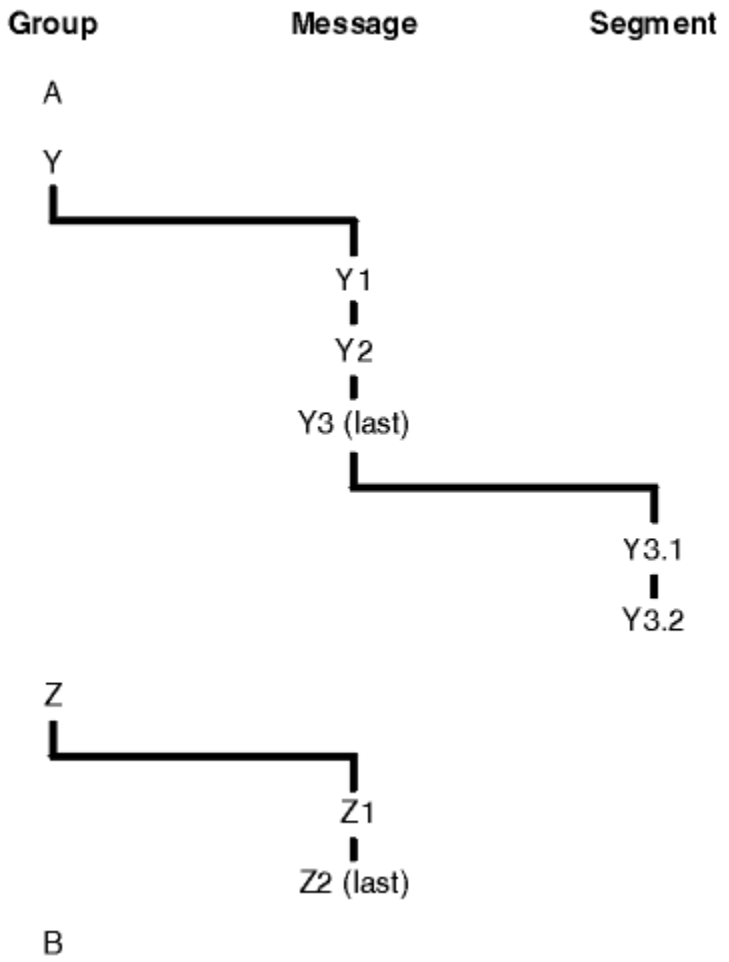


Figure 34. Ordre logique dans une file d'attente

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 sont arrivés à des heures similaires et que le message 2 du groupe Z a dépassé le message 1 du même groupe, l'ordre physique ressemblerait à la figure Figure 35, à la page 258:

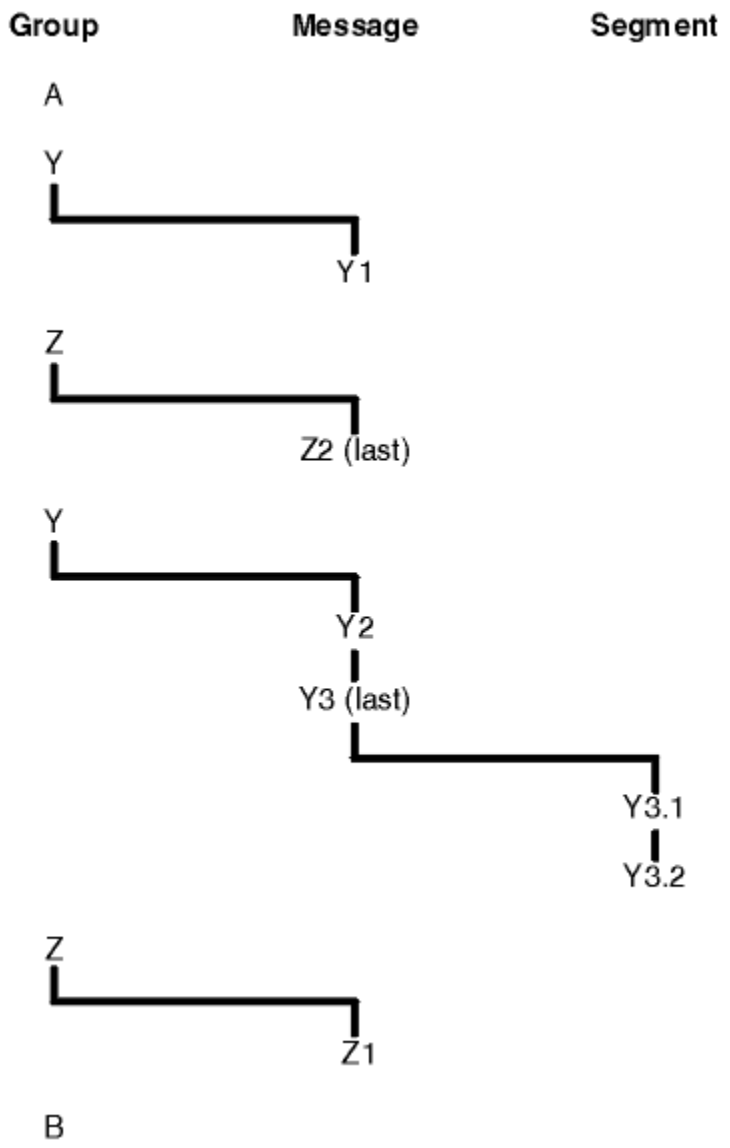


Figure 35. Ordre physique dans une file d'attente

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 : Sous IBM WebSphere 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 WebSphere MQ incrémente automatiquement le numéro de séquence de message.
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 35, à la page 258 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 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 36. Options MQPUT relatives aux messages dans les groupes et les segments de messages logiques

Options que vous spécifiez				**Statut du groupe et du message de journal avant l'appel**		**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

Tableau 36. Options MQPUT relatives aux messages dans les groupes et les segments de messages logiques (suite)

Options que vous spécifiez				**Statut du groupe et du message de journal avant l'appel**		**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	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, autre valeur dans la zone	1	0
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, autre valeur dans la zone	1	Valeur dans la zone

Tableau 36. Options MQPUT relatives aux messages dans les groupes et les segments de messages logiques (suite)

Options que vous spécifiez				Statut du groupe et du message de journal avant l'appel		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***
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, autre valeur 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, autre valeur 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. Tableau 37, à la page 263 montre les différents cas qui peuvent se présenter. 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 37. 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
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 36.

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 d'autres exemples de regroupement de messages, voir «Segmentation d'application des messages logiques», à la page 276 et «Mise en place et obtention d'un groupe couvrant des unités de travail», à la page 265.

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 38, à la page 268 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 38. 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 39, à la page 268 indique quel message est extrait pour les paramètres possibles de ces zones.

Tableau 39. 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

Tableau 39. Utilisation de l'identificateur de groupe (suite)

Pour extraire ...	**options de mise en correspondance**

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*, *MsgSeqNumber*et *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 à ce sujet, voir «Ordre logique et physique», à la page 256 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.

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 23.

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 WebSphere MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- La version du client et la version du gestionnaire de files d'attente éloignées doivent être la version 7 ou une version ultérieure de WebSphere MQ.
- L'application client doit être compilée et liée dans les bibliothèques client WebSphere MQ MQI à 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 WebSphere 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 sur AIX», à la page 272
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 272
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 270
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 WebSphere MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- La version du client et la version du gestionnaire de files d'attente éloignées doivent être la version 7 ou une version ultérieure de WebSphere MQ.
- L'application client doit être compilée et liée dans les bibliothèques client WebSphere MQ MQI à 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 40. Options MQGET et lecture anticipée

	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 WebSphere MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- La version du client et la version du gestionnaire de files d'attente éloignées doivent être la version 7 ou une version ultérieure de WebSphere MQ.
- L'application client doit être compilée et liée dans les bibliothèques client WebSphere MQ MQI à 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 sur 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, afin d'obtenir 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 WebSphere MQ `MQRC_UNEXPECTED_ERROR (2195)` et, s'il y a suffisamment de descripteurs de fichier, un fichier IBM WebSphere 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
```

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, WebSphere MQ fournit un certain nombre de méthodes de traitement des messages dont la taille est supérieure à 4 Mo.

Vous pouvez augmenter l'attribut *MaxMsgLength* jusqu'à 100 Mo sur tous les systèmes WebSphere MQ à la version V6 ou ultérieure. Définissez cette valeur pour refléter la taille des messages utilisant la file d'attente. Sur les systèmes WebSphere MQ autres que WebSphere MQ for z/OS, 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 maximale de message *par défaut* prise en charge dépend de l'environnement dans lequel vous travaillez.

Si vous traitez des messages volumineux, vous pouvez modifier ces attributs indépendamment. Vous pouvez définir la valeur de l'attribut de gestionnaire de files d'attente entre 32768 octets et 100 Mo ; vous pouvez définir la valeur de l'attribut de file d'attente entre 0 et 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.

Segmentation des messages

Utilisez ces informations pour en savoir plus sur la segmentation des messages.

Remarque : Non pris en charge dans IBM WebSphere MQ for z/OS ou par des applications utilisant des classes IBM WebSphere MQ pour JMS.

L'augmentation de la longueur maximale des messages, comme expliqué dans la rubrique «Augmentation de la longueur maximale des messages», à la page 274 , 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 36.

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).

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 265.

Toutefois, vous ne pouvez pas insérer ou obtenir des messages segmentés dans une unité d'oeuvre globale.

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
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.

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 à elle 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
```

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.

Messages de référence

Utilisez ces informations pour en savoir plus sur les messages de référence.

Remarque : Non pris en charge dans WebSphere MQ for z/OS.

Cette méthode permet de transférer un objet LOB d'un noeud à un autre sans stocker l'objet dans des files d'attente WebSphere MQ au niveau des 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 ce faire, vous spécifiez 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 428.

WebSphere 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 WebSphere 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 être acheminé via plusieurs gestionnaires de files d'attente avant d'atteindre sa destination, les données non formatées *sont* présentes dans les files d'attente WebSphere 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 WebSphere 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 WebSphere 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 «Exemples de programmes pour les plateformes réparties», à la page 100.

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 spécifierDurée 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.
- 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.

Pour plus d'informations sur les circonstances dans lesquelles ces actions se produisent, voir MQGMO.

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.

Remarque : Pris en charge uniquement sur WebSphere MQ for z/OS.

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 WebSphere 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 mémorisées DB2 participant 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 36, à la page 281 illustre une séquence typique d'étapes qu'un programme d'application peut contenir lorsqu'une demande MQGET est requise pour ignorer l'annulation.

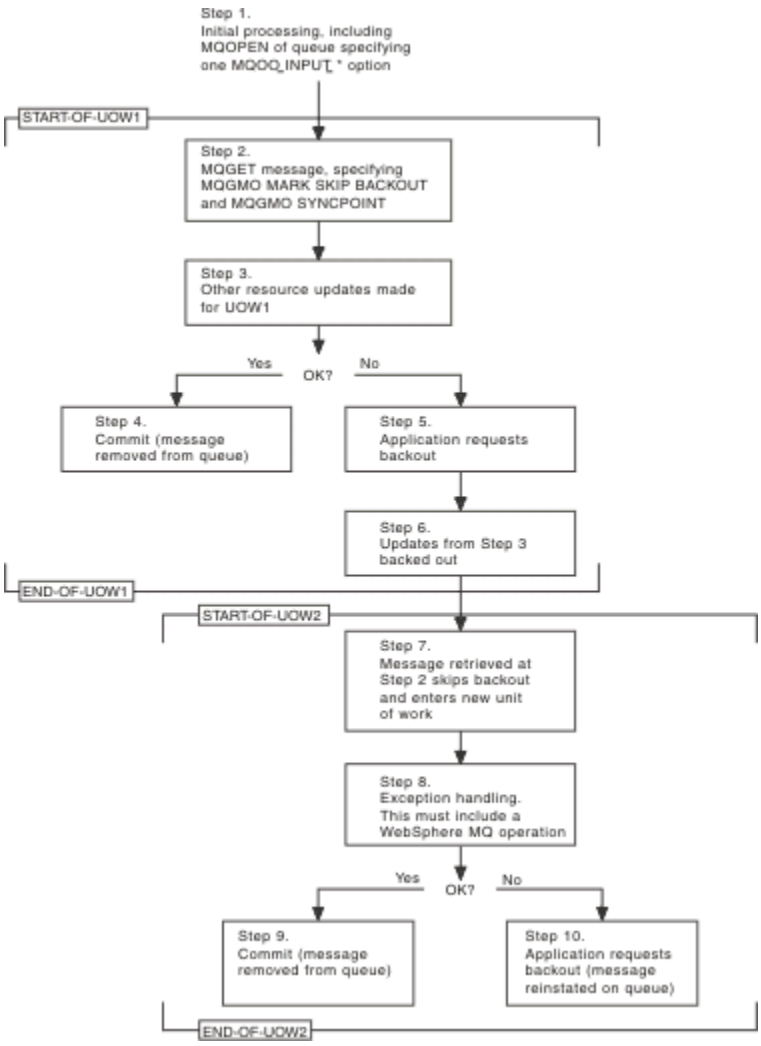


Figure 36. Annulation ignorée à l'aide de MQGMO_MARK_SKIP_BACKOUT

Les étapes de la rubrique Figure 36, à la page 281 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 36, à la page 281 , 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.

Etape 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 .)

Etape 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 conditions suivantes sont remplies:

- 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 432. 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 WebSphere 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 256.)

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 la file d'attente

Lorsque vous parcourez une file d'attente indexée qui ne contient que des messages d'une seule priorité (persistants ou non persistants ou les deux), le gestionnaire de files d'attente utilise l'index pour parcourir lorsque certaines formes de navigation sont utilisées.

Remarque : Pris en charge uniquement sur WebSphere MQ for z/OS.

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 256 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 286, 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 maintenant. 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 de publication / abonnement WebSphere MQ .

Pour une présentation des concepts de publication / abonnement, voir Introduction à la messagerie de publication / abonnement WebSphere MQ.

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 290
- «Ecriture d'applications d'abonné», à la page 297
- «Cycles de vie de publication / abonnement», à la page 315
- «Propriétés des messages de publication / abonnement», à la page 320
- «Ordre des messages», à la page 322
- «Interception des publications», à la page 322
- «Options de publication», à la page 330
- «Options d'abonnement», à la page 330

Concepts associés

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Exemples de programmes WebSphere MQ», à la page 100
Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

«Ecriture d'une application de mise en file d'attente», à la page 202
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 client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Utilisation des services Web dans WebSphere MQ», à la page 977
Vous pouvez développer des applications IBM WebSphere MQ pour les services Web à l'aide du transport IBM WebSphere MQ pour SOAP ou du pont IBM WebSphere MQ pour HTTP.

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

«Traitement des erreurs de programme», à la page 569
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.

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 WebSphere MQ simple est similaire à l'écriture d'une application de point à point WebSphere MQ qui place des messages dans une file d'attente (Tableau 41, à la page 291). La différence est que les messages MQPUT sont envoyés à une rubrique et non à une file d'attente.

Tableau 41. Point à point par rapport au modèle de programme WebSphere 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 291), 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 294), le modèle d'interaction du programme avec WebSphere 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 297
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application WebSphere 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
DISPLAY TOPIC
DISPLAY TPSTATUS

Exemple 1: diffuseur de publications vers une rubrique fixe

Un programme WebSphere 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 38, à la page 292

```
#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 37. Publieur WebSphere 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 38. 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ dans le cadre de la mémoire tampon de messages.

Que démontre le premier exemple? L'exemple imite le plus près possible le modèle traditionnel essayé et testé pour l'écriture de point à point dans les programmes WebSphere MQ . Une caractéristique importante du modèle de programmation WebSphere MQ est que le programmeur n'est pas concerné par l'endroit où les 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. Dans l'architecture du modèle de publication / abonnement WebSphere MQ , la notion d'objet de rubrique d'administration fournit la partie racine, ou sous-arborescence racine, d'une rubrique. 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 294 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 294
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 297
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application WebSphere 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 40, à la page 296.

```
#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 39. Publieur WebSphere 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 40. 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. WebSphere MQ utilise l'arborescence implicite de 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 41, à la page 297). 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 40, à la page 296 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 au préalable à l'aide de WebSphere MQ Explorer 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

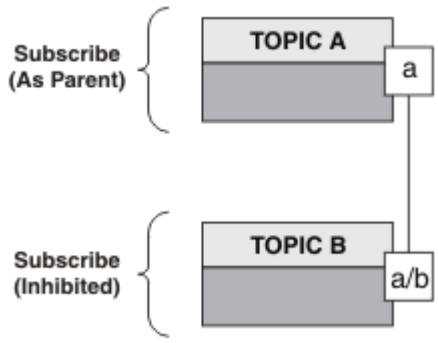


Figure 41. Associations d'objets de rubrique

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 WebSphere 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 lâche 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 le diffuseur de publications et l'abonné connaissent ou non l'arborescence de rubriques sous-jacente, et si les programmes d'abonné connaissent ou non 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 291
Un programme WebSphere MQ pour illustrer la publication dans une rubrique définie par l'administrateur.

«Ecriture d'applications d'abonné», à la page 297
Commencez par écrire des applications d'abonné en étudiant trois exemples: une application WebSphere 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 WebSphere 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 42, à la page 298 , les trois styles de consommateur ou d'abonné sont répertoriés, ainsi que les séquences d'appels de fonction WebSphere 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 administrativement à l'aide de WebSphere 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.

3. Dans le troisième style, l'application d'abonné choisit d'ouvrir et de fermer la file d'attente sous-jacente utilisée pour les publications et d'émettre des abonnements pour remplir la file d'attente avec des publications.

Une façon de comprendre ces styles consiste à étudier les exemples de programmes C répertoriés dans Tableau 42, à la page 298 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 290 .

Tableau 42. Modèles de programme WebSphere MQ de type point à point ou abonnement.

Etape	Consommateur de message MQ	«Exemple 1: consommateur de publication MQ», à la page 298	«Exemple 2: abonné MQ géré», à la page 301	«Exemple 3: Abonné MQ non géré», à la page 306
Connexion à un gestionnaire de files d'attente	MQCONN	MQCONN	MQCONN	MQCONN
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 315.

Exemple 1: consommateur de publication MQ

Le consommateur de publication MQ est un consommateur de message IBM WebSphere 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 WebSphere 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. Pour plus d'informations, voir «Exemple 3: Abonné MQ non géré», à la page 306.

Remarque : Le style de codage compact est destiné à la lisibilité et non à l'utilisation en production.

Les résultats sont affichés dans Figure 43, à la page 300.

```
#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 42. 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 43. Sortie du consommateur de publication MQ

Il existe quelques conseils de programmation de langage WebSphere 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 WebSphere 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 301
L'abonné MQ géré est le modèle préféré pour la plupart des applications d'abonné. 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 306
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. L'exemple illustre différentes manières de combiner des abonnements et des files d'attente.

«Ecriture d'applications de publication», à la page 290
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. L'exemple requiert *aucune* définition d'administration des files d'attente, des rubriques ou des abonnements .

Cette sorte d'abonné géré la plus simple utilise généralement un abonnement *non durable* . L'exemple se concentre sur un abonnement non durable. L'abonnement dure uniquement tant que 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 était 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 des 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. Cycle de vie des abonnements à gestion automatique: *non-Les abonnementsdurables* existent uniquement pendant la durée de l'application d'abonné.
   - Si vous définissez un abonnement géré *durable* , il génère 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: par à l'aide d'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, pour , 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 "sub-rubriques" sans se préoccuper de la gestion des rubriques.
   - est 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. Voir «Exemple 2: diffuseur de publications vers une rubrique de variable», à la page 294.

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 pour une entréeexclusive.

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, «Exemple 3: Abonné MQ non géré», à la page 306, 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 46, à la page 305.

```
#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 44. 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 l'espace de stockage 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. Cela 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 nécessiteaucune 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, vous pouvez donc 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 45. 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 46. Sortie de l'abonné MQ géré

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 les plus couramment utilisées ; les autres sont laissées à la valeur par défaut .

1. Comme l'abonnement est *non durable*, c'est-à-dire qu'il possède 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. Avec les droits appropriés, 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 306.
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 WebSphere 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. Cette empêche l'écrasement 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 est non durable, vous n'êtes probablement pas intéressé par son nom, mais vous pouvez inspecter son statut dans WebSphere MQ Explorer. Vous pouvez fournir le paramètre `sd.SubName` en tant que 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 298
Le consommateur de publication MQ est un consommateur de message IBM WebSphere MQ qui ne s'abonne pas aux rubriques lui-même.

«Exemple 3: Abonné MQ non géré», à la page 306
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. L'exemple illustre différentes manières de combiner des abonnements et des files d'attente.

«Ecriture d'applications de publication», à la page 290
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. 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.

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, vous ne pouvez pas créer plus d'un abonnement pour chaque file d'attente gérée et vous pouvez 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 49, à la page 312 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 WebSphere MQ Explorer.
- 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 43, à la page 309) 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`. Figure 55, à la page 315 affiche 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 48, à la page 311 pour vous familiariser avec ces différents types d'erreur. 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 43, à la page 309 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 la Figure 50, à la page 313 à Figure 55, à la page 315.

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 43, à la page 309, `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 43, à la page 309 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 43. Erreurs de MQSUB avec des descripteurs de file d'attente et des combinaisons d'abonnements différents

	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
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 47. 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 48. 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 Alter | C reate | Resume 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 49. 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 50, à la page 313 , 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 50. Publier 130 dans NYSE/IBM/PRICE

Dans l'exécution Figure 51, à la page 313 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 illustré dans la Figure 55, à la page 315. 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 51. Recevoir la publication conservée

Dans (Figure 52, à la page 314), 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 52. Reprendre l'abonnement

Dans Figure 53, à la page 314 , 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 53. Recevoir la publication conservée avec un nouvel abonnement non durable non géré

Dans Figure 54, à la page 314 , 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 54. Chevauchement d'abonnements

Dans Figure 55, à la page 315 , 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 55. Les rubriques d'abonnement ne peuvent pas être modifiées

Concepts associés

«Exemple 1: consommateur de publication MQ», à la page 298
Le consommateur de publication MQ est un consommateur de message IBM WebSphere MQ qui ne s'abonne pas aux rubriques lui-même.

«Exemple 2: abonné MQ géré», à la page 301
L'abonné MQ géré est le modèle préféré pour la plupart des applications d'abonné. L'exemple requiert *aucune* définition d'administration des files d'attente, des rubriques ou des abonnements .

«Ecriture d'applications de publication», à la page 290
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 WebSphere MQ , tels que les files d'attente, sont créés de manière administrative ou par des programmes d'administration à l'aide du 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[2]. 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 56, à la page 317 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.

[2] L'éditeur doit envoyer les dossiers passagers en tant que messages persistants pour éviter d'autres défaillances possibles, bien sûr.

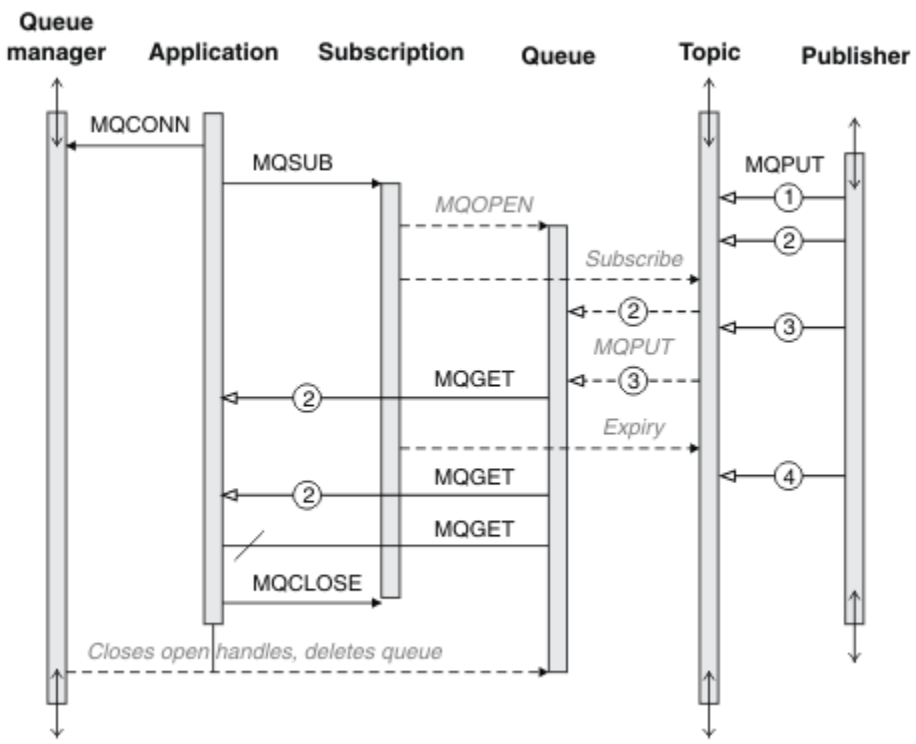


Figure 56. Lignes de vie d'abonné non durables gérées

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 WebSphere 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.

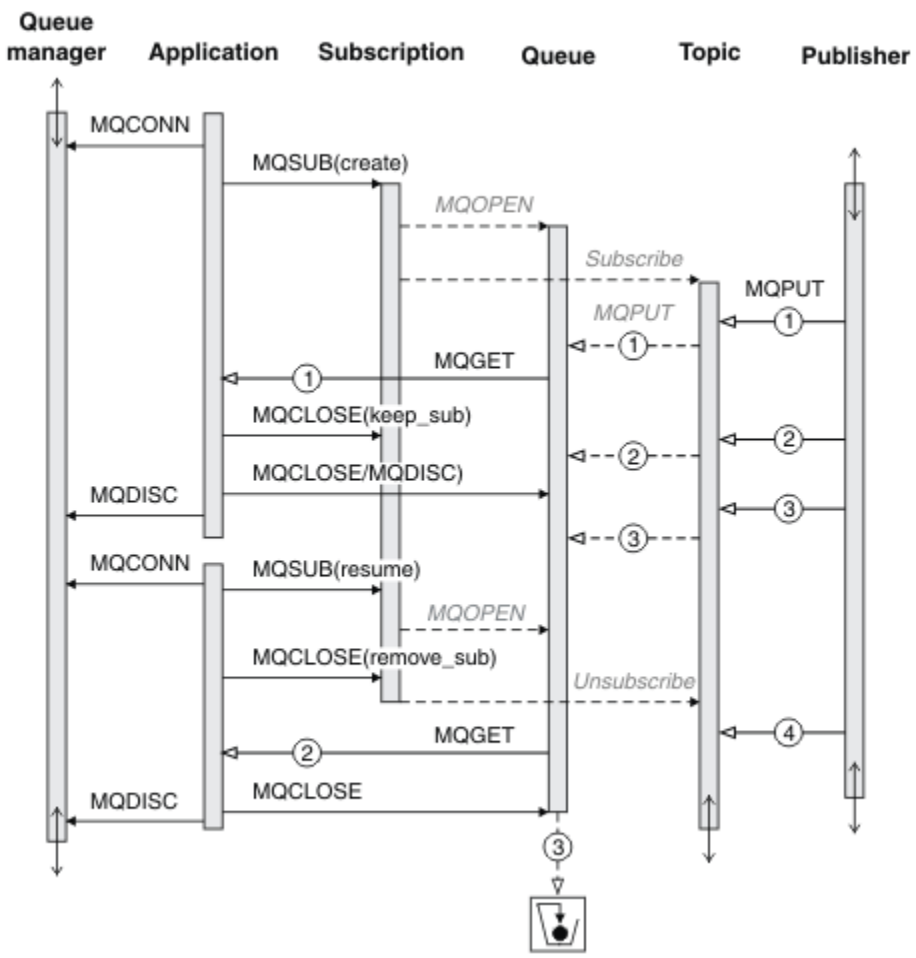


Figure 57. Lignes de vie d'abonné durable gérées

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.

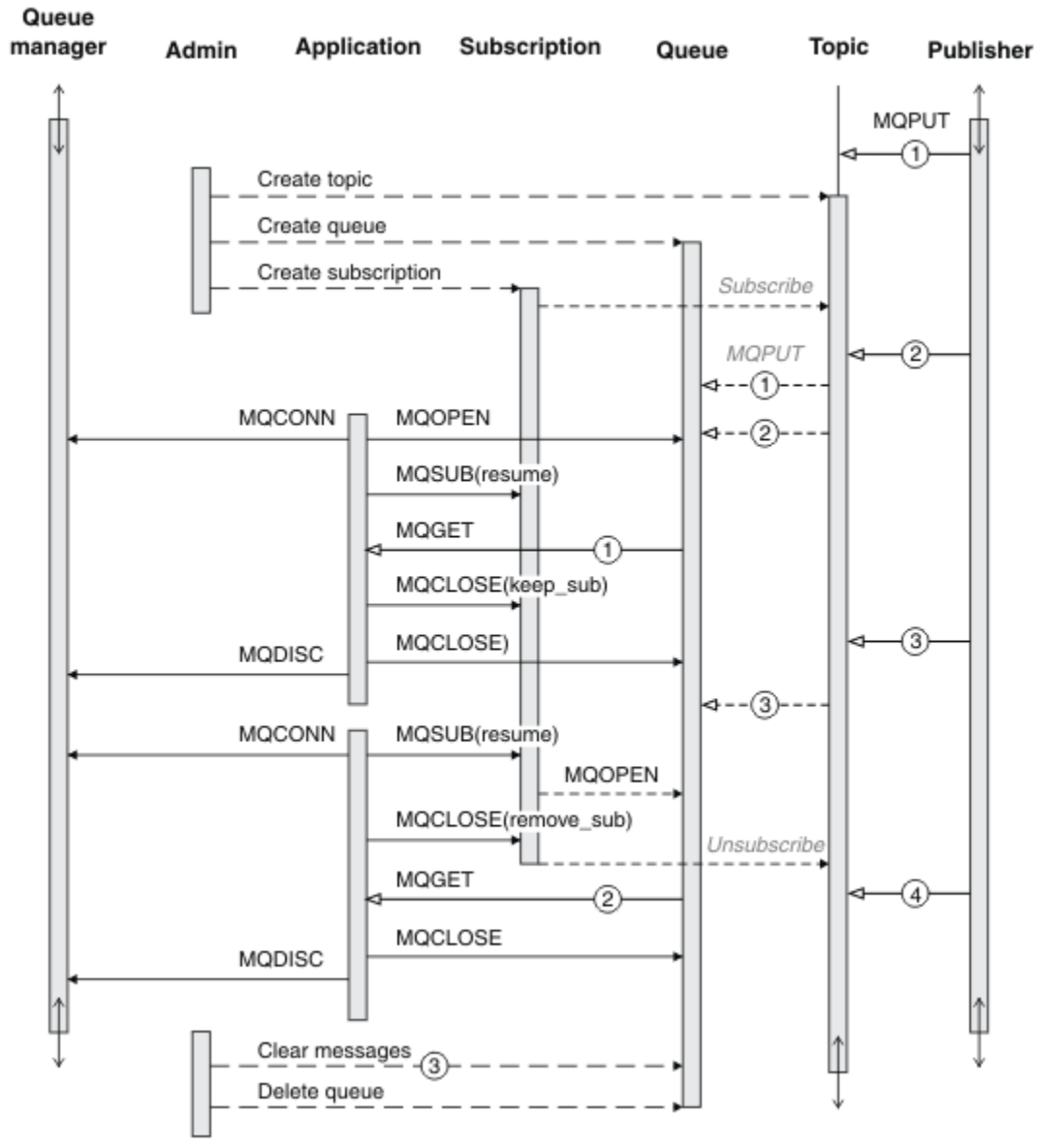


Figure 58. Lignes de vie d'abonné durable non gérées

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 WebSphere 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 39. 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 39. 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 extraire leurs publications, l'utilisation de MQGET par ID de corrélation permet d'obtenir uniquement des publications pour 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 44, à la page 321 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 44. 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 44. 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 WebSphere 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.

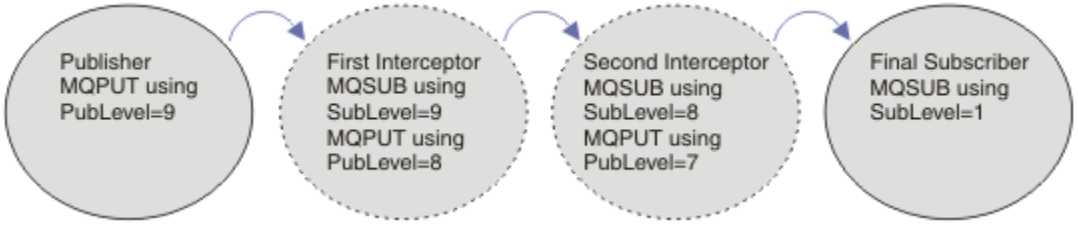


Figure 59. Séquence d'interception des abonnés

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 45, à la page 323.

Tableau 45. 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 45, à la page 323 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 46, à la page 324 dans les zones correspondantes du message republié. L'abonné intercepteur peut modifier ces valeurs. Utilisez l'opérateur OR pour ajouter des valeurs supplémentaires à la zone **MQPMO**.`Options` afin de 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 324.

Tableau 46. 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 46, à la page 324. 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 60, à la page 325 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 60. Directives de préprocesseur

Le Figure 61, à la page 325 répertorie les déclarations utilisées dans les fragments de code. A l'exception des termes mis en évidence, les déclarations sont standard pour une application WebSphere 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 61. Déclarations

Les initialisations qui ne sont pas faciles à effectuer dans des déclarations sont présentées dans Figure 62, à la page 326. 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 WebSphere 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.

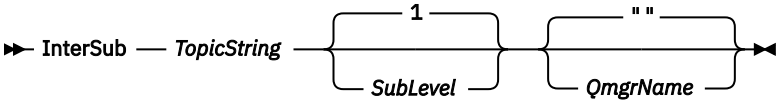
MQPRI_PRIORITY_AS_PUBLISHED
La publication est placée dans la file d'attente de publication avec la même priorité de message que celle avec laquelle elle a été publiée.

```
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 62. Initialisations

Figure 63, à la page 327 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 63. Préparation de l'interception des publications

Le fragment de code principal, Figure 64, à la page 328, 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 des propriétés **MQPMO**.option d'origine 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 64. Intercepter la publication et la republier

Le fragment de code final est illustré dans la Figure 65, à la page 328.

```
    } 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 65. 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 66, à la page 329 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.

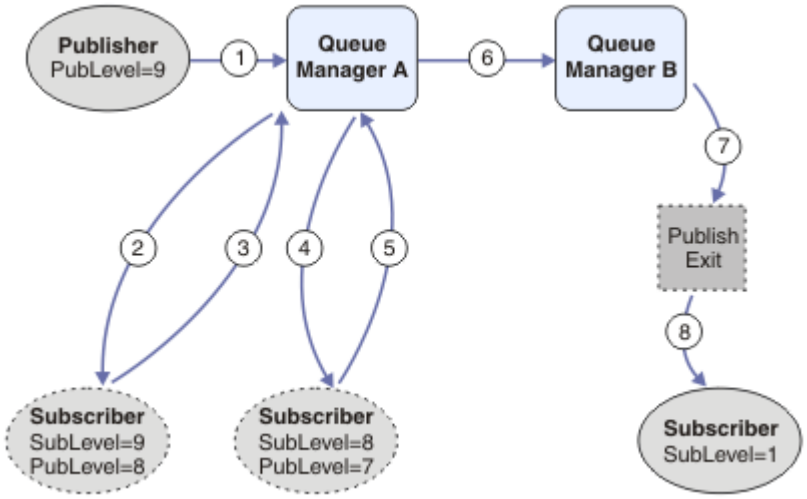


Figure 66. Exit d'interception et de publication dans un cluster

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 67, à la page 330 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 67, à la page 330, 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.

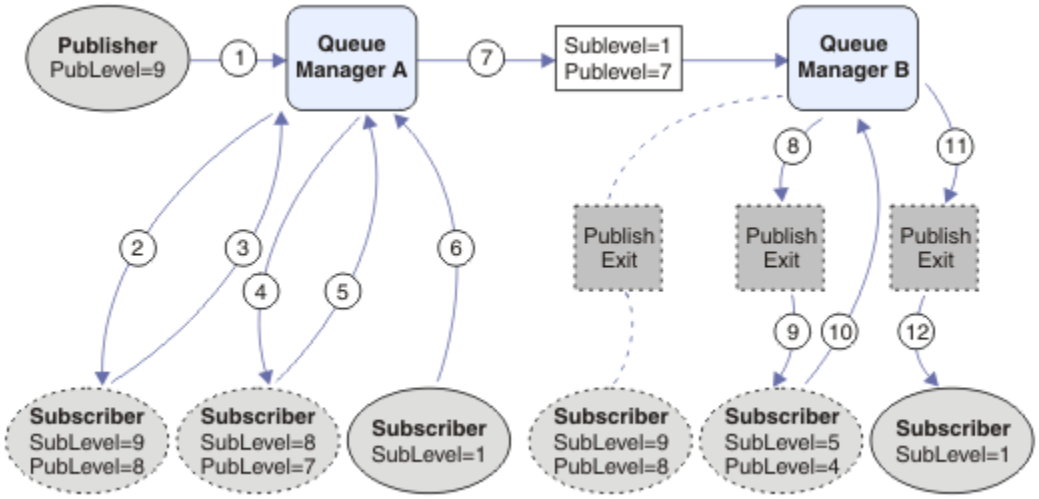


Figure 67. Déploiement complexe de l'interception des abonnés

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 ../com.ibm.mq.ref.dev.doc/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 WebSphere MQ .

Elles affectent la manière dont un gestionnaire de files d'attente traite un objet. Les attributs de chaque type d'objet WebSphere MQ sont décrits en détail dans Attributs d'objets.

Certains attributs sont définis lors de la définition de l'objet et peuvent être modifiés uniquement à l'aide des commandes WebSphere MQ ; un exemple d'attribut de ce type est la priorité par défaut des messages placés dans une file d'attente. 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. Vous devez utiliser:

Pour les plateformes WebSphere MQ for Windows, UNIX and Linux

La fonction MQSC, décrite dans MQSC reference.

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 WebSphere MQ pour définir, modifier ou afficher les attributs, vous devez les identifier à l'aide des mots clés affichés dans les descriptions des commandes des liens vers les rubriques.

Les appels MQINQ et MQSET utilisent des tableaux de sélecteurs pour identifierles 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:

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 215 et «Ouverture et fermeture d'objets», à la page 223.

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 333
- «Certains cas où l'appel MQINQ échoue», à la page 334
- «Définition des attributs de file d'attente», à la page 335

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
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 250
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 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

Interrogation des attributs d'un objet

Utilisez l'appel MQINQ pour vous renseigner sur les attributs de n'importe quel type de IBM WebSphere 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 362 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, vous obtenez les attributs de la file d'attente alias (l'objet WebSphere MQ utilisé pour accéder à une autre file d'attente) 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)
- *DistList* (hors z/OS)
- *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 WebSphere MQ autres que les files d'attente définies en local.

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:

- Appliquer
- 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 WebSphere MQ», à la page 337.

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 WebSphere 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 WebSphere 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 sur d'autres plateformes, à l'exception de z/OS. Pour plus de détails sur ces commandes, voir Référence MQSC *WebSphere MQ Script (MQSC) Command Reference* .

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 WebSphere 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 WebSphere 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 WebSphere MQ», à la page 337
- «Points de synchronisation dans IBM WebSphere MQ sur les systèmes UNIX, Linux, and Windows», à la page 338

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 223

Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
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 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Démarrage des applications IBM WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

«Utilisation de l'interface MQI et des clusters», à la page 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

Remarques sur les points de synchronisation dans les applications IBM WebSphere MQ

Utilisez ces informations pour en savoir plus sur l'utilisation des points de synchronisation dans les applications IBM WebSphere MQ .

La validation en deux phases est prise en charge sous:

- WebSphere MQ for AIX
- WebSphere MQ pour HP-UX
- WebSphere MQ pour Linux
- WebSphere MQ pour Solaris
- WebSphere MQ pour Windows
- CICS for MVS/ESA 4.1
- CICS Transaction Server for z/OS
- TXSeries
- IMS/ESA
- Autres coordinateurs externes utilisant l'interface X/Open XA

La validation en une phase est prise en charge sous:

- WebSphere MQ sur les systèmes UNIX
- WebSphere MQ pour Windows

Remarque : Pour plus de détails sur les interfaces externes, voir «Interfaces avec les gestionnaires de points de synchronisation externes», à la page 340, et la documentation XA *CAE Specification Distributed Transaction Processing: The XA Specification*, publiée par The Open Group. Les gestionnaires de transactions (tels que CICS, IMS, Encinaet 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 WebSphere MQ peuvent être placées dans la portée d'une unité de travail gérée par le gestionnaire de transactions.

Les exemples fournis avec WebSphere MQ montrent WebSphere MQ coordonnant les bases de données compatibles XA. Pour plus d'informations sur ces exemples, voir «Exemples de programmes pour les plateformes réparties», à la page 100.

Dans votre application WebSphere 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. La valeur par défaut du contrôle de point de synchronisation est non.

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. Si le programme s'arrête de manière anormale, une annulation implicite se produit.

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.

Points de synchronisation dans IBM WebSphere MQ sur les systèmes UNIX, Linux, and Windows

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 WebSphere 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. WebSphere MQ peut coordonner de telles unités de travail. Ils peuvent également être coordonnés par un contrôleur de validation externe, tel qu'un autre gestionnaire de transactions ou 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, tels que TXSeries et UDB. Les produits WebSphere MQ (sauf WebSphere MQ for IBM i et WebSphere MQ for z/OS) peuvent coordonner des unités de travail globales à l'aide d'un processus de validation en deux phases.

Unités d'oeuvre locales

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 WebSphere 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.

Unités d'oeuvre globales

Utilisez des unités d'oeuvre globales lorsque vous devez également inclure des mises à jour des ressources appartenant à d'autres gestionnaires de ressources.

Ici, 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 par WebSphere MQ for IBM i ou WebSphere MQ for z/OS. Il n'est pas pris en charge dans un WebSphere MQ environnement client MQI.

Ici, WebSphere 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 WebSphere 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 par 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 préparer 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 WebSphere MQ a été sélectionné ; par exemple, CICS, Encinaou Tuxedo.

Dans cette situation, WebSphere MQ sur les systèmes UNIX and Linux et WebSphere MQ for Windows enregistrent leur intérêt pour le résultat de l'unité d'oeuvre 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 API WebSphere MQ dans les programmes CICS , que vous choisissiez d'utiliser les capacités 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», à la page 42, ainsi que la documentation TXSeries CICS , par exemple *TXSeries for Multiplatforms CICS Administration Guide for Open Systems*.

Interfaces avec les gestionnaires de points de synchronisation externes

WebSphere MQ sur les systèmes UNIX and Linux , et WebSphere MQ for Windows prennent en charge la coordination des transactions par des 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. Le gestionnaire de ressources pour WebSphere MQ sur les systèmes UNIX, Linux et Windows est nommé 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, les systèmes WebSphere MQ on UNIX, Linuxet Windows jouent le rôle de Resource ManagerXA. 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 Resource Manager fournisse une structure appelée *commutateur XA*. Cette structure déclare les fonctionnalités du Resource Manageret les fonctions qui doivent être appelées par le coordinateur de point de synchronisation.

Il existe deux versions de cette structure:

MQRMIXASwitch	Gestion des ressources XA statiques
MQRMIXASwitchDynamic	Gestion des ressources XA dynamiques

Pour la liste des bibliothèques contenant cette structure, voir «Structure de commutation IBM WebSphere MQ XA», à la page 73.

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 activer WebSphere MQ pour 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 permet d'indiquer à WebSphere 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 221.
2. Sur les systèmes Windows , toutes les fonctions déclarées dans le commutateur XA sont déclarées en tant que fonctions _cdecl.
3. 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 (CLIENT JEE) s'exécutant 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.

4. 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.

Démarrage des applications IBM WebSphere MQ à l'aide de déclencheurs

Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

Certaines applications WebSphere MQ qui servent les 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.

WebSphere 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 WebSphere MQ s'exécutant dans les environnements suivants:

- Systèmes UNIX and Linux
- Systèmes Windows

Une application exécutée dans un environnement client est identique à une application exécutée dans un environnement WebSphere 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 les systèmes AIX, HP-UX, IBM i, Solaris, z/OSou Windows , 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 (voir «Conditions d'un événement déclencheur», à la page 348). WebSphere MQ dispose d'une série d'options permettant de contrôler les conditions qui provoquent un événement déclencheur (voir «Contrôle des événements déclencheurs», à la page 353).

Message de déclenchement
Le gestionnaire de files d'attente crée un *message de déclenchement* lorsqu'il reconnaît un événement déclencheur (voir «Conditions d'un événement déclencheur», à la page 348). 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 360).

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 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. 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 sur WebSphere 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. 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. Sur les autres plateformes, il est automatiquement démarré lorsque le gestionnaire de files d'attente démarre 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 357.)

Pour comprendre le fonctionnement du déclenchement, prenez en compte Figure 68, à la page 344, qui est un exemple de type de déclencheur FIRST (MQTT_FIRST).

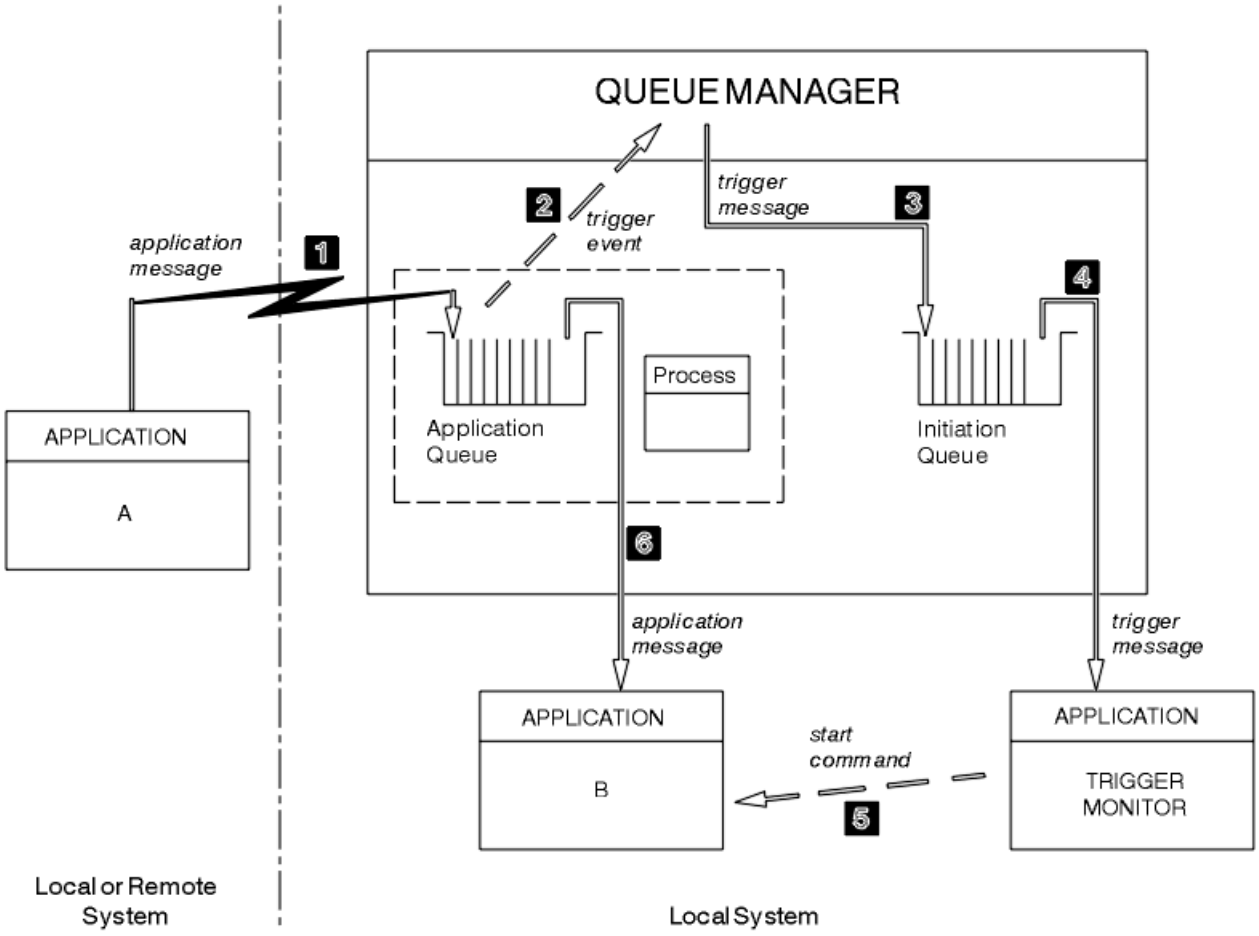


Figure 68. Flux de messages d'application et de déclenchement

Dans Figure 68, à la page 344, 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 69, à la page 346.

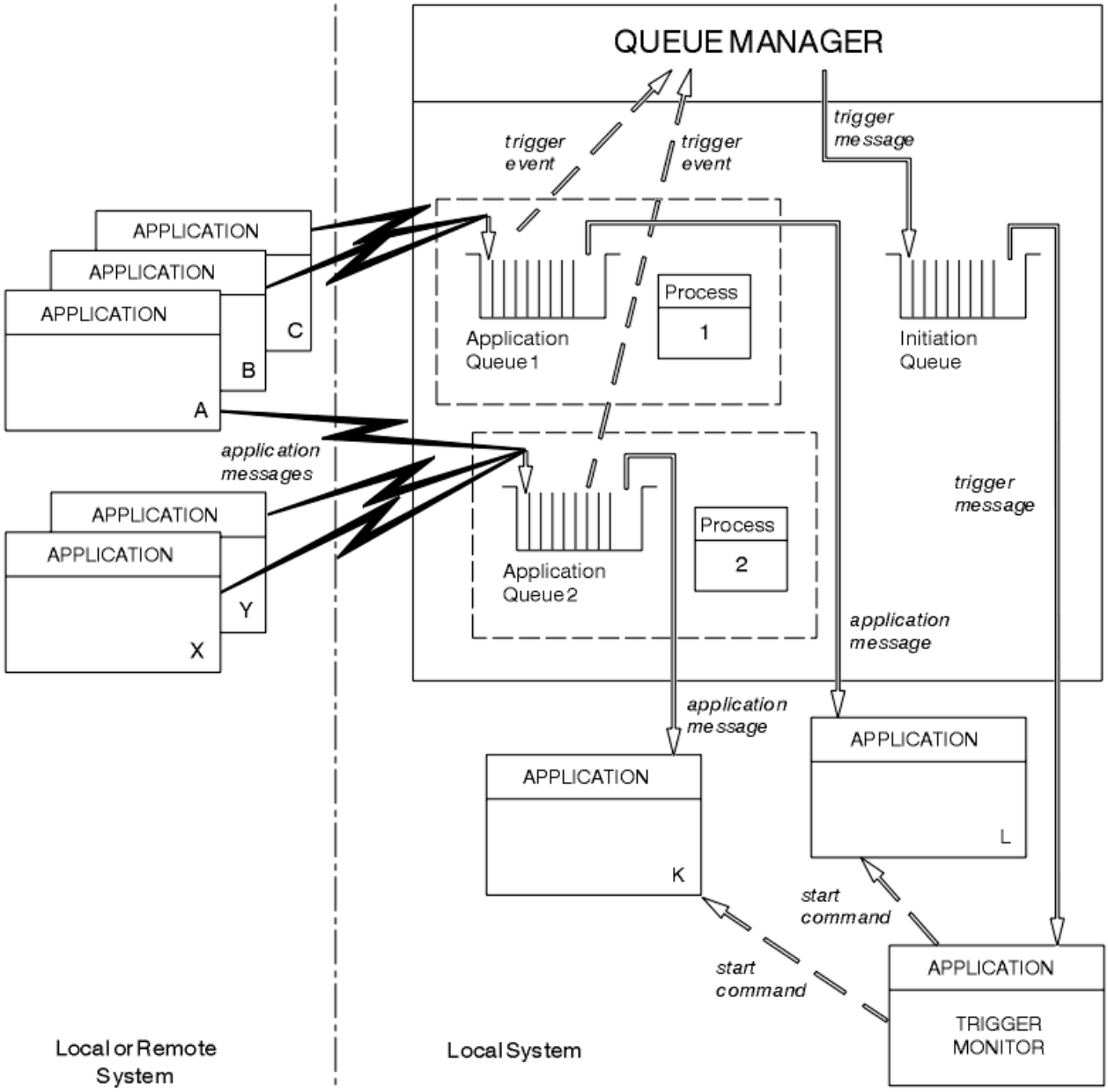


Figure 69. Relation des files d'attente dans le déclenchement

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 des applications WebSphere MQ à l'aide de déclencheurs:

- «Prérequis pour le déclenchement», à la page 347
- «Conditions d'un événement déclencheur», à la page 348
- «Contrôle des événements déclencheurs», à la page 353
- «Conception d'une application qui utilise des files d'attente déclenchées», à la page 355

- «Traitement de la file d'attente d'initialisation par les moniteurs de déclenchement», à la page 357
- «Propriétés des messages de déclenchement», à la page 359
- «Lorsque le déclenchement ne fonctionne pas», à la page 361

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215
Pour utiliser les services de programmation WebSphere 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 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
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 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 361
Il existe des options spéciales sur les appels et les codes retour liés à la mise en cluster.

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)
```

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')
```

   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
Systèmes UNIX, Linuxet Windows	Utilisez DEFINE PROCESS ou SYSTEM.DEFAULT.PROCESS et modification à l'aide de ALTER PROCESS

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 sous IBM WebSphere 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 les commandes
Systèmes UNIX, Linuxet Windows	ALTER QLOCAL

6. Démarrez les instances des moniteurs de déclenchement 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 357.

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 353.

Conditions d'un événement déclencheur

Les références aux files d'attente partagées dans cette rubrique désignent les files d'attente partagées dans un groupe de partage de files d'attente, disponibles uniquement dans WebSphere MQ for z/OS.

Le gestionnaire de files d'attente crée un message de déclenchement lorsque les conditions suivantes sont remplies:

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 :

   a. 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.

   b. 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 :

   a. 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.

   b. 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. Sous WebSphere MQ for z/OS, si la file d'attente d'application est une file d'attente avec l'attribut *Usage* MQUS_NORMAL, les demandes d'extraction pour cette file d'attente 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 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 l'attribut de file d'attente *InhibitPut* doit être MQQA_PUT_ALLOWED).
   - 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 349 à «10», à la page 350 (à l'exception de «3», à la page 349) 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 *TriggerInterval temporisateur* 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 ayant une priorité suffisante (condition «2», à la page 349) et les conditions «6», à la page 350 à «10», à la page 350 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 38.

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)

       les messages de la file d'attente ayant une priorité suffisante (condition «2», à la page 349) et les conditions «4», à la page 349 à «10», à la page 350 (à l'exception de «8», à la page 350) 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. 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)

Les messages ayant une priorité suffisante (condition «2», à la page 349) 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 349 à «10», à la page 350 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. L'attribut de file d'attente *InhibitGet* d'une file d'attente d'application passe de MQQA_GET_INHIBITION à MQQA_GET_ALLOWED, et il existe au moins:

- Un (MQTT_FIRST ou MQTT_EVERY) ou
- *TriggerDepth* (MQTT_DEPTH)

Les messages de priorité suffisante (condition «2», à la page 349) dans la file d'attente et les conditions «4», à la page 349 à «10», à la page 350, à l'exception de «5», à la page 350, sont également satisfaits.

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)

les messages ayant une priorité suffisante (condition «2», à la page 349) 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 349 à «10», à la page 350 (à l'exclusion de «8», à la page 350) sont également satisfaites, et aucune autre application ne dispose de 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 353.

    Si vous définissez MSGDLVSQ=PRIORITY et que la priorité du message est supérieure ou égale à la zone *TriggerMsgPriority* , les messages ne sont *comptés* 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.

Remarque :

1. A partir de l'étape «12», à la page 351 (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 WebSphere 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, WebSphere 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. Toutefois, chacun des trois segments entraîne le traitement de ses propres événements déclencheurs lorsqu'ils sont déplacés via le réseau WebSphere MQ .

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 également «Cas particulier du type de déclencheur FIRST», à la page 355.

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 < xph> < pv>TriggerControl< /pv> < /xph>) après avoir créé 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 348 .

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.
- Sous WebSphere MQ for z/OS, les files d'attente d'application qui nomment une file d'attente d'initiation ouverte sont analysées périodiquement. 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.)
- Sur WebSphere MQ for z/OS si vous définissez *TRIGINT* sur une valeur faible et qu'il n'existe aucune application servant la file d'attente d'application de type FIRST, le déclenchement de l'arrêt du système génère un message de déclenchement chaque fois que l'analyse périodique des files d'attente d'application qui portent le nom de files d'attente d'initialisation ouvertes est effectuée.
- Si une unité de travail est annulée (voir Messages de déclenchement et unités de travail) et que 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.

Moniteurs de déclenchement sur les systèmes UNIX et Windows

Cette rubrique contient des informations sur les moniteurs de déclenchement fournis sur les systèmes UNIX et 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 «Exemples de programmes pour les plateformes réparties», à la page 100 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 WebSphere MQ sur le nombre d'applications pouvant être exécutées.

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
```

Pour exécuter l'application démarrée en arrière-plan sur les systèmes UNIX , placez un & à la fin du *EnvData* de la définition PROCESS.

Remarque : 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 de 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 WebSphere MQ :

runmqtmc
Il est identique à runmqtrm, sauf qu'il est lié aux bibliothèques client WebSphere MQ MQI.

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 d'une autre manière et il déclenche des transactions CICS .

Cette rubrique s'applique uniquement aux systèmes Windows, UNIXet Linux .

Il 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 (nommé dans le fichier qm.ini ou, dans WebSphere 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 du moniteur de déclenchement MQTMC2 : 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 de 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.

Voici les versions disponibles du moniteur de déclenchement:

Version	**Utiliser**
amqltmc0	TXSeries pour AIX, HP-UXet Sun Solaris version 5.1
amqltmc4	TXSeries for Windows, version 5.1
amqltmcc	Version liée au client du moniteur de déclenchement CICS

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.
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 plus d'informations sur l'interface du moniteur de déclenchement, voir MQTMC2.

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 359
- «Messages de redémarrage et de déclenchement du gestionnaire de files d'attente», à la page 360
- «Messages de déclenchement et modifications des attributs d'objet», à la page 360
- «Format des messages de déclenchement», à la page 360

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. Sur tout produit WebSphere MQ sauf WebSphere MQ for z/OS, cette zone peut être utilisée pour indiquer 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 ou CSQQTRMN fourni par WebSphere MQ for z/OS,

l'attribut *ApplId* de l'objet de définition de processus est un identificateur de transaction CICS ou 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*. Les moniteurs de déclenchement fournis par WebSphere MQ for z/OS(CKTI ou CSQQTRMN) n'utilisent 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 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 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 362
- «MQPUT, MQPUT1 et clusters», à la page 363
- «MQINQ et clusters», à la page 364
- «MQSET et clusters», à la page 364
- «Codes retour», à la page 365

Concepts associés

«Présentation de l'interface de file d'attente de messages», à la page 203
Découvrez les composants MQI (Message Queue Interface).

«Connexion et déconnexion d'un gestionnaire de files d'attente», à la page 215

Pour utiliser les services de programmation WebSphere 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 223
Ces informations fournissent un aperçu de l'ouverture et de la fermeture des objets WebSphere MQ .

«Insertion de messages dans une file d'attente», à la page 235
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 250
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 332
Les attributs sont les propriétés qui définissent les caractéristiques d'un objet WebSphere MQ .

«Validation et annulation d'unités de travail», à la page 335
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 WebSphere MQ à l'aide de déclencheurs», à la page 342
Découvrez les déclencheurs et comment démarrer les applications IBM WebSphere MQ à l'aide de déclencheurs.

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.

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 WebSphere 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, 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 locale. 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
    - nomQ
    - 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 230
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 du cluster, défini par l'attribut ClusterWorkloadExit d'un gestionnaire de files d'attente, échoue de manière inattendue ou ne répond pas à temps.

Un message est consigné dans le journal système sur WebSphere MQ for z/OS et fournit des informations supplémentaires 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')

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.

Sur les plateformes autres que z/OS, 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 conditions suivantes sont remplies:

- 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.

Ecriture d'applications client

Informations à connaître pour écrire des applications client sur WebSphere MQ.

Les applications peuvent être générées et exécutées dans l'environnement client WebSphere MQ . L'application doit être générée et liée au client WebSphere MQ MQI 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 les clients WebSphere MQ MQI», à la page 372.

Vous pouvez exécuter une application WebSphere MQ à la fois dans un environnement WebSphere MQ complet et dans un environnement client WebSphere MQ MQI 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 WebSphere MQ , voir «Exécution d'applications dans l'environnement client IBM WebSphere MQ MQI», à la page 374.

Si vous utilisez l'interface de file d'attente de messages (MQI) pour écrire des applications à exécuter dans un environnement client MQI WebSphere MQ , il existe des contrôles supplémentaires à imposer lors d'un appel MQI afin de garantir que le traitement de l'application WebSphere MQ n'est pas interrompu. Pour plus d'informations sur ces contrôles, voir «Utilisation de l'interface de file d'attente de messages (MQI) dans une application client», à la page 367.

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 387
- «Préparation et exécution des applications Microsoft Transaction Server», à la page 41
- «Préparation et exécution des applications JMS WebSphere MQ», à la page 390

Concepts associés

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Exemples de programmes WebSphere MQ», à la page 100

Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

«Ecriture d'une application de mise en file d'attente», à la page 202
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.

«Utilisation des services Web dans WebSphere MQ», à la page 977
Vous pouvez développer des applications IBM WebSphere MQ pour les services Web à l'aide du transport IBM WebSphere MQ pour SOAP ou du pont IBM WebSphere MQ pour HTTP.

«Ecriture d'applications de publication / abonnement», à la page 290
Commencez à écrire des applications de publication / abonnement WebSphere MQ .

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

«Traitement des erreurs de programme», à la page 569
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.

Utilisation de l'interface de file d'attente de messages (MQI) dans une application client

Cette collection de rubriques prend en compte les différences entre l'écriture de votre application WebSphere MQ à exécuter dans un environnement client WebSphere MQ MQI et à exécuter dans l'environnement complet du gestionnaire de files d'attente WebSphere MQ .

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 WebSphere MQ n'est pas interrompu.

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.

Sur les plateformes autres que z/OS, vous pouvez augmenter l'attribut de longueur de message maximale d'un gestionnaire de files d'attente. Des détails sont fournis dans 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 WebSphere 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.

Choix de l'ID de jeu de caractères codés (CCSID) du client ou du serveur

Utilisez le CCSID local pour le client. Le gestionnaire de files d'attente effectue la conversion nécessaire. Utilisez 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 MQI de l'application au module de remplacement client doivent être dans le CCSID local, codées pour le client WebSphere MQ MQI. 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.

Le client Java dans V7, cependant, peut effectuer la conversion si le gestionnaire de files d'attente ne peut pas le faire. Pour plus d'informations, voir « WebSphere Classes MQ pour les connexions client Java», à la page 693.

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:

- Sous Windows:

```
SET MQCCSID=850
```

- Sur les systèmes UNIX:

```
export MQCCSID=850
```

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 WebSphere MQ après le descripteur de message (MQMD), sachez que les zones de 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 WebSphere 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.

MaxMsgLongueur
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 WebSphere 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 WebSphere MQ , elle peut émettre MQCMIT et MQBACK, mais la portée du contrôle de point de synchronisation est limitée aux ressources MQI. L'instruction WebSphere 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 WebSphere 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 WebSphere MQ . Voir Qu'est-ce qu'un client transactionnel étendu? pour plus de détails.

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 WebSphere MQ n'active la lecture anticipée que si certaines conditions sont remplies. Ces conditions sont les suivantes :

- La version du client et la version du gestionnaire de files d'attente éloignées doivent être la version 7 ou une version ultérieure de WebSphere MQ.
- L'application client doit être compilée et liée dans les bibliothèques client WebSphere MQ MQI à 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 269

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 WebSphere 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 120.

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. De même, l'authentification TLS/SSL n'a lieu que 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 deuxième conversation, même s'il peut modifier les zones MQCD, celles-ci ne sont *pas* mises en oeuvre 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 Utilisation de la structure MQCNO sur un appel MQCONNX. 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 d'exit de canal, voir «Programmes d'exit de canal pour les canaux de messagerie», à la page 413.

Descripteurs de connexion partagée sur MQCONNX

Vous pouvez partager des descripteurs entre différentes unités d'exécution au sein d'un même processus, à l'aide de descripteurs de connexion partagée.

Lorsque vous spécifiez un descripteur de connexion partagée, le descripteur de connexion renvoyé par l'appel MQCONNX peut être transmis dans les appels MQI suivants sur n'importe quelle unité d'exécution du processus.

Remarque : Vous pouvez utiliser un descripteur de connexion partagée sur un client WebSphere MQ MQI pour vous connecter à un gestionnaire de files d'attente de serveur qui ne prend pas en charge les descripteurs de connexion partagée.

Pour plus d'informations, voir «Utilisation de MQCONNX», à la page 371.

Génération d'applications pour les clients WebSphere MQ MQI

Les applications peuvent être générées et exécutées dans l'environnement client WebSphere MQ MQI. L'application doit être générée et liée au client WebSphere MQ MQI 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 47. Langages de programmation pris en charge dans les environnements client

Plateforme client	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Oui	Oui	Oui			
HP Integrity NonStop Server	Oui		Oui	Oui		
HP-UX	Oui	Oui	Oui			
Linux	Oui	Oui	Oui			
Solaris	Oui	Oui	Oui			
Fenêtres	Oui	Oui	Oui			Oui

Consultez les rubriques connexes pour obtenir des instructions sur la liaison ou la génération d'applications client dans ces langues.

Liaison d'applications C avec le code client WebSphere MQ MQI

Après avoir écrit votre application WebSphere MQ que vous souhaitez exécuter sur le client WebSphere MQ MQI, vous devez la lier à un gestionnaire de files d'attente.

Vous pouvez lier votre application à un gestionnaire de files d'attente de deux manières:

1. Directement, auquel cas le gestionnaire de files d'attente doit se trouver sur le même poste de travail que votre application
2. Vers un fichier de bibliothèque client, qui vous donne accès aux gestionnaires de files d'attente sur le même poste de travail ou sur un poste de travail différent

WebSphere MQ fournit un fichier de bibliothèque client pour chaque environnement:

AIX
Bibliothèque libmqic.a pour les applications sans unités d'exécution ou bibliothèque libmqic_r.a pour les applications avec unités d'exécution.

HP-UX
Bibliothèque libmqic.sl pour les applications non à unités d'exécution ou bibliothèque libmqic_r.sl pour les applications à unités d'exécution.

Linux
Bibliothèque libmqic.so pour les applications non à unités d'exécution ou bibliothèque libmqic_r.so pour les applications à unités d'exécution.

Solaris
libmqic.so.

Si vous souhaitez utiliser les programmes sur un poste de travail sur lequel seul le client WebSphere MQ MQI for Solaris est installé, vous devez recompiler les programmes pour les lier à la bibliothèque client:

```
$ /opt/SUNWspro/bin/cc -o <prog> <prog> c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Les paramètres doivent être entrés dans le bon ordre, comme indiqué.

Windows
MQIC32.LIB.

Liaison d'applications C++ avec le code client WebSphere MQ MQI

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 liaison de vos applications C + +, voir Génération de programmes C++ WebSphere MQ.

Pour plus de détails sur tous les aspects de l'utilisation de C + +, voir Utilisation de C++

Liaison d'applications COBOL avec le code client IBM WebSphere MQ MQI

Après avoir écrit une application COBOL que vous souhaitez exécuter sur le client IBM WebSphere MQ MQI, vous devez la lier à une bibliothèque appropriée.

IBM WebSphere 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.

HP-UX
Liez votre application COBOL sans unités d'exécution à la bibliothèque libmqicb.sl ou à l'application COBOL avec unités d'exécution à l'aide de libmqicb_r.sl.

Linux
Liez votre application COBOL sans unités d'exécution à la bibliothèque libmqicb.so ou à l'application COBOL avec unités d'exécution à libmqicb_r.so.

Solaris
Liez votre application COBOL sans unités d'exécution à la bibliothèque libmqicb.so ou à l'application COBOL avec unités d'exécution à libmqicb_r.so.

Windows
Liez votre code d'application à la bibliothèque MQICCBB pour COBOL 32 bits. Le client IBM WebSphere MQ MQI pour Windows ne prend pas en charge le langage COBOL 16 bits.

Liaison d'applications Visual Basic avec le code client WebSphere MQ MQI

Vous pouvez lier des applications Visual Basic avec le code client WebSphere MQ MQI sous Windows.

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 2000, Windows XP et Windows 2003

Exécution d'applications dans l'environnement client IBM WebSphere MQ MQI

Vous pouvez exécuter une application IBM WebSphere MQ à la fois dans un environnement IBM WebSphere MQ complet et dans un environnement client IBM WebSphere MQ MQI 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 client IBM WebSphere MQ MQI?

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 client IBM WebSphere MQ MQI, n'oubliez pas que:

- Chaque application exécutée dans l'environnement client IBM WebSphere MQ MQI dispose de 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 client IBM WebSphere MQ MQI aux gestionnaires de files d'attente

Une application s'exécutant dans un environnement client IBM WebSphere MQ MQI 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 exécutée dans un environnement client IBM WebSphere MQ émet un appel MQCONN ou MQCONNX, le client identifie la manière dont il doit établir la connexion. Lorsqu'un appel MQCONNX est émis par une application sur un client IBM WebSphere MQ , la bibliothèque client MQI recherche les informations de canal 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 440.
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 contient des paramètres ServerConnection, le canal qu'il définit est utilisé. Pour plus d'informations, voir Configuration d'un client à l'aide d'un fichier de configuration et la section CHANNELS du fichier de configuration du 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.
5. Si un fichier `mqclient.ini` est défini et qu'il contient des attributs de répertoire ChannelDefinitionet de fichier ChannelDefinition, ces attributs sont utilisés pour localiser la table de définition de canal du client. Pour plus d'informations, voir Configuration d'un client à l'aide d'un fichier de configuration et la section CHANNELS du fichier de configuration du 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 qui sont établis à partir de `DefaultPrefix` dans le fichier `mqs.ini` . Si la recherche d'une table de définition de client échoue, le client utilise les chemins suivants:
   - Systèmes UNIX and Linux : `/var/mqm/AMQCLCHL.TAB`
   - Windows: `C:\Program Files\IBM\Websphere MQ\amqclchl.tab`

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.

Si vous recevez un code retour MQRC_Q_MGR_NOT_AVAILABLE de votre application avec un message d'erreur dans le fichier journal des erreurs de `AMQ9517 -Fichier endommagé`, voir Migration and client channel definition tables (CCDT).

Concepts associés

Table de définition de canal du client

Tâches associées

Configuration des connexions entre le serveur et le client

Référence associée

MQSERVER

MQCHLLIB

MQCHLTAB

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 les variables d'environnement MQSERVER, MQCHLLIB et MQCHLTAB.

Pour plus d'informations sur ces variables, voir MQSERVER, MQCHLLIB et MQCHLTAB .

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 Utilisation de la structure MQCNO sur un appel MQCONNX .

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).

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.

Remarque : Le même fichier peut être utilisé par plusieurs clients IBM WebSphere MQ . Vous pouvez accéder à différentes versions de ce fichier à l'aide des variables d'environnement MQCHLLIB et MQCHLTAB IBM WebSphere MQ . Pour plus d'informations sur les variables d'environnement, voir Utilisation des variables d'environnement WebSphere MQ .

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'estpas prise en charge par WebSphere 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 70, à la page 378 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.

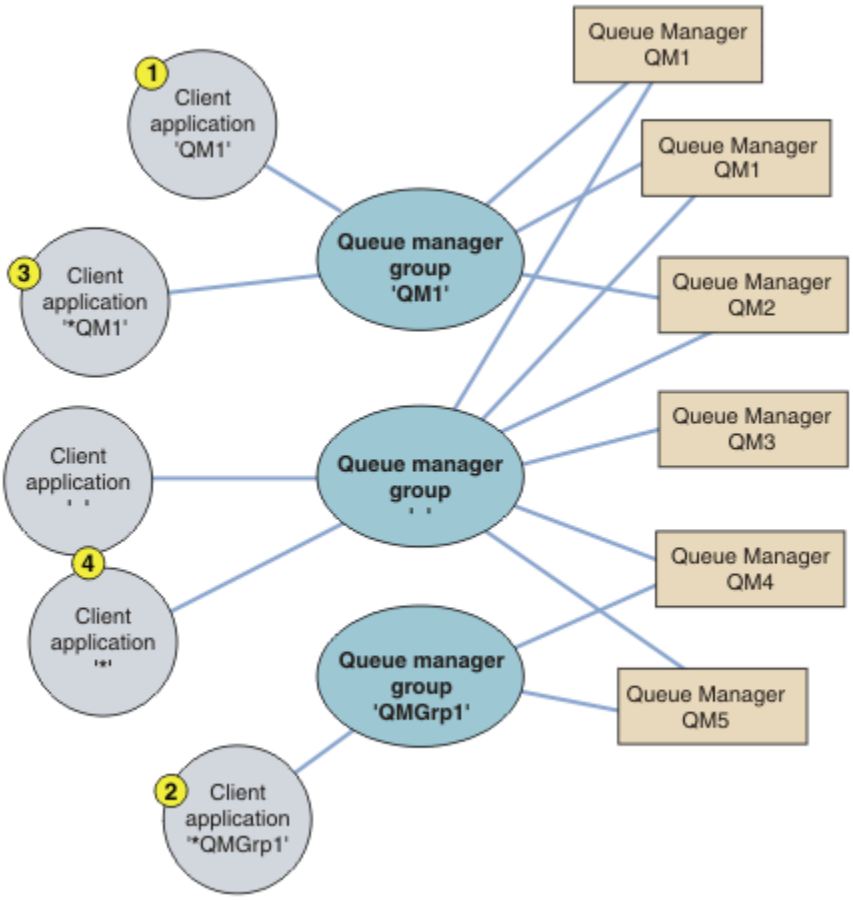


Figure 70. Groupes de gestionnaires de files d'attente

Quatre exemples d'utilisation de cette table de connexion client sont décrits à l'aide des applications client numérotées dans Figure 70, à la page 378.

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 WebSphere MQ sélectionne le groupe de gestionnaires de files d'attente correspondant, QM1. Le groupe contient trois canaux de connexion et le client WebSphere MQ MQI tente de se connecter à `QM1` en utilisant chacun de ces canaux à son tour jusqu'à ce qu'il trouve un programme d'écoute WebSphere MQ pour la connexion connectée 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 WebSphere MQ sélectionne le groupe de gestionnaires de files d'attente correspondant, QMGrp1. Ce groupe contient deux canaux de connexion client et le client WebSphere MQ MQI tente de se connecter à *tout gestionnaire de files d'attente* à l'aide de chaque canal. Dans cet exemple, le client WebSphere MQ MQI 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.

Equilibrer 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. Le client WebSphere MQ MQI rend aléatoires 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.

Vous avez le choix entre distribuer la table de connexion client à chaque client ou placer la table de définition de canal du client sur un système de fichiers partagé auquel chaque client doit faire référence. Vous pouvez également utiliser la version de programmation de la table de définition de canal du client prise en charge dans l'appel MQCONNX MQI et appeler un service pour transmettre la table de définition de canal du client à l'application 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.

Connexion à des groupes de partage de files d'attente

Vous pouvez connecter votre application à un gestionnaire de files d'attente faisant partie d'un groupe de partage de files d'attente. Pour ce faire, utilisez le nom du groupe de partage de files d'attente à la place du nom du gestionnaire de files d'attente dans l'appel MQCONN ou MQCONNX.

Le nom des groupes de partage de files d'attente comporte jusqu'à quatre caractères. Il doit être unique sur votre réseau et être différent de celui d'un gestionnaire de files d'attente.

La définition de canal du client doit utiliser l'interface générique du groupe de partage de files d'attente pour se connecter à un gestionnaire de files d'attente disponible dans le groupe. Pour plus d'informations, voir Connexion d'un client à un groupe de partage de files d'attente. Une vérification est effectuée pour s'assurer que le gestionnaire de files d'attente auquel se connecte le programme d'écoute est membre du groupe de partage de files d'attente.

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 client WebSphere MQ MQI 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 client WebSphere MQ MQI sélectionne un canal à partir d'une table de définition de canal du client, 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 client WebSphere MQ MQI. (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.

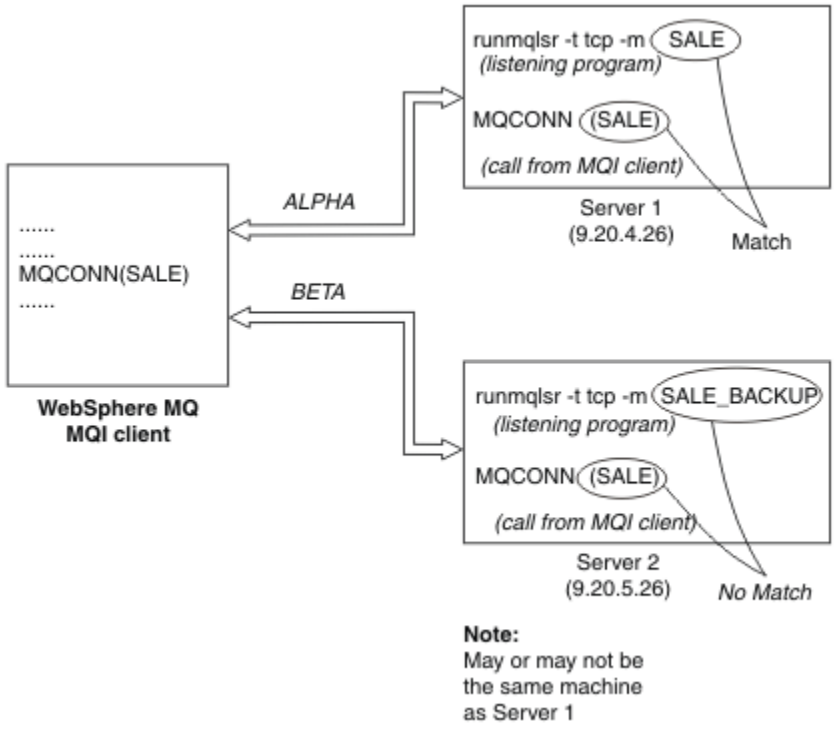


Figure 71. Exemple MQCONN

Les définitions des canaux dans ces exemples sont les suivantes:

Définitions de vente:

```
DEFINE CHANNEL(ALPHA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere MQ MQI client')

DEFINE CHANNEL(ALPHA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) DESCR('WebSphere MQ MQI client connection to server 1') +
QMNAME(SALE)

DEFINE CHANNEL(BETA) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.5.26) DESCR('WebSphere MQ MQI client connection to server 2') +
QMNAME(SALE)
```

Définition SALE_BACKUP:

```
DEFINE CHANNEL(BETA) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server connection to WebSphere 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 est effectuée-cette opération est *infructueuse* car la liaison de communication est rompue.
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 384.

Remarque : Si cette application s'exécute dans un environnement autre qu'un client WebSphere MQ MQI et 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 accessible 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 WebSphere MQ s'exécutant sur des clients WebSphere MQ MQI contribuent au déclenchement 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 dans un gestionnaire de files d'attente z/OS , les messages sont placés dans une unité de travail. 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 client Windows 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.

moniteur de déclenchement

Le moniteur de déclenchement fourni par les produits nonz/OS WebSphere MQ s'exécute dans les environnements client pour les systèmes UNIX, Linux et Windows .

Pour exécuter le moniteur de déclenchement, exécutez l'une des commandes suivantes:

- 

Sur les plateformes Windows, UNIX et Linux :

```
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, WebSphere MQ n'impose aucune restriction quant au nombre d'applications pouvant être exécutées.

Pour exécuter l'application démarrée en arrière-plan sur les systèmes UNIX 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 établir 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 CICSENV.CMD .

WebSphere MQ peuvent être exécutées dans un environnement de serveur WebSphere MQ ou sur un client WebSphere MQ sans changer de code. Toutefois, dans un environnement de serveur WebSphere 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 redéfinie en tant que client, la prise en charge des points de synchronisation est perdue. MQCMIT et MQBACK doivent être utilisés pour l'application exécutée sur un client WebSphere MQ MQI.

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 client WebSphere MQ MQI, suivez les instructions de la rubrique Configuration d'un client transactionnel étendu.

Notez toutefois que les informations qui concernent spécifiquement la préparation des applications CICS et Tuxedo, y compris les exemples de programme fournis avec WebSphere MQ, supposent que vous préparez les applications à s'exécuter sur un système serveur WebSphere MQ . Par conséquent, les informations concernent uniquement les bibliothèques WebSphere 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. Par exemple, pour les applications écrites en C sous AIX, HP-UXou Solaris, utilisez la bibliothèque libmqic au lieu de libmqm. Sur les systèmes Windows , utilisez la bibliothèque mqic.lib à la place de mqm.lib.
- Au lieu des bibliothèques système du serveur indiquées dans Tableau 48, à la page 387, pour AIX, HP-UXet Solaris, et Tableau 49, à la page 388, pour les systèmes Windows , utilisez les bibliothèques système client équivalentes. Si une bibliothèque système de serveur n'est pas répertoriée dans ces tables, utilisez la même bibliothèque sur un système client.

Tableau 48. Bibliothèques système client sous AIX, HP-UXet Solaris

Bibliothèque pour un système serveur WebSphere MQ	**Bibliothèque équivalente à utiliser sur un système client WebSphere MQ**
libmqmxa	libmqcxa

Tableau 49. Bibliothèques système client sur les systèmes Windows	
Bibliothèque pour un système serveur WebSphere MQ	**Bibliothèque équivalente à utiliser sur un système client WebSphere 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 WebSphere MQ sous CICS, elle passe normalement de l'utilisateur CICS à l'ID utilisateur de la transaction. Toutefois, lorsque vous exécutez une application client WebSphere MQ MQI sous CICS, elle conserve les droits d'accès privilégiés CICS.

Exemples de programmes CICS et Tuxedo

Exemples de programmes CICS et Tuxedo à utiliser sur les systèmes AIX, HP-UX, Solaris et Windows.

Le Tableau 50, à la page 388 répertorie les exemples de programmes CICS et Tuxedo qui sont fournis pour être utilisés sur les systèmes client AIX, HP-UXet Solaris. Le Tableau 51, à la page 388 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 123 et «Exemples TUXEDO», à la page 162.

Tableau 50. Exemples de programmes pour les systèmes client AIX, HP-UXet Solaris		
Description	**Source**	**Module exécutable**
Programme CICS	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 51. Exemples de programmes pour les systèmes client Windows		
Description	**Source**	**Module exécutable**
Transaction CICS	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	-

Tableau 51. Exemples de programmes pour les systèmes client Windows (suite)		
Description	**Source**	**Module exécutable**
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	-

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 des erreurs WebSphere MQ sont documentés dans la rubrique 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 de bibliothèques 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.

Préparation et exécution des applications Microsoft Transaction Server

Pour préparer une application MTS à s'exécuter en tant qu'application client WebSphere MQ MQI, suivez les instructions ci-dessous en fonction de votre environnement.

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

Pour préparer une application MTS à s'exécuter en tant qu'application client WebSphere MQ MQI, 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 sous Windows», à la page 477 mais liez le composant à la bibliothèque mqicxa.lib à la place de mqic.lib.
- Si le composant utilise les classes C++ WebSphere MQ , suivez les instructions de la rubrique «Génération de programmes C++ sous Windows», à la page 677 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 sous Windows», à la page 481 , mais lorsque vous définissez le projet Visual Basic, entrez `MqType=3` dans la zone **Arguments de compilation conditionnels** .
- Si le composant utilise WebSphere MQ Automation Classes for ActiveX (MQAX), définissez une variable d'environnement, GMQ_MQ_LIB, avec la valeur `mqic32xa.dll` .

  Vous pouvez définir la variable d'environnement à partir de votre application ou vous pouvez la définir de sorte que sa portée soit à l'échelle du système. Toutefois, sa définition en tant que système peut entraîner un comportement incorrect de toute application MQAX existante, qui ne définit pas la variable d'environnement à partir de l'application.

Préparation et exécution des applications JMS WebSphere MQ

Vous pouvez exécuter des applications JMS WebSphere MQ en mode client avec WebSphere Application Server comme gestionnaire de transactions. Certains messages d'avertissement peuvent s'afficher.

Pour préparer et exécuter des applications JMS WebSphere MQ en mode client, avec WebSphere Application Server comme gestionnaire de transactions, suivez les instructions de la rubrique «Utilisation des classes WebSphere MQ pour JMS», à la page 741.

Lorsque vous exécutez une application client JMS WebSphere MQ , 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 installables WebSphere MQ

Vous pouvez étendre les fonctions du gestionnaire de files d'attente à l'aide d'exits utilisateur, d'exits API ou de services optionnels. 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 «Ecriture et compilation d'exits et de services installables», à la page 390.

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

Ecriture et compilation d'exits et de services installables

Vous pouvez écrire et compiler des exits sans liaison à des bibliothèques IBM WebSphere MQ sous UNIX, Linux et Windows.

Pourquoi et quand exécuter cette tâche

Cette rubrique s'applique aux systèmes Windows, UNIX and Linux uniquement. 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 WebSphere MQ est installé dans un emplacement autre que celui par défaut, vous devez écrire et compiler vos exits sans lien vers des bibliothèques IBM WebSphere MQ .

Vous pouvez écrire et compiler des exits sur des systèmes Windows, UNIX and Linux sans lier les bibliothèques IBM WebSphere 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 UNIX and Linux , IBM WebSphere 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 406.
   - 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 416.
   - 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 WebSphere MQ for Windows», à la page 438 et «Ecriture d'un exit de conversion de données pour WebSphere MQ sur les systèmes UNIX and Linux», à la page 434.
- 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 432.

• 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 440.

• 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 442.

• 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 444.

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 «Exemples de programmes WebSphere MQ», à la page 100.

3. Compilez l'exit:
   • N'établissez pas de lien vers les bibliothèques IBM WebSphere MQ .
   • N'incluez pas de chemin RPath intégré à des bibliothèques IBM WebSphere 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 407.
     - Exits de canal, exits de publication, exits de charge de travail de cluster: «Compilation de programmes d'exit de canal sous Windows, systèmes UNIX and Linux», à la page 431.
     - Exits de conversion de données: «Ecriture des exits de conversion de données», à la page 432.
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 WebSphere MQ Explorer 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 WebSphere MQ nommé dans le chemin. Par exemple, l'exit est placé dans /exits/installation2 et dans /exits, mais pas dans /exits/installation1. L' WebSphere MQ installation2 utilise l'exit de /exits/installation2. L' WebSphere MQ installation1 utilise l'exit du répertoire /exits .

5. Si nécessaire, configurez l'exit:
   - Services installables: «Configuration des services et des composants», à la page 401.
   - Exits API: «Configuration des exits API», à la page 411.
   - Exits de canal: «Configuration des exits de canal», à la page 432.
   - Exits de publication: «Configuration des exits de publication», à la page 443.
   - Exits de préconnexion: «Section PreConnect du fichier de configuration du client», à la page 441.

Services et composants installables pour UNIX, Linux et 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 WebSphere MQ sont les suivantes:

- Pour vous permettre de choisir d'utiliser les composants fournis par les produits WebSphere MQ , ou de les remplacer ou de les augmenter par d'autres.
- Permettre aux fournisseurs de participer, en fournissant des composants pouvant utiliser de nouvelles technologies, sans apporter de modifications internes aux produits WebSphere MQ .
- Permettre à WebSphere MQ d'exploiter les nouvelles technologies plus rapidement et à moindre coût, et donc de fournir des produits plus tôt et à des prix plus bas.

Les *services installables* et les *composants de service* font partie de la structure de produit WebSphere 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 éditeurs de logiciels indépendants et à d'autres tiers de fournir des composants installables pour augmenter ou remplacer ceux fournis par les produits

WebSphere MQ . Le Tableau 52, à la page 394 récapitule les services et les composants qui peuvent être utilisés.

Tableau 52. Récapitulatif des composants de service installables

fonction installable	Composant fourni	Function	Conditions requises
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 **Remarque :** L'attribut *Scope* des files d'attente partagées doit être défini sur CELL.	• 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.

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 WebSphere 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 398. 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.

La Figure 72, à la page 395 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.

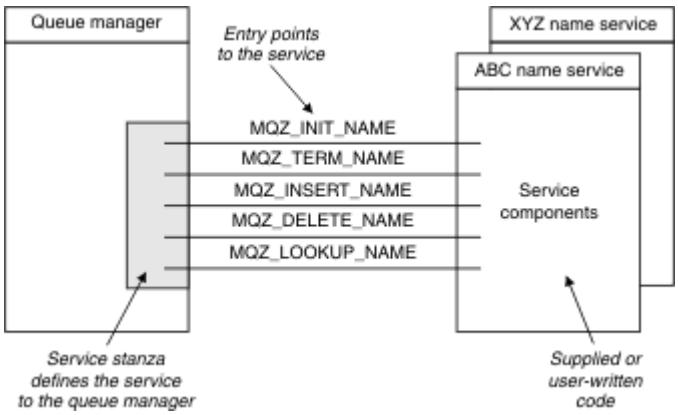


Figure 72. Description des services, des composants et des points d'entrée

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 UNIX 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\MQSeries\CurrentVersion\Configuration\QueueManager\`*qmname*`\` où *qmname* est le nom du gestionnaire de files d'attente.

Pour les systèmes UNIX and Linux , les composants de service sont démarrés dans l'ordre dans lequel ils sont définis dans le fichier qm.ini . Sous Windows, étant donné que le registre Windows est utilisé, WebSphere 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 WebSphere 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 Les commandes de contrôle.

La méthode d'accès aux objets (OAM) fonctionne avec l' *entité* d'un principal ou d'un groupe.

- Sur les systèmes UNIX and Linux :
  - le principal est un ID utilisateur ou un ID associé à un programme d'application exécuté pour le compte d'un utilisateur.
  - le groupe est une collection de principaux UNIX ou Linux définie par le système.
  - 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.
- Sur les systèmes Windows :
  - le principal est un ID utilisateur Windows ou un ID associé à un programme d'application exécuté pour le compte d'un utilisateur.
  - le 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 l'autorisation de l'entité associée à l'opération pour voir si elle peut:

- 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.

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 distantes ***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 WebSphere 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 WebSphere 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 permettant d'insérer ou de supprimer une file d'attente dans un répertoire de composant sont gérés par le composant `ABC_name_serv`; il s'agit du 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 53, à la page 401 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. Les 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 53, à la page 401). 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 53. 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

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.

1. 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 indiquez l'emplacement du module.

   Chaque service utilisé doit comporter une section *`Service`* qui définit le service dans le gestionnaire de files d'attente.

   Pour chaque composant d'un service, il doit y avoir une section *`ServiceComponent`* . Identifie le nom et le chemin du module contenant le code de ce composant.

   Pour plus d'informations, voir «Format de section de service», à la page 402 et «Format de section de composant de service», à la page 402

   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.

Format de section de service
La section Service contient le nom du service et le nombre de points d'entrée définis pour le service.

Le format de la section est le suivant:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

où :

<service_name>
Nom du service. Ceci est défini par le service.

<entries>
Nombre de points d'entrée définis pour le service. Cela inclut les points d'entrée d'initialisation et de fin.

Format de section de service pour les systèmes Windows
Sur les systèmes Windows , la section *Service* inclut un attribut SecurityPolicy .

Le format de la strophe est le suivant:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
  SecurityPolicy=<policy>
```

où :

<service_name>
Nom du service. Ceci est défini par le service.

<entries>
Nombre de points d'entrée définis pour le service. Cela inclut les points d'entrée d'initialisation et de fin.

<policy>
NTSIDsRequired (identificateur de sécurité Windows) ou Default. Si vous ne spécifiez pas NTSIDsRequired, la valeur Default est utilisée. Cet attribut est valide uniquement si Name a la valeur AuthorizationService.

Voir aussi «Configuration des sections de service d'autorisation: systèmes Windows», à la page 403.

Format de section de composant de service

Le format de la strophe de composant de service est le suivant:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

où :

<service_name>
Nom du service. Cette valeur doit correspondre à la valeur Name spécifiée dans une section de service.

<component_name>
Nom descriptif du composant de service. Il doit être unique et contenir uniquement les caractères valides pour les noms des objets WebSphere MQ (par exemple, les noms de file d'attente). Ce nom

apparaît dans les messages de l'opérateur générés par le service. Nous vous recommandons d'utiliser un nom commençant par une marque de la société ou une chaîne distinctive similaire.

<module_name>
Nom du module devant contenir le code de ce composant.

<size>
Taille en octets de la zone de données de composant transmise au composant lors de chaque appel. Indiquez zéro si aucune donnée de composant n'est requise.

Ces deux strophes peuvent se produire dans n'importe quel ordre et les clés de strophe sous elles peuvent également se produire dans n'importe quel ordre. Pour l'une ou l'autre de ces sections, toutes les clés de section doivent être présentes. Si une clé de section est dupliquée, la dernière est utilisée.

Au démarrage, le gestionnaire de files d'attente traite chaque entrée de composant de service dans le fichier de configuration. Il charge ensuite le module de composant spécifié, en appelant le point d'entrée du composant (qui doit être le point d'entrée pour l'initialisation du composant), en lui transmettant un descripteur de configuration.

Configuration des sections de service d'autorisation: systèmes UNIX and Linux
Sur les systèmes UNIX and Linux , 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 `/var/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 `mqsnoaut`. 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 sur WebSphere MQ for AIX. *`MQ_INSTALLATION_PATH`* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

```
Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=MQ_INSTALLATION_PATH/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

Figure 73. UNIX and Linux sections de service d'autorisation dans qm.ini

La strophe de composant de service (`MQSeries.UNIX.auth.service`) définit le composant de service d'autorisation par défaut, la méthode d'accès aux objets (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.

Configuration des sections de service d'autorisation: systèmes Windows
Sous WebSphere MQ for Windows , chaque gestionnaire de files d'attente possède sa propre section dans le registre.

La section *`Service`* et la section *`ServiceComponent`* du composant d'autorisation par défaut sont ajoutées automatiquement au registre, mais peuvent être remplacées à l'aide de `mqsnoaut`. Toute autre strophe *`ServiceComponent`* doit être ajoutée manuellement.

Vous pouvez également ajouter l'attribut `SecurityPolicy` à l'aide des services WebSphere MQ . L'attribut `SsecurityPolicy` s'applique uniquement si le service spécifié dans la section *`Service`* est le service d'autorisation, c'est-à-dire le service OAM par défaut. L'attribut `SecurityPolicy` permet de spécifier la règle de sécurité pour chaque gestionnaire de files d'attente. Les valeurs possibles sont les suivantes:

`Default`
Indiquez `Default` si vous souhaitez que la stratégie de sécurité par défaut soit appliquée. Si un ID de sécurité Windows (ID de sécurité NT) n'est pas transmis à la méthode d'accès aux objets (OAM) pour un ID utilisateur particulier, une tentative est effectuée pour obtenir l'ID de sécurité approprié en effectuant une recherche dans les bases de données de sécurité appropriées.

`NTSIDsRequired`
Requiert qu'un SID NT soit transmis à la méthode d'accès aux objets (OAM) lors de l'exécution de contrôles de sécurité.

Pour plus d'informations sur le format de la section Service, voir «Format de section de service pour les systèmes Windows», à la page 402. Pour plus d'informations générales sur la sécurité, voir Configuration de la sécurité sur les systèmes Windows, UNIX and Linux.

La section de composant de service, `MQSeries.WindowsNT.auth.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.

Configuration des sections de service annuaire: systèmes Unix et Linux
Insérez une brève description ici ; utilisée pour le premier paragraphe et le résumé.

Les exemples suivants de sections de fichier de configuration UNIX and Linux pour le service annuaire indiquent un composant de service annuaire fourni par la société ABC (fictive).

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=/usr/lib/abcname
    ComponentDataSize=1024
```

Figure 74. Sections de service de nom dans qm.ini (pour les systèmes UNIX and Linux)

Remarque : Sur les systèmes Windows , les informations de section de service annuaire sont stockées dans le registre.

Actualisation de la méthode d'accès aux objets (OAM) après la modification de l'autorisation d'un utilisateur
Dans WebSphere MQ, vous pouvez actualiser les informations du groupe d'autorisations de la méthode d'accès aux objets (OAM) 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

Ecriture et compilation des exits API

Les exits API vous permettre d'écrire un code modifiant le comportement des appels API WebSphere MQ, tels que MQPUT et MQGET, puis d'insérer ce code immédiatement avant ou après ces appels.

Remarque : Non pris en charge sur WebSphere 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:

- Pour la *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.
- Pour une *flexibilité*, vous pouvez réagir aux changements rapides de votre environnement métier sans modifier 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.
- Pour la *surveillance* de 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 WebSphere MQ, le gestionnaire de files d'attente appelle automatiquement votre code d'exit aux points enregistrés.

Les routines d'exit API à exécuter sont identifiées dans des strophes sur les systèmes IBM i, Windows, UNIX and Linux . 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. ApiExitcommun, dans le fichier mqs.ini , identifie les routines, pour l'ensemble de WebSphere 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 405 dans cette liste).
2. Le modèle ApiExit, dans le fichier mqs.ini , identifie les routines, pour l'ensemble de WebSphere MQ, copiées dans l'ensemble ApiExitLocal (voir l'élément «3», à la page 405 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 411 , détermine l'ordre dans lequel les routines définies dans les sections s'exécutent.

Utilisation des exits API dans plusieurs installations de WebSphere MQ

Assurez-vous que les exits d'API écrits pour la version antérieure de WebSphere MQ sont utilisés avec toutes les versions car les modifications apportées aux exits dans la version 7.1 risquent de ne pas fonctionner avec une version antérieure. Pour plus d'informations sur les modifications apportées aux exits, voir «Ecriture et compilation d'exits et de services installables», à la page 390.

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 WebSphere 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.

Ecriture des exits API

Vous pouvez écrire des exits pour chaque appel d'API à l'aide du langage de programmation C.

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 WebSphere MQ pour un accès ultérieur
- MQCLOSE, pour fermer une ressource WebSphere MQ précédemment ouverte pour l'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 consulter les attributs d'une ressource WebSphere MQ 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.

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 118.

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 et compilation d'exits et de services installables», à la page 390.

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 WebSphere 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, le descripteur ExitMsgest effacé des propriétés et renseigné 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.

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 118. 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. 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 WebSphere MQ a été installé.

Remarque aux utilisateurs :

1. Les conseils relatifs à la programmation des applications 64 bits sont répertoriés dans la rubrique Normes de codage sur les plateformes 64 bits

Avec l'introduction des clients de multidiffusion, les exits API et les exits de conversion de données doivent pouvoir s'exécuter côté client car certains messages peuvent ne pas passer par le gestionnaire de files d'attente. Les bibliothèques suivantes font désormais partie des packages client ainsi que des packages serveur:

Tableau 54. Bibliothèques qui se trouvent désormais dans les packages client et serveur	
Système d'exploitation	**Bibliothèques**
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb
Linux & HP-UX	32 bits & 64 bits: libmqm.so & libmqm_r.so
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
Solaris	32 bits & 64 bits: libmqm.so

Compilation des exits API sur les systèmes Unix et Linux
Exemples de compilation des exits API sur les systèmes UNIX et Linux .

Sur toutes les plateformes, le point d'entrée du module est MQStart.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Sous 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 -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits/amqsaxe_r \
    amqsaxe0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -q64 -e MQStart -bE:amqsaxe.exp -bM:SRE -o /var/mqm/exits64/amqsaxe_r \
    amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Sur la plateforme HP-UX Itanium

Applications 32 bits

Non unités d'exécution

Compilez le code source de l'exit d'API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Lier le code source de l'exit API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe
rm amqsaxe.o
```

Unité d'exécution

Compilez le code source de l'exit d'API:

```
c89 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Lier le code source de l'exit API

```
ld +b: -b amqsaxe.o +ee MQStart -o /var/mqm/exits/amqsaxe_r
rm amqsaxe.o
```

Applications 64 bits

Non unités d'exécution

Compilez le code source de l'exit d'API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Lier le code source de l'exit API

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe
rm amqsaxe.o
```

Unité d'exécution

Compilez le code source de l'exit d'API:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o amqsaxe.o amqsaxe0.c -IMQ_INSTALLATION_PATH/inc
```

Lier le code source de l'exit API

```
ld -b amqsaxe.o +ee MQStart -o  /var/mqm/exits64/amqsaxe_r
rm amqsaxe.o
```

ActivéLinux

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 \
     -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

Applications 32 bits

Non unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe amqsaxe0.c \
     -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m64 -shared -fPIC -o /var/mqm/exits64/amqsaxe_r amqsaxe0.c \
    -IMQ_INSTALLATION_PATH/inc
```

Sous Solaris

Compilez le code source de l'exit API en exécutant l'une des commandes suivantes:

Applications 32 bits

Plateforme SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

plateformex86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Applications 64 bits

Plateforme SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

plateformex86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -IMQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

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 75, à la page 410 ou Figure 76, à la page 411 pour compiler et lier `amqsaxe0.c` sous Windows . Les commandes fonctionnent avec Microsoft Visual Studio 2005, 2008 ou 2010. Les exemples supposent que le répertoire WebSphere MQ `C:\Program Files\IBM\WebSphere 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 75. Compilez et liez `amqsaxe0.c` sous 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 76. Compilez et liez amqsaxe0.c sous Windows 64 bits

Concepts associés

«Exemple de programme d'exit API», à la page 118
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.

Configuration des exits API

Vous configurez IBM WebSphere 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:

- Utilisation de IBM WebSphere MQ Explorer (sous Windows et Linux (plateformesx86 et x86-64))
- Utilisation de la commande **amqmdain** (sous Windows)
- Utilisation des fichiers mqs.ini et qm.ini directement (sur les systèmes Windows, UNIX and Linux).

  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 le répertoire `/var/mqm` sur UNIX and Linux et dans WorkPath spécifié dans la clé `HKLM\SOFTWARE\IBM\WebSphere MQ` sur les systèmes Windows .

  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:

  Sur les systèmes UNIX and Linux :

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Sur les systèmes Windows :

  ```
  C:\Program Files\IBM\WebSphere 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 WebSphere MQ Explorer sur la page de propriétés IBM WebSphere 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 WebSphere MQ Explorer sur la page de propriétés IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 432.

Certaines de ces informations s'appliquent également aux exits sur les canaux MQI, qui connectent les clients MQI WebSphere MQ 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 55, à la page 414 présente les types d'exit de canal disponibles pour chaque type de canal.

Tableau 55. 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

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 77, à la page 415.

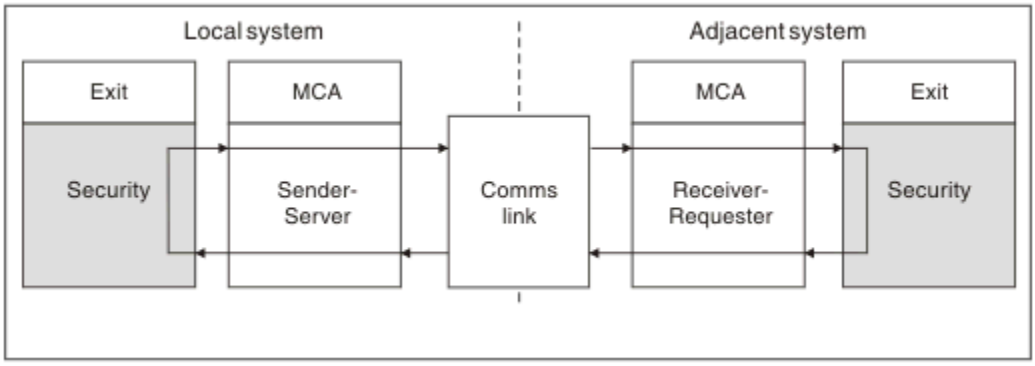


Figure 77. Boucle d'exit de sécurité

Lors 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 78, à la page 415.

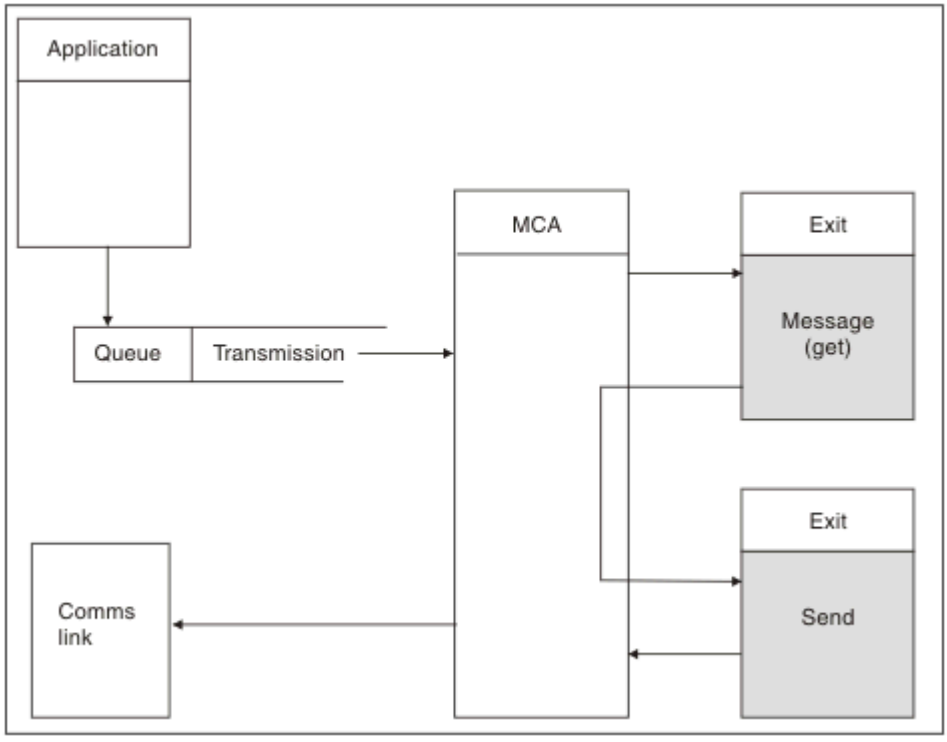


Figure 78. Exemple d'exit d'émission à l'extrémité émettrice du canal de transmission de messages

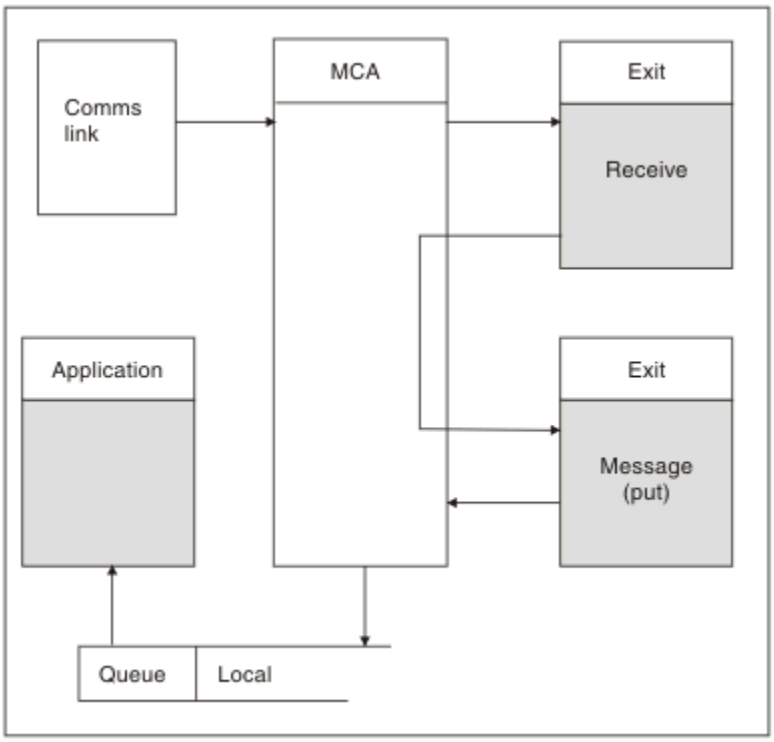


Figure 79. Exemple d'exit de réception à l'extrémité réceptrice du canal de transmission de messages

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 79, à la page 416. (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 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 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 WebSphere 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 de la structure de données MQCD, la nouvelle valeur est ignorée par le processus de canal IBM WebSphere 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:
  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 la longueur MaxSegment, 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 413 .

Ecriture de programmes d'exit de canal sous Windows, systèmes UNIX and Linux
Vous pouvez utiliser les informations suivantes pour vous aider à écrire des programmes d'exit de canal pour les systèmes Windows, UNIX and Linux .

Suivez les instructions décrites dans «Ecriture et compilation d'exits et de services installables», à la page 390. Utilisez les informations spécifiques à l'exit de canal suivantes, le cas échéant:

L'exit doit être écrit en C et est une DLL sous Windows.

Définissez une routine MQStart () factice dans l'exit et spécifiez `MQStart` comme point d'entrée dans la bibliothèque. Figure 80, à la page 418 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 80. 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 81, à la page 418. Pour plus d'informations sur l'écriture de programmes d'exit de canal, voir «Ecriture de programmes d'exit de canal», à la page 416.

```
EXPORTS
ChannelExit
```

Figure 81. Exemple de fichier DEF pour Windows

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 WebSphere MQ . La prise en charge de SSL et TLS est également fournie par IBM WebSphere MQ pour authentifier vos utilisateurs et fournir des contrôles de chiffrement et d'intégrité des données pour vos données. Pour plus d'informations sur SSL et TLS, voir WebSphere MQ support for SSL and TLS. 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.

Pour les exits de sécurité écrits avant IBM WebSphere MQ Version 7.1 , il convient de noter que les versions antérieures de IBM WebSphere MQ ont interrogé le fournisseur de sockets sécurisés sous-jacent (par exemple GSKit) afin de déterminer le nom distinctif du sujet du certificat du partenaire distant (SSLPEER) et le nom distinctif de l'émetteur (SSLCERTI). Dans IBM WebSphere MQ Version 7.1 , la prise en charge a été ajoutée pour une série de nouveaux attributs de sécurité. Pour accéder à ces attributs, IBM WebSphere MQ Version 7.1 obtient le codage DER du certificat et l'utilise pour déterminer le nom distinctif du sujet et de l'émetteur. 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

Dans IBM WebSphere MQ Version 7.1, un attribut SERIALNUMBER est également inclus dans le nom distinctif du sujet et contient le numéro de série du certificat du partenaire distant. Certains attributs de nom distinctif sont également renvoyés dans une séquence différente de celle des versions précédentes. Par conséquent, la composition des zones SSLPEER et SSLCERTI est modifiée dans Version 7.1 à partir des éditions précédentes et il est donc recommandé d'examiner et de mettre à jour les exits de sécurité ou les applications dépendant de ces zones.

Les filtres de nom d'homologue WebSphere MQ existants spécifiés via la zone SSLPEER d'une définition de canal ne sont pas affectés et continueront de fonctionner de la même manière que dans les versions précédentes. En effet, l'algorithme de correspondance de nom d'homologue WebSphere MQ a été mis à jour pour traiter les filtres SSLPEER existants sans qu'il soit nécessaire de modifier les définitions de canal. Cette modification est susceptible d'affecter les exits de sécurité et les applications qui dépendent des valeurs de nom distinctif de sujet et de nom distinctif d'émetteur renvoyées par l'interface de programmation PCF.

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 82, à la page 420 à Figure 85, à la page 422. 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 82, à la page 420:

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 82. 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 83. Echange initié par l'expéditeur sans 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_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figure 84. Echange initié par le récepteur 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_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

Figure 85. 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. Dans IBM i, les programmes d'exit de sécurité qui ont été compilés avec "Utiliser les droits adoptés" (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 SSL 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 SSL.
   - Pour modifier MQCD.MCAUserIdentifier basé sur le nom.

Concepts associés

Enregistrements d'authentification de canal

Concepts SSL (Secure Sockets Layer) et TLS (Transport Layer Security)

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 86, à la page 423 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 86. Code squelette de l'exit de sécurité

Le MQStart de point d'entrée WebSphere MQ 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
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
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
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
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
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
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
WebSphere MQ for Windows fournit un exit de sécurité qui fournit l'authentification pour les canaux WebSphere MQ à l'aide de l'interface SSPI (Security Services Programming Interface). SSPI fournit les fonctions de sécurité intégrées de Windows.

Cet exit de sécurité est destiné au client WebSphere MQ et au serveur WebSphere 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 sous Windows, à l'aide des services d'authentification NTLM. L'authentification bidirectionnelle est fournie sous 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 sur les systèmes Windows», à la page 176

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 sur 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.

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. WebSphere 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.

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 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 indiqué dans MQXQH.) Si vous utilisez des messages de référence, c'est-à-dire des messages qui contiennent uniquement 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 et 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», à la page 277 et «Exécution des exemples de message de référence», à la page 147 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. Sur les systèmes UNIX , si vous fournissez un exit de message pour une raison quelconque, la conversion automatique des ID utilisateur en caractères minuscules ne fonctionne pas. Voir Sécurité des objets sur les systèmes UNIX and Linux.
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 passés 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, il est possible d'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 WebSphere 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 WebSphere 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 WebSphere 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'aucune définition de canal n'existe. 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 WebSphere 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, Les zones de données SendUser, ReceiveExit et ReceiveUserelles-mêmes.

Pour plus d'informations, voir Définition automatique des canaux.

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.

Compilation de programmes d'exit de canal sous Windows, systèmes UNIX and Linux

Utilisez les exemples suivants pour vous aider à compiler des programmes d'exit de canal pour les systèmes Windows, UNIX and Linux .

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 UNIX et Linux

Linux UNIX

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 .

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
```

Exemples de commandes du compilateur et de l'éditeur de liens pour les exits de canal sous HP-UX

```
$ c89 +DD64 +z -c -D_HPUX_SOURCE -o exit.o exit.c -I/opt/mqm/inc
$ ld -b exit.o +ee MQStart +ee ChannelExit -o
```

```
/var/mqm/exits64/exit -L/usr/lib/pa20_64 -lpthread
$ rm exit.o
```

Exemples de commandes de compilateur et d'éditeur de liens pour les exits de canal sur les plateformes 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
```

Exemples de commandes du compilateur et de l'éditeur de liens pour les exits de canal sur les plateformes 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
```

Exemples de commandes du compilateur et de l'éditeur de liens pour les exits de canal sous Solaris:

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

Sur le client, un exit 32 bits ou 64 bits peut être utilisé. Cet exit doit être lié à mqic_r.

Sous AIX, toutes les fonctions appelées par IBM WebSphere 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 WebSphere MQ ; pour plus d'informations, voir MQCD ou Commandes Script (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 WebSphere 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 WebSphere 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 WebSphere 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 (DCI) WebSphere MQ .

Avec l'introduction des clients de multidiffusion, les exits API et les exits de conversion de données doivent pouvoir s'exécuter côté client car certains messages peuvent ne pas passer par le gestionnaire de files d'attente. Les bibliothèques suivantes font désormais partie des packages client ainsi que des packages serveur:

Tableau 56. Bibliothèques qui se trouvent désormais dans les packages client et serveur

Système d'exploitation	**Bibliothèques**
Windows	32 bits & 64 bits: mqm.dll & mqm.pdb
Linux & HP-UX	32 bits & 64 bits: libmqm.so & libmqm_r.so
AIX	32 bits & 64 bits: libmqm.a & libmqm_r.a
Solaris	32 bits & 64 bits: libmqm.so

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 conditions suivantes sont remplies:

- 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.

Ecriture d'un exit de conversion de données pour WebSphere MQ sur les systèmes UNIX and Linux

Informations sur les étapes à prendre en compte lors de l'écriture de programmes d'exit de conversion de données pour WebSphere MQ sur les systèmes UNIX and 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 434. Le fichier source squelette et la copie sont en lecture seule.

   Le fichier source squelette est appelé amqsvfc0.c.
5. Sous WebSphere MQ for AIX, un fichier d'exportation 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 WebSphere 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 434.
   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 435. 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 434 ci-dessus.
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 UNIX and Linux», à la page 435.
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 WebSphere 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.

Compilation des exits de conversion de données sur les systèmes UNIX and Linux
Exemples de compilation d'un exit de conversion de données sur les systèmes UNIX 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 WebSphere MQ est installé.

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 -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
xlc_r -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits/MYFORMAT_r \
     MYFORMAT.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
 xlc_r -q64 -e MQStart -bE:MYFORMAT.exp -bM:SRE -o /var/mqm/exits64/MYFORMAT_r \
     MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Plateforme HP-UX Itanium

Compilez et liez le code source d'exit en émettant l'un des ensembles de commandes suivants:

Applications 32 bits

Non unités d'exécution

Compilez le code source de l'exit:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Liez l'objet d'exit:

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT -L/usr/lib/hpux32
rm MYFORMAT.o
```

Unité d'exécution

Compilez le code source de l'exit:

```
c89 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Liez l'objet d'exit:

```
ld +b: -b MYFORMAT.o +ee MQStart -o  \
    /var/mqm/exits/MYFORMAT_r -L/usr/lib/hpux32 \
    -lpthread
rm MYFORMAT.o
```

Applications 64 bits

Non unités d'exécution

Compilez le code source de l'exit:

```
 c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Liez l'objet d'exit:

```
ld -b MYFORMAT.o +ee MQStart \
    -o  /var/mqm/exits64/MYFORMAT \
    -L/usr/lib/hpux64
rm MYFORMAT.o
```

Unité d'exécution

Compilez le code source de l'exit:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o MYFORMAT.o MYFORMAT.c -IMQ_INSTALLATION_PATH/inc
```

Liez l'objet d'exit:

```
ld -b MYFORMAT.o +ee MQStart  \
    -o  /var/mqm/exits64/MYFORMAT_r  \
    -L/usr/lib/hpux64 -lpthread
rm MYFORMAT.o
```

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 \
      -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m31 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Applications 32 bits

Non unités d'exécution

```
 gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT MYFORMAT.c
      -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/MYFORMAT_r MYFORMAT.c
     -IMQ_INSTALLATION_PATH/inc
```

Applications 64 bits

Non unités d'exécution

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT MYFORMAT.c
      -IMQ_INSTALLATION_PATH/inc
```

Unité d'exécution

```
 gcc -m64 -shared -fPIC -o /var/mqm/exits64/MYFORMAT_r MYFORMAT.c
      -IMQ_INSTALLATION_PATH/inc
```

Solaris

Compilez le code source de l'exit à l'aide de l'une des commandes suivantes:

Applications 32 bits

Plateforme SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

plateformex86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Applications 64 bits

Plateforme SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

plateformex86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -IMQ_INSTALLATION_PATH/inc -R/usr/lib/64  -lsocket -lnsl -ldl
```

Ecriture d'un exit de conversion de données pour WebSphere MQ for Windows

Informations sur les étapes à prendre en compte lors de l'écriture de programmes d'exit de conversion de données pour WebSphere 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 WebSphere 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 438.

   amqsvfc0.c se trouve dans *MQ_INSTALLATION_PATH*\Tools\C\Samples où *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ est installé. (Le répertoire d'installation par défaut est `C:\Program Files\IBM\WebSphere 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 438.

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 438. 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 438.

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 WebSphere 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 d'exit sous le répertoire de données WebSphere 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\MQSeries\CurrentVersion\Configuration\ClientExitPa
th\
```

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 WebSphere 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.

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 Version 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 WebSphere 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 WebSphere MQ , cela permet également de trouver la bibliothèque appropriée sur un système distant.

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 d'interception d'API	32 bits	32 bits et 64 bits
Exit de conversion de données	32 bits	64 bits
Exits de canal serveur (tous types)	32 bits	32 bits
Exits de canal client (tous types)	32 bits	64 bits
Exit de service installable	32 bits	32 bits
Module de trace de service	32 bits	32 bits et 64 bits
Exit WLM de cluster	32 bits	32 bits
Exit de routage de publication / d'abonnement	32 bits	32 bits
Fichiers de commutation de chargement de base de données	32 bits	32 bits et 64 bits
Bibliothèques AX du gestionnaire de transactions externe	32 bits	64 bits

Référencement des définitions de connexion à l'aide d'un exit de préconnexion à partir d'un référentiel

Les clients WebSphere MQ MQI peuvent être configurés 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 référençant le fichier CCDT, une approche flexible consiste à stocker les définitions de client dans un référentiel global tel qu'un annuaire LDAP, un registre et un référentiel WebSphere 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 WebSphere 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

pExitParamètres

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.

CodepComp
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*/
```

Section PreConnect du fichier de configuration du client
Utilisez la section PreConnect pour configurer l'exit PreConnect dans le fichier `mqclient.ini` .

Les attributs suivants peuvent être inclus dans la section PreConnect :

Data=*< URL>*
URL du référentiel dans lequel les définitions de connexion sont stockées. Par exemple, lors de l'utilisation d'un serveur LDAP:

Données = ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com

Function=*<myFunc>*
Nom du point d'entrée fonctionnel dans la bibliothèque qui contient le code d'exit PreConnect .

La définition de fonction est conforme au prototype d'exit PreConnect MQ_PRECONNECT_EXIT.

La longueur maximale de cette zone est MQ_EXIT_NAME_LENGTH.

Module=*< amqldapi>*
Nom du module contenant le code d'exit d'API.

Si cette zone contient le nom de chemin d'accès complet du module, elle est utilisée telle qu'elle est.

Sequence=*< numéro_séquence>*
Séquence dans laquelle cet exit est appelé par rapport à d'autres exits. 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 séquentielle des sorties soit continue ; une séquence de 1, 2, 3 a le même résultat qu'une séquence de 7, 42, 1096. Cet attribut est une valeur numérique non signée.

Plusieurs sections PreConnect peuvent être définies dans le fichier `mqclient.ini` . L'ordre de traitement de chaque exit est déterminé par l'attribut Séquence de la section.

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.

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 et compilation d'exits et de services installables», à la page 390pour 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.

WebSphere 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 WebSphere 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 WebSphere 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 et compilation d'exits et de services installables», à la page 390.

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 WebSphere 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 UNIX et Linux , créez une strophe appelée `PublishSubscribe`. La section `PublishSubscribe` possède les attributs suivants:

PublishExitPath=[path]|*module_name*
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=*function_name*
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`.

PublishExitData=*string*
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, WebSphere 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 alors que le message est en cours de transfert et que le message est annulé, un nouveau gestionnaire de files d'attente cible est sélectionné.

Sur les plateformes autres que z/OS, 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

Pour les plateformes autres que z/OS, 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 et compilation d'exits et de services installables», à la page 390et utilisez l'exemple de programme «Exemple d'exit de charge de travail de cluster», à la page 445 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 416.

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)
```

Exemple d'exit de charge de travail de cluster

WebSphere 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.

Sur les plateformes autres que z/OS

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 57. Exemple d'emplacement de programme d'exit de charge de travail de cluster (non z/OS)

Plateforme	**Chemin d'accès au fichier**
AIX, HP-UX, Sun Solaris	*MQ_INSTALLATION_PATH*/samp
Fenêtres	*MQ_INSTALLATION_PATH*\Tools\c\Samples

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere 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 :

Sur les systèmes UNIX and Linux :

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

Sous Windows :

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

Désormais, au lieu d'utiliser l'algorithme de gestion de charge de travail fourni, WebSphere MQ appelle cet exit pour acheminer tous les messages vers le gestionnaire de files d'attente de votre choix.

Génération d'une application IBM WebSphere MQ

Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

Génération de votre application sous 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ», à la page 84 .

Lorsque vous exécutez des applications serveur à unités d'exécution ou client à unités d'exécution, définissez la variable d'environnement AIXTHREAD_SCOPE = S.

Préparation des programmes C sous 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 Normes de codage sur les plateformes 64 bits.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Pour les applications 32 bits:

```
  $ xlc_r -o amqsput_32 amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_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 WebSphere MQ soient résolus lors de la liaison des bibliothèques.

Si vous souhaitez utiliser les programmes sur une machine sur laquelle seul le client WebSphere MQ MQI pour AIX est installé, recompilez les programmes pour les lier à la bibliothèque client (`-lmqic`).

Liaison de bibliothèques

Vous avez besoin des bibliothèques suivantes:

- Liez vos programmes à la bibliothèque appropriée fournie par WebSphere 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 WebSphere MQ à unités d'exécution simples à partir d'une unité de compilation unique, exécutez les commandes suivantes.

  Pour les applications 32 bits:

```
$ xlc_r -o amqsputc_32_r amqsput0.c -IMQ_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 -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64
-lmqm_r
```

  où `amqsput0` est un exemple de programme.

  Si vous souhaitez utiliser les programmes sur une machine sur laquelle seul le client WebSphere MQ MQI pour AIX est installé, recompilez les programmes pour les lier à la bibliothèque client (`-lmqic`).

  Remarque :

  1. Si vous écrivez un service installable (voir Administration 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.
  2. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA, tel que IBM TXSeries, Encinaou BEA Tuxedo, vous devez établir un lien vers les bibliothèques `libmqmxa.a` (ou `libmqmxa64.a` si votre gestionnaire de transactions traite le type'long'comme étant 64 bits) et `libmqz.a` dans une application non à unités d'exécution et vers les bibliothèques `libmqmxa_r.a` (ou `libmqmxa64_r.a`) et `libmqz_r.a` dans une application à unités d'exécution.
  3. Vous devez lier des applications sécurisées aux bibliothèques WebSphere MQ à unités d'exécution. Toutefois, une seule unité d'exécution dans une application sécurisée sur les systèmes WebSphere MQ on UNIX and Linux peut être connectée à la fois.
  4. Vous devez lier les bibliothèques WebSphere MQ avant les autres bibliothèques de produit.

Préparation de programmes COBOL dans AIX

Utilisez ces informations lors de la préparation de programmes COBOL dans AIX à l'aide d' IBM COBOL Set et Micro Focus COBOL.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere 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 WebSphere 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 \
-I<COBCPY>
```

Application client non à unités d'exécution 32 bits

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-I<COBCPY>
```

Application de serveur à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

Application client à unités d'exécution 32 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

Application serveur 64 bits sans unités d'exécution

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -I<COBCPY>
```

Application client 64 bits sans unités d'exécution

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -I<COBCPY>
```

Application serveur à unités d'exécution 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -I<COBCPY>
```

Application client à unités d'exécution 64 bits

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -I<COBCPY>
```

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>
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.

Préparation des programmes d'application CICS sous AIX

Utilisez ces informations lors de la préparation des programmes CICS dans AIX.

Des modules de commutation XA sont fournis pour vous permettre de lier CICS à IBM WebSphere MQ:

Tableau 58. 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 WebSphere MQ *amqzsc* , qui est fourni avec le produit.

Liez toujours vos transactions C à la bibliothèque IBM WebSphere MQ autorisant les unités d'exécution multiples *libmqm_r.a.*, et vos transactions COBOL avec la bibliothèque COBOL *libmqmcb_r.a.*.

Pour plus d'informations sur la prise en charge des transactions CICS , voir Administration.

Prise en charge de TXSeries CICS

IBM WebSphere MQ on 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 WebSphere 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 WebSphere MQ qui sont chargés dans la même région CICS en langage 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 CICS COBOL à l'aide de IBM COBOL Set for AIX

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere 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  \
                   -IMQ_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 WebSphere MQ est installé.

Pour utiliser Micro Focus COBOL, procédez comme suit:

1. Ajoutez le module de bibliothèque d'exécution COBOL IBM WebSphere MQ à la bibliothèque d'exécution à l'aide de la commande suivante:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Remarque : Avec `cicsmkcobol`, IBM WebSphere 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 WebSphere 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 IBM WebSphere MQ sur les systèmes UNIX and 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 WebSphere 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 WebSphere 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 WebSphere 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. Les échecs sont envoyés à la file d'attente SYSTEM.SAMPLE.CICS.DLQ. Utilisez l'exemple de script MQSC AMQSCIC0.TST pour créer ces files d'attente et des exemples de files d'attente d'entrée.

Génération de votre application sur HP Integrity NonStop Server

Ces informations décrivent les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lorsque vous générez le client IBM WebSphere MQ pour que les applications HP Integrity NonStop Server s'exécutent sous HP Integrity NonStop Server.

C, COBOL et pTAL sont pris en charge.

En-têtes OSS et Guardian et bibliothèques publiques

Fournit des listes d'en-têtes OSS et Guardian et des bibliothèques publiques. Sont répertoriés les en-têtes OSS, les bibliothèques d'importation publique et d'exécution publique OSS, les en-têtes Guardian et les bibliothèques d'importation publique et d'exécution publique Guardian.

«En-têtes OSS», à la page 453

«Bibliothèques d'importation publiques et exécutables publiques OSS», à la page 453

«En-têtes Guardian», à la page 454

«Bibliothèques publiques exécutables et d'importation publiques de Guardian», à la page 455

En-têtes OSS

Tableau 59. En-têtes OSS

Objet	Emplacement	Description
cmqbc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqfc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqec.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqpsc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqxc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqzc.h	<mqinstall>/inc	En-tête IBM WebSphere MQ en langage C (OSS)
cmqcobol.cpy	<mqinstall>/inc	IBM WebSphere MQ copybook COBOL (OSS)
cmqbt.tal	<mqinstall>/inc	En-tête IBM WebSphere MQ pTAL (OSS)
cmqcft.tal	<mqinstall>/inc	En-tête IBM WebSphere MQ pTAL (OSS)
cmqpst.tal	<mqinstall>/inc	En-tête IBM WebSphere MQ pTAL (OSS)
cmqt.tal	<mqinstall>/inc	En-tête IBM WebSphere MQ pTAL (OSS)
cmqxt.tal	<mqinstall>/inc	En-tête IBM WebSphere MQ pTAL (OSS)

Bibliothèques d'importation publiques et exécutables publiques OSS

Tableau 60. Bibliothèques d'importation publiques et exécutables publiques OSS

Objet	Emplacement	Description
libmqic.so	<mqinstall>/bin	Bibliothèque d'exécutables publics IBM WebSphere MQ (OSS non à unités d'exécution)
libmqic_r.so	<mqinstall>/bin	Bibliothèque d'exécutables publics IBM WebSphere MQ (OSS à unités d'exécution multiples)
libmqic.so	<mqinstall>/lib	Bibliothèque d'importation publique IBM WebSphere MQ (OSS non à unités d'exécution)

Tableau 60. Bibliothèques d'importation publiques et exécutables publiques OSS (suite)		
Objet	**Emplacement**	**Description**
libmqic_r.so	<mqinstall>/lib	Bibliothèque d'importation publique IBM WebSphere MQ (OSS à unités d'exécution multiples)
mqicb	<mqinstall>/lib	Bibliothèque d'importation publique IBM WebSphere MQ pour COBOL (OSS)

En-têtes Guardian

Tableau 61. En-têtes Guardian		
Objet	**Emplacement**	**Description**
cmqbch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqfch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqech	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqpsch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqxch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqzch	<mqinstall>/inc/G	En-tête en langage C IBM WebSphere MQ (Guardian)
cmqcobol	<mqinstall>/inc/G	Fichier de stockage COBOL IBM WebSphere MQ (Guardian)
Commande cmqbt	<mqinstall>/inc/G	En-tête IBM WebSphere MQ pTAL (Guardian)
Cmqcft	<mqinstall>/inc/G	En-tête IBM WebSphere MQ pTAL (Guardian)
cmqpst	<mqinstall>/inc/G	En-tête IBM WebSphere MQ pTAL (Guardian)
cmqt	<mqinstall>/inc/G	En-tête IBM WebSphere MQ pTAL (Guardian)
cmqxt	<mqinstall>/inc/G	En-tête IBM WebSphere MQ pTAL (Guardian)

Bibliothèques publiques exécutables et d'importation publiques de Guardian

Tableau 62. Bibliothèques publiques exécutables et d'importation publiques de Guardian		
Objet	**Emplacement**	**Description**
mqic	<mqinstall>/bin/G	Bibliothèque d'exécutables publique IBM WebSphere MQ (Guardian)
mqicb	<mqinstall>/lib/G	Bibliothèque d'importation publique IBM WebSphere MQ pour COBOL (Guardian)

Préparation des programmes C dans HP Integrity NonStop Server

Cette rubrique contient des informations à prendre en compte lorsque vous préparez des programmes C dans HP Integrity NonStop Server , ainsi que des exemples de commandes que vous utilisez lorsque vous générez des applications lorsque vous utilisez le compilateur OSS C et lorsque vous utilisez le compilateur Guardian C.

Les programmes C précompilés sont fournis dans le répertoire MQ_INSTALLATION_PATH/opt/mqm/samp/bin . Pour générer un exemple à partir du code source, utilisez le compilateur c89 .

Vous devez lier vos programmes à la bibliothèque appropriée fournie par IBM WebSphere MQ. Le tableau suivant répertorie les bibliothèques auxquelles vous devez établir un lien lorsque vous préparez des programmes C sous HP Integrity NonStop Server.

Tableau 63. . Bibliothèques de liens HP Integrity NonStop Server	
Bibliothèque	**Description**
libmqic.so	OSS non fileté
libmqic_r.so	OSS à unités d'exécution multiples
mqic	Guardian

Les applications IBM WebSphere MQ natives à unités d'exécution multiples doivent utiliser la fonction Posix User Threads (PUT). Ce produit ne prend pas en charge les unités d'exécution Standard Posix (SPT).

Génération d'applications à l'aide du compilateur OSS C

Cette section contient des exemples de commandes utilisées pour générer des programmes qui sont ciblés pour OSS ou Guardian lorsque vous utilisez le compilateur OSS.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.

L'exemple suivant génère une application OSS client C sans unités d'exécution:

```
c89 -Wsystype=oss -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
```

L'exemple suivant génère une application OSS client C à unités d'exécution multiples:

```
c89 -Wsystype=oss -D_PUT_MODEL_ -o amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic_r -lput
```

L'exemple suivant génère une application client Guardian C:

```
c89 -Wsystype=guardian -o /G/vol/subvol/amqsputc amqsput0.c -IMQ_INSTALLATION_PATH/opt/mqm/inc
-LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
```

Génération d'applications à l'aide du compilateur Guardian C

Cette section contient des exemples de commandes utilisées pour générer des programmes ciblés pour Guardian lorsque vous utilisez le compilateur Guardian.

MQ_INSTALLATION_PATH représente le volume et le sous-volume Guardian dans lesquels IBM WebSphere MQ est installé.

L'exemple suivant génère une application client Guardian C:

```
CCOMP /in AMQSPUT0/ AMQSPUTC;&
       runnable,systype guardian,nolist,&
       ssv0 "$system.system",&
       ssv1 "MQINSTALLATION_SUBVOL",&
       ld(-LMQINSTALLATION_SUBVOL -lmqic)
```

Préparation de programmes COBOL

Cette rubrique contient des informations à prendre en compte lors de la préparation de programmes C pour le client IBM WebSphere MQ for HP Integrity NonStop Server. Il contient des exemples de commandes que vous utilisez lorsque vous générez des applications lorsque vous utilisez le compilateur OSS ECOBOL et lorsque vous utilisez le compilateur Guardian ECOBOL.

Pour générer un exemple COBOL à partir du code source, utilisez le compilateur ECOBOL.

Le tableau suivant répertorie les bibliothèques nécessaires lors de la préparation de programmes COBOL sous HP Integrity NonStop Server. Vous devez lier vos programmes à la bibliothèque appropriée fournie par IBM WebSphere MQ.

Tableau 64. . Bibliothèques de liens HP Integrity NonStop Server

Bibliothèque	**Description**
libmqic.so	OSS non fileté
mqic	Guardian

Lorsque vous exécutez une application COBOL qui se connecte à un gestionnaire de files d'attente, vous devez d'abord définir la variable *SAVE-ENVIRONMENT* sur ON. Pour définir la variable *SAVE-ENVIRONMENT* sur ON:

- Pour OSS, entrez la commande suivante:

```
export SAVE-ENVIRONMENT=ON
```

- Pour Guardian, entrez la commande suivante:

```
param SAVE-ENVIRONMENT ON
```

Si vous ne définissez pas la variable *SAVE-ENVIRONMENT* sur ON, lorsque l'application tente de se connecter à un gestionnaire de files d'attente, elle échoue avec le code anomalie 2058 (080A) (RC2058): MQRC_Q_MGR_NAME_ERROR.

Génération d'applications à l'aide du compilateur OSS ECOBOL

Cette section contient des exemples de commandes utilisées pour générer des programmes ciblés pour OSS ou Guardian lorsque vous utilisez le compilateur OSS ECOBOL.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.

L'exemple suivant génère une application OSS client COBOL:

```
ecobol -Wsystype=oss
       -Wcobol="ansi;port"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

L'exemple suivant génère une application COBOL client Guardian:

```
ecobol -Wsystype=guardian
       -Wcobol="ansi;port;save all"
       -Wcobol="consult MQ_INSTALLATION_PATH/opt/mqm/lib/mqicb"
       -Wcopylib=MQ_INSTALLATION_PATH/opt/mqm/inc/cmqcobol.cpy
       -LMQ_INSTALLATION_PATH/opt/mqm/lib/G -lmqic
       -o amq0put0
       MQ_INSTALLATION_PATH/opt/mqm/samp/amq0put0.cbl
```

Génération d'applications à l'aide du compilateur Guardian ECOBOL

Cette section contient des exemples de commandes utilisées pour générer des programmes qui sont ciblés pour Guardian lorsque vous utilisez le compilateur ECOBOL Guardian.

MQ_INSTALLATION_SUBVOL représente le volume et le sous-volume Guardian dans lesquels IBM WebSphere MQ est installé.

L'exemple suivant génère une application COBOL client Guardian:

```
ECOBOL /in MQSPUTL/ MQSPUT,MQINSTALLATION_SUBVOL.cmqcobol;
       call-shared;ansi;port;save all;nolist;runnable;
       consult MQINSTALLATION_SUBVOL.mqicb;
       eld(-LMQINSTALLATION_SUBVOL -lmqic)
```

Préparation des programmes pTAL

Apprenez à générer des programmes pTAL pour le client IBM WebSphere MQ sur la plateforme HP Integrity NonStop Server .

Pour générer un exemple pTAL à partir du code source, utilisez le compilateur EPTAL.

Remarque :

- Les applications pTAL IBM WebSphere MQ doivent utiliser une routine principale écrite en langage C ou COBOL.
- Les applications pTAL ne peuvent être générées que dans Guardian.

Le tableau suivant répertorie la bibliothèque requise lors de la préparation des programmes pTAL sous HP Integrity NonStop Server. Vous devez lier vos programmes à la bibliothèque appropriée fournie par IBM WebSphere MQ.

Tableau 65. . Bibliothèque de liens HP Integrity NonStop Server

Bibliothèque	**Description**
mqic	Guardian

Génération d'applications à l'aide du compilateur Guardian EPTAL

Cette section contient des exemples de commandes utilisées pour générer des programmes ciblés pour Guardian lorsque vous utilisez le compilateur Guardian EPTAL.

MQINSTALLATION_SUBVOL représente le volume et le sous-volume Guardian dans lesquels IBM WebSphere MQ est installé.

Les applications pTAL IBM WebSphere MQ doivent utiliser une routine principale écrite en langage C ou COBOL.

L'exemple suivant génère une application pTAL client Guardian:

```
ASSIGN SSV0, $SYSTEM.SYSTEM
ASSIGN SSV1, MQINSTALLATION_SUBVOL

EPTAL /in MQINSTALLATION_SUBVOL.MQSPUTT/ MQSPUTO;nolist

CCOMP /in MQINSTALLATION_SUBVOL.MQSPTMC/ MQSPUT;
       runnable,systype guardian,extensions,nolist,
       ssv0 "$system.system",
       ssv1 "MQINSTALLATION_SUBVOL",
       eld(MQSPUTO -LMQINSTALLATION_SUBVOL -lmqic)
```

Génération de votre application sous HP-UX

Ces informations décrivent les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications WebSphere MQ for HP-UX à exécuter sous HP-UX.

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 WebSphere MQ for HP-UX 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 WebSphere MQ for HP-UX 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 WebSphere MQ», à la page 84 .

Dans cette rubrique, nous utilisons la barre oblique inversée (\) pour fractionner les commandes longues sur plusieurs lignes. N'entrez pas ce caractère ; entrez chaque commande sur une seule ligne.

Préparation des programmes C sous HP-UX

Cette rubrique contient des informations à prendre en compte lors de la préparation de programmes C dans HP-UX; avec des exemples pour la plateforme IA64 (IPF).

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Travaillez dans votre environnement normal. Les programmes C précompilés sont fournis dans le répertoire *MQ_INSTALLATION_PATH*/samp/bin .

Pour plus d'informations sur la programmation des applications 64 bits, voir Codage des normes sur les plateformes 64 bits.

Pour utiliser SSL, les clients WebSphere MQ MQI sur HP-UX doivent être générés à l'aide d'unités d'exécution POSIX .

Voici quelques exemples à prendre en compte:

- «Plateforme IA64 (IPF)», à la page 458
- «Liaison de bibliothèques», à la page 460

Plateforme IA64 (IPF)

Générez des exemples de amqsput0, cliexit et srvexit sur la plateforme IA64(IPF).

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application client dans un environnement 32 bits sans unités d'exécution:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application client dans un environnement 32 bits à unités d'exécution:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqic_r -lpthread
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application client dans un environnement 64 bits sans unités d'exécution:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application client dans un environnement 64 bits à unités d'exécution:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application serveur dans un environnement 32 bits sans unités d'exécution:

```
c89 -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application serveur dans un environnement 32 bits à unités d'exécution:

```
c89 -mt -Wl,+b,: +e -D_HPUX_SOURCE -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -lmqm_r -lpthread
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application serveur dans un environnement 64 bits sans unités d'exécution:

```
c89 +DD64 +e -D_HPUX_SOURCE -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

L'exemple suivant génère l'exemple de programme amqsput0 en tant qu'application serveur dans un environnement 64 bits à unités d'exécution:

```
c89 -mt +DD64 +e -D_HPUX_SOURCE -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
  -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

L'exemple suivant génère un cliexit d'exit client dans un environnement 32 bits sans unités d'exécution:

```
c89 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic
```

L'exemple suivant génère un cliexit d'exit client dans un environnement 32 bits à unités d'exécution:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqic_r -lpthread
```

L'exemple suivant génère un cliexit d'exit client dans un environnement 64 bits sans unités d'exécution:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
```

```
ld -b cliexit.o +ee MQStart -o /var/mqm/exits64/cliexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic
```

L'exemple suivant génère un cliexit d'exit client dans un environnement 64 bits à unités d'exécution:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o cliexit.o cliexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b cliexit.o +ee MQStart -o /var/mqm/exits/cliexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqic_r -lpthread
```

L'exemple suivant génère un exit serveur srvexit dans un environnement 32 bits sans unités d'exécution:

```
c89 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32 -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm
```

L'exemple suivant génère un exit serveur srvexit dans un environnement 32 bits à unités d'exécution:

```
c89 -mt +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld +b: -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_32_r -LMQ_INSTALLATION_PATH/lib \
-L/usr/lib/hpux32 -lmqm_r -lpthread
```

L'exemple suivant génère un exit serveur srvexit dans un environnement 64 bits sans unités d'exécution:

```
c89 +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c
-IMQ_INSTALLATION_PATHMQ_INSTALLATION_PATH/inc
ld -b srvexit.o +ee MQStart -o /var/mqm/exits64/srvexit_64 \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm
```

L'exemple suivant génère un exit serveur srvexit dans un environnement 64 bits à unités d'exécution:

```
c89 -mt +DD64 +e +z -c -D_HPUX_SOURCE -o srvexit.o srvexit.c -IMQ_INSTALLATION_PATH/inc
ld  -b srvexit.o +ee MQStart -o /var/mqm/exits/srvexit_64_r \
-LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -lmqm_r -lpthread
```

Liaison de bibliothèques

Vous devez lier vos programmes à la bibliothèque appropriée fournie par WebSphere MQ.

Le tableau suivant indique la bibliothèque à utiliser dans différents environnements

Plateforme matérielle	Environnement à unités d'exécution ou non	Type de programme / d'exit	Fichier de bibliothèque
IA64 (IPF)	Unité d'exécution	Serveur et client pour C	libmqm_r.so
IA64 (IPF)	Unité d'exécution	Client pour C	libmqic_r.so
IA64 (IPF)	Sans unités d'exécution	Serveur et client pour C	libmqm.so
IA64 (IPF)	Sans unités d'exécution	Client pour C	libmqic.so

Remarque :

1. Si vous écrivez un service installable (voir Administration pour plus d'informations), vous devez établir un lien vers la bibliothèque `libmqmzf.sl` .
2. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encinaou BEA Tuxedo, vous devez établir un lien vers les bibliothèques `libmqmxa.sl` (ou `libmqmxa64.sl` si votre gestionnaire de transactions traite le type'long'comme étant 64 bits) et `libmqz.sl` dans une application non à unités d'exécution et vers les bibliothèques `libmqmxa_r.sl` (ou `libmqmxa64_r.sl`) et `libmqz_r.sl` dans une application à unités d'exécution.

3. Vous devez lier les bibliothèques WebSphere MQ avant les autres bibliothèques de produit.

Préparation des programmes COBOL sous HP-UX

Apprenez à préparer des programmes COBOL dans HP-UX, à l'aide de Micro Focus Server Express avec WebSphere MQ sur la plateforme IA64 (IPF) et à exécuter des programmes dans l'environnement client WebSphere MQ MQI.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Remarques à l'intention des utilisateurs

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. 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.

Compilez les programmes à l'aide du compilateur Micro Focus. Les fichiers de copie qui déclarent les structures se trouvent dans *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

Compilation de programmes 32 bits:

```
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb Server for COBOL
$ cob32 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb Client for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r Threaded Server for COBOL
$ cob32 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r Threaded Client for COBOL
```

Compilation de programmes 64 bits:

```
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb Server for COBOL
$ cob64 -xv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb Client for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r Threaded Server for COBOL
$ cob64 -xtv amqsput.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r Threaded Client for COBOL
```

où `amqsput` est un exemple de programme

Vérifiez que vous avez spécifié des tailles de pile d'exécution adéquates. La valeur minimale recommandée est de 16 Ko.

Vous devez lier vos programmes à la bibliothèque appropriée fournie par WebSphere MQ. Le tableau suivant indique la bibliothèque à utiliser dans différents environnements

Plateforme matérielle	Type de programme / d'exit	Fichier de bibliothèque
IA64 (IPF)	Serveur pour COBOL	libmqmcb.so
IA64 (IPF)	Client pour COBOL	libmqicb.so
IA64 (IPF)	Applications à unités d'exécution	libmqmcb_r.so

Utilisation de Micro Focus Server Express avec WebSphere MQ sur la plateforme IA64 (IPF)

Pour plus de détails sur l'utilisation de Micro Focus Server Express avec WebSphere MQ sur la plateforme HP/IPF, voir «Modèles d'espace adresse pris en charge par WebSphere MQ for HP-UX sur IA64 (IPF)», à la page 463 .

Programmes à exécuter dans l'environnement client WebSphere MQ MQI

Si vous utilisez l'unité logique 6.2 pour connecter votre client MQI à un serveur, liez votre application à libsna.a, qui fait partie du produit SNAplusAPI . Utilisez les options `-lV3` et `-lstr` dans votre commande de compilation et de liaison.

- L'option `-lV3` permet à votre programme d'accéder à la bibliothèque de signalisation AT & T (SNAplusAPI utilise les signaux AT & T)
- L'option `-lstr` lie votre programme au composant Streams

Préparation des programmes CICS sous HP-UX

Apprenez à générer des programmes de transaction CICS dans HP-UX.

Pour générer l'exemple de transaction CICS , amqscic0.ccs, exécutez la commande suivante:

```
$ export USERLIB="-lmqm_r"
$ cicstcl -l C amqscic0.ccs
```

Un module de commutation XA est fourni pour vous permettre de lier CICS à WebSphere MQ:

Tableau 66. Code essentiel pour les applications CICS (HP-UX)		
Description	**C (source)**	**C (exec)**
Routine d'initialisation XA	amqzscix.c	amqzsc

Pour plus d'informations sur la prise en charge des transactions CICS , voir Administration.

Prise en charge de TXSeries CICS

WebSphere MQ sous HP-UX 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 MQ .

Ecrivez les programmes WebSphere MQ qui sont chargés dans la même région CICS en langage 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`.

Exemple de transaction CICS C

L'exemple de source C pour une transaction CICS WebSphere 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 le nom de file d'attente contenu dans l'en-tête de transmission du message. Les échecs sont envoyés à la file d'attente SYSTEM.SAMPLE.CICS.DLQ. Utilisez l'exemple de script MQSC AMQSCIC0.TST pour créer ces files d'attente et des exemples de files d'attente d'entrée.

Préparation de programmes COBOL CICS à l'aide de Micro Focus COBOL

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Pour utiliser Micro Focus COBOL, procédez comme suit:

1. Ajoutez le module de bibliothèque d'exécution COBOL WebSphere MQ à la bibliothèque d'exécution à l'aide de la commande suivante:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
   ```

   Remarque : Avec `cicsmkcobol`, WebSphere 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 ces fonctions, n'incluez pas la WebSphere MQ `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 COBOL Micro Focus et permet à la bibliothèque COBOL d'exécution CICS d'appeler WebSphere MQ sur les systèmes UNIX and 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 de CICS for HP-UX
   - Nouvelle version ou édition de tout produit de base de données pris en charge (pour les transactions COBOL uniquement)
   - Nouvelle version ou édition de WebSphere 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
   ```

Modèles d'espace adresse pris en charge par WebSphere MQ for HP-UX sur IA64 (IPF)

HP-UX fournit plusieurs modèles d'espace adresse qui peuvent être exploités par les applications WebSphere MQ .

HP-UX prend en charge deux modèles d'espace adresse:

- MGAS-Espace adresse global en grande partie (valeur par défaut utilisée par WebSphere MQ)
- MPAS-Espace d'adressage privé en grande quantité

Les applications qui se connectent à WebSphere MQ peuvent utiliser les modèles d'espace adresse MGAS ou MPAS. Les applications générées à l'aide du modèle MPAS qui se connectent à WebSphere MQ à l'aide de la mémoire partagée peuvent entraîner des coûts de performances mineurs en raison de l'inefficacité du mappage des pages de mémoire partagée utilisées par WebSphere MQ dans l'espace adresse virtuel du programme MPAS.

Les applications COBOL créées à l'aide de Micro Focus Server Express utilisent le modèle MPAS par défaut.

Vous pouvez utiliser le programme **chatr** pour vérifier et modifier le modèle d'adressage utilisé par un programme.

Si vous rencontrez des problèmes lors de la connexion à WebSphere MQ à partir de programmes MPAS 32 bits, envisagez d'utiliser le modèle d'adressage MGAS ou de générer votre application en tant qu'application MPAS 64 bits plutôt qu'en tant qu'application MPAS 32 bits.

Vous trouverez plus de détails sur les modèles d'espace adresse MGAS et MPAS dans la documentation HP-UX .

Génération de votre application sur Linux

Ces informations décrivent les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération des applications WebSphere 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++.

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 WebSphere 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 WebSphere 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.so	Serveur pour C
libmqic.so & libmqm.so	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.so	Serveur pour C

Fichier de bibliothèque	Type de programme / d'exit
libmqic_r.so & libmqm_r.so	Client pour C

Remarque :

1. Si vous écrivez un service installable (voir Administration pour plus d'informations), vous devez établir un lien vers la bibliothèque `libmqmzf.so` .
2. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encinaou BEA Tuxedo, vous devez établir un lien vers les bibliothèques `libmqmxa.so` (ou `libmqmxa64.so` si votre gestionnaire de transactions traite le type'long'comme étant 64 bits) et `libmqz.so` dans une application non à unités d'exécution et vers les bibliothèques `libmqmxa_r.so` (ou `libmqmxa64_r.so`) et `libmqz_r.so` dans une application à unités d'exécution.
3. Vous devez lier les bibliothèques WebSphere MQ avant les autres bibliothèques de produit.

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 WebSphere MQ est installé.

Application client C, 31 bits, sans unité d'exécution

```
gcc -m31  -o famqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Application client C, 31 bits, avec unités d'exécution

```
gcc -m31  -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Application serveur C, 31 bits, sans unités d'exécution

```
gcc -m31  -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Application serveur C, 31 bits, avec unités d'exécution

```
gcc -m31  -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Application client C++, 31 bits, sans unités d'exécution

```
g++ -m31 -fsigned-char  -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Application client C + +, 31 bits, avec unités d'exécution

```
g++ -m31 -fsigned-char  -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r -lpthread
```

Application serveur C + +, 31 bits, sans unités d'exécution

```
g++ -m31 -fsigned-char  -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Application serveur C + +, 31 bits, à unités d'exécution

```
g++ -m31 -fsigned-char  -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 31 bits, sans unité d'exécution

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

Exit client C, 31 bits, avec unités d'exécution

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

Exit de serveur C, 31 bits, sans unité d'exécution

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm
```

Exit de serveur C, 31 bits, unité d'exécution

```
gcc -m31 -shared -fPIC  -o /var/mqm/exits/srvexit_32_r srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqm_r -lpthread
```

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 WebSphere MQ est installé.

Application client C, 32 bits, sans unités d'exécution

```
gcc -m32 -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic
```

Application client C, 32 bits, avec unités d'exécution

```
gcc -m32 -o amqsputc_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqic_r -lpthread
```

Application serveur C, 32 bits, sans unités d'exécution

```
gcc -m32 -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Application serveur C, 32 bits, à unités d'exécution

```
gcc -m32 -o amqsput_32_r amqsput0.c -IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm_r -lpthread
```

Application client C++ 32 bits, sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Application client C++ 32 bits, avec unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Application serveur C + +, 32 bits, sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Application serveur C + +, 32 bits, avec unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 32 bits, sans unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic
```

Exit client C, 32 bits, avec unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/cliexit_32_r cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
-lmqic_r -lpthread
```

Exit de serveur C, 32 bits, sans unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32 srvexit.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -lmqm
```

Exit de serveur C, 32 bits, à unités d'exécution

```
gcc -m32 -shared -fPIC -o /var/mqm/exits/srvexit_32_r srvexit.c
IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
-Wl,-rpath=/usr/lib
lmqm_r -lpthread
```

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 WebSphere MQ est installé.

Application client C, 64 bits, sans unités d'exécution

```
gcc -m64  -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic
```

Application client C, 64 bits, avec unités d'exécution

```
gcc -m64  -o amqsputc_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqic_r
-lpthread
```

Application serveur C, 64 bits, sans unités d'exécution

```
gcc -m64  -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm
```

Application serveur C, 64 bits, à unités d'exécution

```
gcc -m64  -o amqsput_64_r amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -lmqm_r
-lpthread
```

Application client C++ 64 bits, sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application client C++ 64 bits, avec unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Application serveur C++ 64 bits, sans unités d'exécution

```
g++ -m64 -fsigned-char  -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Application serveur C++ 64 bits, avec unités d'exécution

```
g++ -m64 -fsigned-char  -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Exit client C, 64 bits, sans unités d'exécution

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic
```

Exit client C, 64 bits, avec unités d'exécution

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/cliexit_64_r cliexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqic_r -lpthread
```

Exit de serveur C, 64 bits, sans unités d'exécution

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm
```

Exit de serveur C, 64 bits, avec unités d'exécution

```
gcc -m64 -shared -fPIC  -o /var/mqm/exits64/srvexit_64_r srvexit.c
-IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64
-Wl,-rpath=/usr/lib64 -lmqm_r -lpthread
```

Préparation de programmes COBOL dans Linux

Découvrez la préparation de programmes COBOL dans Linux et la préparation de programmes COBOL à l'aide de Micro Focus COBOL.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere 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

Fichier de bibliothèque	Type de programme / d'exit
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 de Micro Focus COBOL

Définissez les variables d'environnement avant de compiler votre programme comme suit:

```
export COBCPY=<COBCPY>
export LIB=MQ_INSTALLATION_PATHlib:$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.

Génération de votre application sous Solaris

Ces informations décrivent les tâches supplémentaires et les modifications apportées aux tâches standard que vous devez effectuer lors de la génération d'applications WebSphere MQ for Solaris à exécuter sous Solaris.

Les langages de programmation COBOL, C et C + + sont pris en charge. Pour plus d'informations sur la préparation de vos programmes C + +, voir Utilisation de C++.

En plus de coder les appels MQI dans votre code source, vous devez ajouter les fichiers d'inclusion appropriés. Familiarisez-vous avec le contenu de ces fichiers. Pour une description complète, voir «Fichiers de définition de données IBM WebSphere MQ», à la page 84 .

Dans cette rubrique, le caractère barre oblique inversée (\) est utilisé pour fractionner les commandes longues sur plusieurs lignes. N'entrez pas ce caractère, entrez chaque commande sur une seule ligne.

Préparation des programmes C sous Solaris

Les programmes C précompilés sont fournis dans le répertoire *MQ_INSTALLATION_PATH*/samp/bin .

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Pour plus d'informations sur la programmation d'applications 64 bits, voir Normes de codage sur les plateformes 64 bits.

Si vous souhaitez utiliser les programmes sur une machine sur laquelle seul le WebSphere MQ MQI client for Solaris est installé, compilez les programmes pour les lier à la bibliothèque client (`-lmqic`).

Si vous utilisez le compilateur non pris en charge `?usr?ucb?cc`, votre application peut se compiler et se lier correctement. Toutefois, lorsque vous exécutez l'application, celle-ci échoue lorsqu'elle tente de se connecter au gestionnaire de files d'attente.

Remarque : Les clients SSL et TLS Solaris x86 32 bits configurés pour une opération compatible avec FIPS 140-2 échouent lors de l'exécution sur des systèmes Intel. Cet échec survient car le chargement du fichier de bibliothèque GSKit-Crypto Solaris x86 32 bits compatible avec la norme FIPS 140-2 échoue sur le jeu de circuits Intel. Sur les systèmes affectés, l'erreur AMQ9655 est signalée dans le journal des erreurs du client. Pour résoudre ce problème, désactivez la conformité à la norme FIPS 140-2 ou recompilez l'application client 64 bits, car le code 64 bits n'est pas affecté.

Liaison de bibliothèques

Vous devez établir un lien avec les bibliothèques WebSphere MQ appropriées à votre type d'application:

Fichiers de bibliothèque	Type de programme / d'exit
libmqm.so	Serveur pour C
libmqic.so & libmqm.so	Client pour C

Remarque :

1. Si vous écrivez un service installable (pour plus d'informations, voir Administration), lien vers la bibliothèque `libmqmzf.so` .
2. Si vous produisez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encinaou BEA Tuxedo, vous devez établir un lien vers les bibliothèques `libmqmxa.so` (ou `libmqmxa64.so` si votre gestionnaire de transactions traite le type'long'comme étant 64 bits) et `libmqz.so` .
3. Vous devez lier les bibliothèques WebSphere MQ avant les autres bibliothèques de produit.

Génération d'applications sur x86-64

Cette rubrique contient des exemples de commandes utilisées pour générer des programmes dans divers environnements sur la plateforme x86-64 .

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Application client C, 32 bits

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Application client C, 64 bits

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

Application serveur C, 32 bits

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Application serveur C, 64 bits

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

Application client C++ 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

Application client C++ 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Application serveur C++ 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Application serveur C++ 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Exit client C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

Exit client C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Exit de serveur C, 32 bits

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

Exit de serveur C, 64 bits

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Génération d'applications sur SPARC

Cette rubrique contient des exemples de commandes utilisées pour générer des programmes dans divers environnements sur la plateforme SPARC.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Application client C, 32 bits

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Application client C, 64 bits

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Application serveur C, 32 bits

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Application serveur C, 64 bits

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Application client C++ 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

Application client C++ 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Application serveur C++ 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib
-RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Application serveur C++ 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Exit client C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
```

```
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqic
-lsocket -lnsl -ldl
```

Exit client C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Exit de serveur C, 32 bits

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib
-R/usr/lib/32 -lmqm
-lsocket -lnsl -ldl
```

Exit de serveur C, 64 bits

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64
-R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Préparation de programmes COBOL sous Solaris

En savoir plus sur la préparation des programmes COBOL dans Solaris.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere 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. 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.

Compilez les programmes à l'aide du compilateur Micro Focus. Les fichiers de copie qui déclarent les structures se trouvent dans *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB=MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="<COBCPY>"
```

Compilation de programmes 32 bits:

- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb`

  Serveur pour COBOL
- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb`

  Client pour COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r`

  Serveur à unités d'exécution pour COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r`

  Client à unités d'exécution pour COBOL

Compilation de programmes 64 bits:

- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb`

  Serveur pour COBOL
- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb`

  Client pour COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r`

  Serveur à unités d'exécution pour COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r`

  Client à unités d'exécution pour COBOL

où *amqs0put0.cbl* est un exemple de programme.

Vous devez lier votre programme à l'un des éléments suivants:

- libmqmcb.so

  Serveur pour COBOL
- libmqicb.so

  Client pour COBOL

Préparation des programmes CICS sous Solaris

Informations sur la préparation des programmes CICS sous Solaris.

Un module de commutation XA est fourni pour vous permettre de lier CICS à WebSphere MQ:

Tableau 67. Code essentiel pour les applications CICS (Solaris)

Description	**C (source)**	**C (exec)**
Routine d'initialisation XA	amqzscix.c	amqzsc - TXSeries pour Solaris

Liez toujours vos transactions à l'unité d'exécution sécurisée WebSphere MQ library libmqm.so.

Pour plus d'informations sur la prise en charge des transactions CICS , voir Administration.

Prise en charge de TXSeries CICS

WebSphere MQ for Solaris prend en charge TXSeries CICS à l'aide de l'interface XA.

Ecrivez les programmes WebSphere MQ qui sont chargés dans la même région CICS en langage 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 Micro Focus COBOL

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Pour utiliser Micro Focus COBOL, procédez comme suit:

1. Ajoutez le module de bibliothèque d'exécution COBOL WebSphere MQ à la bibliothèque d'exécution à l'aide de la commande suivante:

   ```
   cicsmkcobol -L/usr/lib/dce -LMQ_INSTALLATION_PATH/lib  \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
   ```

   Remarque : Avec `cicsmkcobol`, WebSphere 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, déplacez ces fonctions des applications COBOL vers votre propre bibliothèque, par exemple, `myMQ.so`. Après avoir déplacé ces fonctions, n'incluez pas la WebSphere MQ `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 COBOL Micro Focus et permet à la bibliothèque COBOL d'exécution CICS d'appeler WebSphere MQ sur les systèmes UNIX and 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 de TXSeries for Solaris
   - Nouvelle version ou édition de tout produit de base de données pris en charge (pour les transactions COBOL uniquement)
   - Nouvelle version ou édition de WebSphere 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

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 CICS WebSphere 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. Les échecs sont envoyés à la

file d'attente SYSTEM.SAMPLE.CICS.DLQ. Utilisez l'exemple de script MQSC AMQSCIC0.TST pour créer ces files d'attente et des exemples de files d'attente d'entrée.

Génération de votre application sur des systèmes Windows

Les publications des systèmes 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 WebSphere MQ for Windows sous Windows . Les langages de programmation ActiveX, C, C + +, COBOL et Visual Basic sont pris en charge. Pour plus d'informations sur la préparation de vos programmes ActiveX , voir Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX). 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 WebSphere 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 WebSphere 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 WebSphere MQ», à la page 84 .

Génération d'applications 64 bits sous Windows

Les applications 32 bits et 64 bits sont prises en charge sur IBM WebSphere MQ for Windows Version 7.5. Les fichiers exécutables et les fichiers de bibliothèque de IBM WebSphere 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 WebSphere MQ sont fournies dans les emplacements suivants:

Tableau 68. Emplacement des bibliothèques IBM WebSphere 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 WebSphere 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.

Préparation des programmes C sous Windows

Travaillez dans votre environnement Windows standard ; WebSphere 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 WebSphere 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 WebSphere 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 pour plus d'informations), vous devez établir un lien vers la bibliothèque mqmzf.lib .
- Si vous créez une application pour la coordination externe par un gestionnaire de transactions compatible XA tel que IBM TXSeries Encinaou BEA Tuxedo, vous devez établir une liaison avec la bibliothèque mqmxa.lib ou mqmxa.lib .
- Si vous écrivez un exit CICS , accédez à la bibliothèque mqmcics4.lib .
- Vous devez lier les bibliothèques WebSphere MQ avant les autres bibliothèques 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 WebSphere MQ for Windows à WebSphere MQ sur les systèmes UNIX and Linux , où l'utilisation de minuscules est nécessaire.

Préparation des programmes CICS et Transaction Server

L'exemple de source C pour une transaction CICS WebSphere MQ est fourni par AMQSCIC0.CCS. Vous le générez à l'aide des fonctions CICS standard. Par exemple, pour TXSeries pour Windows 2000:

1. Définissez la variable d'environnement (entrez le code suivant sur une seule ligne):

```
set CICS_IBMC_FLAGS=-IMQ_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 WebSphere 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.

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 WebSphere MQ est installé.

Pour préparer des programmes COBOL sur des systèmes Windows , liez votre programme à l'une des bibliothèques suivantes fournies par IBM WebSphere MQ:

Fichier de bibliothèque	Type de programme ou d'exit
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb	Serveur 32 bits pour IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb	Serveur 32 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb	Client 32 bits pour IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb	Client 32 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb	Serveur 64 bits pour IBM COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcb	Serveur 64 bits pour Micro Focus COBOL
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccbb	Client 64 bits pour IBM COBOL

Fichier de bibliothèque	Type de programme ou d'exit
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 WebSphere MQ .

Vous pouvez utiliser le compilateur IBM COBOL Set ou le compilateur Micro Focus COBOL en fonction du programme:

- Les programmes commençant par `amqi` sont adaptés au compilateur IBM COBOL Set,
- Les programmes démarrant `amqm` sont adaptés au compilateur Micro Focus COBOL, et
- Les programmes démarrant `amq0` conviennent à l'un ou l'autre compilateur.

IBM et Micro Focus COBOL

Reliez tous les programmes IBM WebSphere 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 IBM VisualAge COBOL:

1. Définissez la variable d'environnement SYSLIB pour inclure le chemin d'accès aux fichiers de stockage COBOL IBM WebSphere MQ VisualAge (entrez le code suivant sur une ligne):

   ```
   set SYSLIB=MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook\VAcobol;%SYSLIB%
   ```

2. Pour une utilisation sur le serveur IBM WebSphere MQ :

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqmcbb.lib"
   ```

3. A utiliser sur le client IBM WebSphere MQ :

   ```
   cob2 amq0put0.cbl -qlib "MQ_INSTALLATION_PATH\
   Tools\Lib\mqiccbb.lib"
   ```

   Remarque : Bien que vous deviez utiliser l'option de compilation `CALLINT (SYSTEM)`, il s'agit de la valeur par défaut pour `cob2`.

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 WebSphere 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 WebSphere MQ :

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

- Ou liez le fichier objet à utiliser sur le client IBM WebSphere 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 document *Transaction Server for Windows NT, V4 Application Programming* Guide.

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
```

Préparation des programmes Visual Basic sous Windows

Utilisez ces informations lorsque vous envisagez d'utiliser des programmes Visual Basic sous Windows.

Remarque : Les versions 64 bits des fichiers du 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	Prise en charge de MQAI
CMQCFB.BAS	Prise en charge de PCF
CMQXB.BAS	Prise en charge des exits de canal
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 90 .

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 WebSphere 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 sur 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
   ```

Exit de sécurité SSPI

WebSphere MQ for Windows fournit un exit de sécurité pour le client WebSphere MQ MQI et le serveur WebSphere MQ . Il s'agit d'un programme d'exit de canal qui fournit l'authentification pour les canaux WebSphere MQ à l'aide de l'interface SSPI (Security Services Programming Interface). L'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 sur les systèmes Windows», à la page 176.

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 69. 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 «WebSphere Contrôle d'accès MQ et principaux Windows», à la page 484). 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.

WebSphere Contrôle d'accès MQ et principaux Windows

Le contrôle d'accès fourni par WebSphere 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 WebSphere 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/
```

Utilisation des services de protocole LDAP (Lightweight Directory Access Protocol) avec WebSphere MQ for Windows

Cette rubrique explique ce qu'est un service d'annuaire et le rôle joué par un protocole d'accès à l'annuaire (DAP). Il explique également comment les applications WebSphere MQ peuvent utiliser un annuaire LDAP (Lightweight Directory Access Protocol) en utilisant un exemple de programme comme guide.

Remarque : L'exemple de programme est conçu pour une personne qui connaît déjà LDAP.

Les rubriques suivantes fournissent des informations supplémentaires sur les services d'annuaire, LDAP et l'utilisation de LDAP avec WebSphere MQ.

- «Service d'annuaire», à la page 484
- «protocole LDAP (Lightweight Directory Access Protocol)», à la page 485
- «Utilisation de LDAP avec WebSphere MQ», à la page 485

Service d'annuaire

Un répertoire est un référentiel d'informations sur les objets, qui est organisé de telle sorte qu'il est facile de trouver les informations sur un objet spécifique.

Un exemple courant est un annuaire téléphonique, où l'information (adresse et numéro de téléphone) est stockée sur les personnes et les entreprises. Un autre exemple est un carnet d'adresses pour un système de messagerie, dans lequel les adresses électroniques, et éventuellement d'autres informations telles que les numéros de téléphone, sont stockées pour les personnes.

Sur les systèmes informatiques, les répertoires peuvent stocker des informations sur les ressources informatiques, telles que les imprimantes ou les disques partagés. Par exemple, vous pouvez utiliser un répertoire pour savoir où se trouve l'imprimante couleur la plus proche. Dans une application WebSphere MQ , un répertoire peut être utilisé pour fournir l'association entre un service d'application (tel que le traitement des comptes clients) et la file d'attente à utiliser pour les messages nécessitant ce service (éventuellement identifiés par le nom de la file d'attente et le nom de son gestionnaire de files d'attente hôte).

Les répertoires sont implémentés en tant que systèmes client-serveur, où le serveur d'annuaire contient toutes les informations et répond aux demandes des clients. Les clients peuvent être des programmes

d'interface utilisateur, qui fournissent les informations directement à l'utilisateur, ou des programmes d'application qui doivent localiser des ressources pour effectuer leur travail. Un service d'annuaire comprend le serveur d'annuaire, les programmes d'administration, les bibliothèques client et les programmes nécessaires à la configuration, à la mise à jour et à la lecture de l'annuaire.

protocole LDAP (Lightweight Directory Access Protocol)

Il existe de nombreux services d'annuaire, tels que Novell Directory Services, DCE Cell Directory Service, Banyan StreetTalk, Windows Directory Services, X.500et les services de carnet d'adresses associés aux produits de messagerie. X.500 a été proposé comme norme pour les services d'annuaire mondiaux par l'Organisation internationale de normalisation (ISO). Il a besoin d'une pile de protocoles OSI pour ses communications, et en grande partie à cause de cela, son utilisation a été limitée aux grandes organisations et aux établissements universitaires. Un serveur d'annuaire X.500 communique avec ses clients à l'aide du protocole DAP (Directory Access Protocol).

LDAP (Lightweight Directory Access Protocol) a été créé en tant que version simplifiée de DAP. Il est plus facile à implémenter, omet certaines des fonctionnalités moins utilisées de DAP, et s'exécute sur TCP/IP. À la suite de ces changements, il est rapidement adopté comme protocole d'accès à l'annuaire à la plupart des fins, en remplacement de la multitude de protocoles propriétaires précédemment utilisés. Les clients LDAP peuvent toujours accéder à un serveur X.500 via une passerelle (X.500 requiert toujours la pile de protocoles OSI), ou de plus en plus les implémentations X.500 incluent généralement la prise en charge native de LDAP ainsi que l'accès DAP.

Les annuaires LDAP peuvent être distribués et utiliser la réplication pour permettre un accès efficace à leur contenu.

Pour une description plus complète de LDAP, voir *Understanding LDAP*, une publication IBM Redbooks .

Utilisation de LDAP avec WebSphere MQ

Dans les configurations WebSphere MQ , les informations qui définissent les files d'attente de messages et de transmission sont stockées en local. Cela signifie que dans un réseau WebSphere MQ , les différentes définitions sont distribuées et qu'aucun répertoire central de ces informations n'est disponible pour consultation. La messagerie éloignée entre les applications WebSphere MQ est généralement assurée par l'utilisation de définitions locales de files d'attente éloignées. L'application émet d'abord un appel MQOPEN en utilisant le nom indiqué dans la définition locale de la file d'attente éloignée. Pour placer le message dans la file d'attente éloignée, l'application émet ensuite MQPUT, en spécifiant le descripteur renvoyé par l'appel MQOPEN. La définition de file d'attente éloignée fournit le nom de la file d'attente de destination, le gestionnaire de files d'attente de destination et, le cas échéant, une file d'attente de transmission. Dans cette technique, l'application doit connaître au moment de l'exécution le nom spécifié dans la définition de file d'attente locale.

Une variante du précédent évite l'utilisation de définitions locales de files d'attente éloignées. L'application peut spécifier le nom de la file d'attente de destination complète, qui inclut le nom du gestionnaire de files d'attente éloignées dans le cadre de MQOPEN. L'application doit donc connaître ces deux noms lors de l'exécution. Le gestionnaire de files d'attente local doit être correctement configuré avec la définition de file d'attente locale et avec une file d'attente de transmission nommée de manière appropriée (ou par défaut) et un canal associé qui est distribué à la cible.

Dans le cas où les gestionnaires de files d'attente source et cible sont définis comme membres du même cluster, les aspects de file d'attente de transmission et de canal des deux scénarios précédents peuvent être ignorés. Si la file d'attente de transmission cible est une file d'attente de cluster, une définition locale d'une file d'attente éloignée n'est pas non plus requise. Toutefois, comme dans les cas précédents décrits, l'application doit toujours connaître le nom de la file d'attente de destination.

Un service d'annuaire peut être utilisé pour supprimer cette dépendance d'application sur les noms de file d'attente (ou la combinaison des noms de file d'attente et de gestionnaire de files d'attente). Le mappage entre les critères d'application et les noms d'objet WebSphere MQ peut être conservé dans un répertoire et mis à jour de manière dynamique, indépendamment des applications. Lors de l'exécution, l'application WebSphere MQ qui souhaite d'abord envoyer un message interroge le répertoire à l'aide de critères basés

sur l'application, par exemple: service_name = "comptes clients", extrait les noms d'objet WebSphere MQ appropriés, puis utilise ces valeurs renvoyées dans l'appel MQOPEN.

Un autre exemple d'utilisation d'un annuaire est celui d'une société qui possède de nombreux petits dépôts ou bureaux, WebSphere MQ Les clients MQI peuvent être utilisés pour envoyer des messages aux serveurs WebSphere MQ situés dans les grands bureaux. Les clients doivent connaître le nom de la machine hôte, le canal MQI et le nom de la file d'attente pour chaque serveur auquel ils envoient des messages. Il peut parfois être nécessaire de déplacer un serveur WebSphere MQ vers une autre machine ; chaque client qui communique avec le serveur doit connaître la modification. Un service d'annuaire LDAP peut être utilisé pour stocker les noms des machines hôte (ainsi que les noms de canal et de file d'attente) et les programmes client peuvent extraire les informations de l'annuaire à chaque fois qu'ils souhaitent envoyer un message à un serveur. Dans ce cas, seul le répertoire doit être mis à jour si un nom d'hôte (ou un nom de canal ou de file d'attente) a été modifié.

Plusieurs destinations pour un message d'application peuvent être stockées dans un répertoire, celui choisi dépendant de la disponibilité ou du partage de charge.

WebSphere MQ peut également utiliser un annuaire LDAP pour stocker les informations d'authentification à utiliser avec SSL (Secure Sockets Layer). Les classes WebSphere MQ pour Java peuvent également stocker des informations dans un annuaire LDAP.

Exemple de programme LDAP

L'exemple de programme est conçu pour une personne qui connaît LDAP et qui l'utilise probablement déjà. Il est destiné à montrer comment les applications WebSphere MQ peuvent utiliser un annuaire LDAP.

Génération de l'exemple de programme

Ce programme a été généré et testé uniquement sous Windows à l'aide de TCP/IP. Outre les considérations générales mentionnées dans «Préparation des programmes C sous Windows», à la page 477, notez les points suivants:

- Ce programme étant conçu pour s'exécuter en tant que programme client, il doit être lié à MQIC.LIB .
- Outre les fichiers et les bibliothèques d'en-tête WebSphere MQ , ce programme doit être généré à l'aide de fichiers et de bibliothèques d'en-tête de client LDAP.

  Par exemple, à l'aide du client IBM eNetwork , liez le programme à LIBLDAPSTATICE.LIB et LIBLBERSTATICSSL.LIB .

configuration du répertoire

Pour que l'exemple de programme puisse être exécuté, un serveur d'annuaire LDAP doit être configuré avec des exemples de données.

Le fichier MQuser.ldif, dans le répertoire `tools\c\samples` , contient des exemples de données au format LDIF (LDAP Data Interchange Format). Vous pouvez éditer ce fichier en fonction de vos besoins. Il contient des données pour une société fictive appelée MQuser qui dispose d'un service de transport comprenant trois bureaux. Chacun de ces bureaux dispose d'une machine qui exécute un serveur WebSphere MQ .

Au minimum, vous devez éditer les trois lignes qui contiennent les noms d'hôte des machines exécutant les serveurs WebSphere MQ : lignes 18, 27 et 36:

```
host: LondonHost
  ...
host: SydneyHost
  ...
host: WashingtonHost
```

Vous devez remplacer `LondonHost`, `SydneyHost`et `WashingtonHost` par les noms de trois de vos machines qui exécutent des serveurs WebSphere MQ . Vous pouvez également modifier les noms de canal et de file d'attente si vous le souhaitez (l'exemple utilise les noms des valeurs par défaut du

système). Vous pouvez également augmenter ou diminuer le nombre de bureaux dans les exemples de données.

Configuration du serveur d'annuaire IBM Tivoli

Pour plus d'informations sur l'installation du répertoire, voir IBM Tivoli Directory Server (ITDS)-Guide d'administration. Dans la rubrique `Installing and Configuring Server`, parcourez les sections `Installing Server` et `Basic Server Configuration`. Si nécessaire, lisez la rubrique `Administrator Interface` pour vous familiariser avec le fonctionnement de l'interface.

Dans la rubrique `Configuring - How Do I`, suivez les instructions pour démarrer l'administrateur, puis parcourez la section `Configure Database` et créez une base de données par défaut. Ignorez la section `Configure replica` et, à l'aide de la section `Work with Suffixes`, ajoutez un suffixe `o=MQuser`.

Avant d'ajouter des entrées à la base de données, vous devez étendre le schéma d'annuaire en ajoutant des définitions d'attribut et une définition de classe d'objets. Ceci est décrit dans le manuel IBM Tivoli Directory Server -Guide d'administration dans le chapitre `Reference Information` de la section `Directory Schema`. Deux exemples de fichiers sont inclus pour vous aider dans cette tâche. Le fichier `mq.at.conf` inclut les définitions d'attribut que vous devez ajouter au fichier `?etc?slapd.at.conf`. Pour ce faire, incluez l'exemple de fichier en éditant `slapd.at.conf` et en ajoutant une ligne:

```
include <pathname>/mq.at.conf
```

Vous pouvez également éditer le fichier `slapd.at.conf` et y ajouter directement le contenu de l'exemple de fichier, c'est-à-dire ajouter les lignes:

```
# MQ attribute definitions
attribute mqChannel           ces   mqChannel        1000   normal
attribute mqQueueManager      ces   mqQueueManager   1000   normal
attribute mqQueue             ces   mqQueue          1000   normal
attribute mqPort              cis   mqPort           64     normal
```

De même pour la définition de classe d'objets, vous pouvez inclure l'exemple de fichier en éditant `etc?slapd.oc.conf` et en ajoutant la ligne suivante:

```
include <pathname>/mq.oc.conf
```

ou vous pouvez ajouter le contenu de l'exemple de fichier directement à `slapd.oc.conf,` , c'est-à-dire ajouter les lignes:

```
# MQ object classdefinition
objectclass mqApplication
      requires
              objectClass,
              cn,
              host,
              mqChannel,
              mqQueue
      allows
              mqQueueManager,
              mqPort,
              description,
              l,
              ou,
              seeAlso
```

Vous pouvez maintenant démarrer le serveur d'annuaire (Administration, Serveur, Démarrage) et y ajouter les exemples d'entrées. Pour ajouter les exemples d'entrées, accédez à la page Administration, Ajouter des entrées de l'administrateur, entrez le nom de chemin complet de l'exemple de fichier `MQuser.ldif` et cliquez sur Soumettre.

Le serveur d'annuaire est maintenant en cours d'exécution et chargé avec des données adaptées à l'exécution de l'exemple de programme.

Configuration du serveur d'annuaire Netscape

Dans la page d'administration de Netscape Server, cliquez sur **Créer Netscape Directory Server**.

Un formulaire contenant les informations de configuration doit maintenant vous être présenté. Remplacez le suffixe d'annuaire par **o = MQuser** et ajoutez un mot de passe pour l'utilisateur non restreint. Vous pouvez également modifier d'autres informations en fonction de votre installation. Cliquez sur **OK**pour créer le répertoire. Cliquez sur **Return to Server Administration** et démarrez le serveur d'annuaire. Cliquez sur le nom du répertoire pour démarrer le serveur d'administration du serveur d'annuaire pour le nouveau répertoire.

Avant d'ajouter des entrées à la base de données, étendez le schéma d'annuaire en ajoutant des définitions d'attribut et une définition de classe d'objets. Cliquez sur l'onglet **Schéma** de la page Serveur d'annuaire. Vous voyez maintenant un formulaire qui vous permet d'ajouter de nouveaux attributs. Ajoutez les attributs suivants (laissez l'ID objet de l'attribut vide pour tous les attributs):

```
Attribute Name              Syntax
--------------              ------
mqChannel                   Case Exact String
mqQueueManager              Case Exact String
mqQueue                     Case Exact String
mqPort                      Integer
```

Ajoutez une nouvelle classe objectClass en cliquant sur **Create ObjectClass** dans le panneau latéral. Entrez **mqApplication** comme ObjectClass Name, sélectionnez **applicationProcess** comme ObjectClass parent et laissez la zone **ObjectClass OID** vide. Ajoutez maintenant des attributs à objectClass. Sélectionnez **host**, **mqChannel**et **mqQueue** comme attributs requis, puis sélectionnez **mqQueueManager** et **mqPort** comme attributs autorisés. Appuyez sur le bouton **Create New ObjectClass** pour créer l'objet objectClass.

Pour ajouter les exemples de données, cliquez sur l'onglet **Gestion de la base de données** et sélectionnez **Ajouter des entrées** dans le panneau latéral. Entrez le nom de chemin de l'exemple de fichier de données `<pathname>\MQuser.ldif`, entrez le mot de passe et cliquez sur **OK**.

L'exemple de programme s'exécute en tant qu'utilisateur non autorisé et, par défaut, Netscape Directory n'autorise pas les utilisateurs non autorisés à effectuer des recherches dans le répertoire. Pour cela, cliquez sur l'onglet **Contrôle d'accès** . Entrez le mot de passe de l'utilisateur non restreint et cliquez sur **OK** pour charger les entrées de contrôle d'accès du répertoire. Elles doivent actuellement être vides. Appuyez sur le bouton **Nouveaux ICA** pour créer une entrée de contrôle d'accès. Dans la zone de saisie qui s'affiche, cliquez sur **Refuser** (souligné) et dans la boîte de dialogue qui s'affiche, remplacez-la par **Autoriser**. Ajoutez un nom, par exemple **MQuser-access**, puis cliquez sur **Choisir un suffixe** pour sélectionner **o = MQuser**. Entrez **o = MQuser** comme cible, entrez le mot de passe de l'utilisateur non restreint et cliquez sur **Soumettre**.

Le serveur d'annuaire est maintenant en cours d'exécution et chargé avec des données adaptées à l'exécution de l'exemple de programme.

Exécution de l'exemple de programme

Vous devez maintenant disposer d'un serveur d'annuaire LDAP en cours d'exécution et rempli avec les données d'échantillon. Les données indiquent trois machines hôte qui doivent toutes exécuter des serveurs WebSphere MQ . Vérifiez que le gestionnaire de files d'attente par défaut est en cours d'exécution sur chaque machine (sauf si vous avez modifié les exemples de données pour spécifier un gestionnaire de files d'attente différent).

Démarrez également le programme d'écoute WebSphere MQ sur chaque machine ; l'exemple utilise TCP/IP avec le numéro de port par défaut WebSphere MQ , de sorte que vous pouvez démarrer le programme d'écoute à l'aide de la commande suivante:

```
runmqlsr -t tcp
```

Pour tester l'exemple, vous pouvez également exécuter un programme pour lire les messages arrivant sur chaque serveur WebSphere MQ . Par exemple, vous pouvez utiliser l'exemple de programme amqstrg:

```
amqstrg SYSTEM.DEFAULT.LOCAL.QUEUE
```

L'exemple de programme utilise trois variables d'environnement, une obligatoire et deux facultatives. La variable requise est LDAP_BASEDN, qui indique le nom distinctif de base pour la recherche dans le répertoire. Pour utiliser les exemples de données, affectez la valeur `ou=Transport, o=MQuser`, par exemple, à l'invite de commande sur les systèmes Windows , entrez:

```
set LDAP_BASEDN=ou=Transport, o=MQuser
```

Les variables facultatives sont LDAP_HOST et LDAP_VERSION. La variable LDAP_HOST indique le nom de l'hôte sur lequel s'exécute le serveur LDAP. Elle correspond par défaut à l'hôte local s'il n'est pas spécifié. La variable LDAP_VERSION indique la version du protocole LDAP à utiliser et peut être 2 ou 3. La plupart des serveurs LDAP prennent désormais en charge la version 3 du protocole ; ils prennent tous en charge l'ancienne version 2. Cet exemple fonctionne également avec l'une ou l'autre version du protocole, et s'il n'est pas spécifié, il prend par défaut la version 2.

Vous pouvez maintenant exécuter l'exemple en entrant le nom du programme suivi du nom de l'application WebSphere MQ à laquelle vous souhaitez envoyer des messages, dans le cas des exemples de données, les noms d'application sont Londres, Sydney et Washington. Par exemple, pour envoyer des messages à l'application Londres:

```
amqsldpc London
```

Si le programme ne parvient pas à se connecter au serveur WebSphere MQ , un message d'erreur approprié s'affiche. Si la connexion aboutit, vous pouvez commencer à saisir des messages, chaque ligne que vous entrez (terminée par < return> ou < enter>) est envoyée en tant que message distinct, une ligne vide arrête le programme.

Conception du programme

Le programme comporte deux parties distinctes: la première utilise les variables d'environnement et la valeur de ligne de commande pour interroger un serveur d'annuaire LDAP ; la seconde établit la connexion WebSphere MQ à l'aide des informations renvoyées par l'annuaire et envoie les messages.

Les appels LDAP utilisés dans la première partie du programme diffèrent légèrement selon que LDAP version 2 ou 3 est utilisé, et ils sont décrits en détail par la documentation fournie avec les bibliothèques client LDAP. Cette section donne une brève description.

La première partie du programme vérifie qu'il a été appelé correctement et lit les variables d'environnement. Il établit ensuite une connexion avec le serveur d'annuaire LDAP sur l'hôte spécifié:

```
if (ldapVersion == LDAP_VERSION3)
{
  if ((ld = ldap_init(ldapHost, LDAP_PORT)) == NULL)
     ...
}
else
{
  if ((ld = ldap_open(ldapHost, LDAP_PORT)) == NULL )
     ...
}
```

Lorsqu'une connexion est établie, le programme définit certaines options sur le serveur à l'aide de l'appel "ldap_set_option", puis s'authentifie sur le serveur en s'y liant:

```
if (ldapVersion == LDAP_VERSION3)
{
  if (ldap_simple_bind_s(ld, bindDN, password) != LDAP_SUCCESS)
     ...
```

```
}
else
{
  if (ldap_bind_s(ld, bindDN, password, LDAP_AUTH_SIMPLE) !=
      LDAP_SUCCESS)
     ...
}
```

Dans l'exemple de programme `bindDN` et `password` sont définis sur NULL, ce qui signifie que le programme s'authentifie en tant qu'utilisateur anonyme, c'est-à-dire qu'il ne dispose pas de droits d'accès spéciaux et qu'il ne peut accéder qu'aux informations accessibles au public. Dans la pratique, la plupart des organisations limitent l'accès aux informations qu'elles stockent dans les annuaires afin que seuls les utilisateurs autorisés puissent y accéder.

Le premier paramètre de l'appel de liaison `ld` est un descripteur utilisé pour identifier cette session LDAP particulière dans le reste du programme. Après l'authentification, le programme recherche dans le répertoire les entrées qui correspondent au nom de l'application:

```
rc = ldap_search_s(ld,                 /* LDAP Handle             */
                   baseDN,             /* base distinguished name */
                   LDAP_SCOPE_ONELEVEL, /* one-level search        */
                   filterPattern,      /* filter search pattern   */
                   attrs,              /* attributes required     */
                   FALSE,              /* NOT attributes only     */
                   &ldapResult);       /* search result           */
```

Il s'agit d'un simple appel synchrone au serveur qui renvoie directement les résultats. Il existe d'autres types de recherche plus appropriés pour les requêtes complexes ou lorsqu'un grand nombre de résultats est attendu. Le premier paramètre de la recherche est le descripteur `ld` qui identifie la session. Le deuxième paramètre est le nom distinctif de base, qui indique où la recherche doit commencer dans l'annuaire, et le troisième paramètre est la portée de la recherche, c'est-à-dire les entrées relatives au point de départ. Ces deux paramètres définissent ensemble les entrées du répertoire dans lesquelles la recherche est effectuée. Le paramètre suivant, `filterPattern` , indique ce que nous recherchons. Le paramètre `attrs` répertorie les attributs que nous voulons récupérer à partir de l'objet lorsque nous l'avons trouvé. L'attribut suivant indique si nous voulons uniquement les attributs ou leurs valeurs également ; la valeur FALSE signifie que nous voulons les valeurs d'attribut. Le paramètre final est utilisé pour renvoyer le résultat.

Le résultat peut contenir de nombreuses entrées de répertoire, chacune avec les attributs spécifiés et leurs valeurs. Nous devons extraire les valeurs que nous voulons du résultat. Dans cet exemple de programme, nous nous attendons à ce qu'une seule entrée soit trouvée. Par conséquent, nous ne regardons que la première entrée du résultat:

```
ldapEntry = ldap_first_entry(ld, ldapResult);
```

Cet appel renvoie un descripteur qui représente la première entrée et nous définissons une boucle for pour extraire tous les attributs de l'entrée:

```
for (attribute = ldap_first_attribute(ld, ldapEntry, &ber);
     attribute != NULL;
     attribute = ldap_next_attribute(ld, ldapEntry, ber ))
{
```

Pour chacun de ces attributs, nous extrayons les valeurs qui lui sont associées. Là encore, nous n'attendons qu'une seule valeur par attribut, donc nous n'utilisons que la première valeur ; nous déterminons quel attribut nous avons et stockons la valeur dans la variable de programme appropriée:

```
values = ldap_get_values(ld, ldapEntry, attribute);
if (values != NULL && values[0] != NULL)
{
  if (stricmp(attribute, MQ_HOST_ATTR) == 0)
  {
```

```
    mqHost = strdup(values[0]);
    ...
```

Enfin, nous mettons en ordre la libération de la mémoire (ldap_value_free, ldap_memfree, ldap_msgfree) et la fermeture de la session en *annulant la liaison* à partir du serveur:

```
ldap_unbind(ld);
```

Nous vérifions que nous avons trouvé toutes les valeurs WebSphere MQ dont nous avons besoin dans le répertoire et, si tel est le cas, nous appelons sendMessages() pour nous connecter au serveur WebSphere MQ et envoyer les messages WebSphere MQ .

La deuxième partie de l'exemple de programme est la routine sendMessages() qui contient tous les appels WebSphere MQ . Il est modélisé sur l'exemple de programme amqsput0 , les différences étant que les paramètres du programme ont été étendus et que MQCONNX est utilisé à la place de l'appel MQCONN.

Développement d'applications pour IBM WebSphere 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 IBM WebSphere 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. Utilisez le démon pour périphériques IBM WebSphere MQ Telemetry pour simplifier la connexion de nombreux périphériques de petite taille à IBM WebSphere MQ.

Concepts associés

WebSphere MQ Telemetry

Concepts et scénarios de télémétrie pour la surveillance et le contrôle

Tâches associées

Installation de WebSphere MQ Telemetry

Administration de WebSphere MQ Telemetry

Identification et résolution des problèmes pour WebSphere MQ Telemetry

Référence associée

Référence WebSphere MQ Telemetry

IBM WebSphere MQ Telemetry exemples de programmes

Des exemples de script sont fournis pour illustrer l'utilisation de base de l'application client MQ Telemetry Transport v3 . Utilisez les scripts pour publier un message et vous abonner à une rubrique.

Avant de commencer

Démarrez le service de télémétrie (MQXR) pour exécuter les exemples de programmes.

L'ID utilisateur doit être membre du groupe d'utilisateurs mqm.

Exécutez d'abord le script SampleMQM , puis le script MQTTV3Sample pour effectuer une publication et un abonnement. Exécutez l'exemple de script CleanupMQM pour supprimer le gestionnaire de files d'attente créé par le script SampleMQM .

Comme le script SampleMQM crée et utilise un gestionnaire de files d'attente appelé QM1, ne s'exécute pas en l'état sur un système doté d'un gestionnaire de files d'attente QM1 . Tout changement peut avoir des implications sur la configuration du gestionnaire de files d'attente existant.

Pourquoi et quand exécuter cette tâche

- L'application SampleMQM crée et démarre un gestionnaire de files d'attente activé pour la télémétrie appelé QM1. Le script configure également une file d'attente de transmission par défaut pour QM1,

et crée et démarre un canal d'écoute par défaut sur le port 1883. Ce canal n'effectue aucune authentification des clients qui lui sont connectés. Le canal possède l'attribut d'identificateur utilisateur d'agent de canal de message (MCAUSER), défini sur'guest'sur les systèmes Windows ou sur'nobody'sur les systèmes Linux . Les clients connectés au canal sont traités comme l'utilisateur'guest'ou l'utilisateur'nobody', en fonction du système sur lequel il s'exécute. Le script autorise'guest'sur les systèmes Windows et'nobody'sur les systèmes Linux à pouvoir publier et s'abonner à n'importe quelle rubrique sur QM1

- L'application MQTTV3Sample est conservée à l'emplacement suivant:
  - Sous Windows *MQ_INSTALLATION_PATH*\mqxr\samples

    où *MQ_INSTALLATION_PATH* est l'emplacement dans lequel IBM WebSphere MQ est installé.
  - Sous Linux *MQ_INSTALLATION_PATH*/mqxr/samples

  L'application MQTTV3Sample fonctionne en tant que diffuseur de publications et envoie un message unique à une rubrique sur le serveur. Il agit également en tant qu'abonné, en écoutant les messages du serveur.
- L'exemple de script CleanupMQM se termine et supprime QM1 qui a été créé par le script SampleMQM . Utilisez l'exemple de script CleanupMQM si vous souhaitez réexécuter le script SampleMQM ou supprimer QM1.

Procédure

1. Tapez la commande suivante sur une ligne de commande pour exécuter le script SampleMQM.
   - Sous Windows la commande d'exécution d'un script SampleMQM est la suivante :

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\SampleMQM.bat
     ```
   - Sous AIX et Linux, la commande permettant d'exécuter le script SampleMQM est la suivante :

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/SampleMQM.sh
     ```

   où *MQ_INSTALLATION_PATH* est l'emplacement dans lequel IBM WebSphere MQ est installé.

   Le gestionnaire de files d'attente MQXR_SAMPLE_QM est créé.
2. Tapez la commande suivante pour exécuter la première partie du script MQTTV3Sample ;
   - Sous Windows, sur une ligne de commande, tapez la commande suivante :

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -a subscribe
     ```
   - Sous AIX et Linux, dans une fenêtre shell, entrez la commande suivante:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -a subscribe
     ```
3. Tapez la commande suivante pour exécuter la deuxième partie du script MQTTV3Sample ;
   - Sous Windows, sur une autre ligne de commande, tapez la commande suivante :

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\RunMQTTV3Sample.bat -m "Hello from an MQTT v3
     application"
     ```
   - Sous AIX et Linux, dans une autre fenêtre shell, entrez la commande suivante:

     ```
     MQ_INSTALLATION_PATH/mqxr/samples/RunMQTTV3Sample.sh -m "Hello from an MQTT v3 application"
     ```
4. Pour supprimer le gestionnaire de files d'attente créé par le script SampleMQM, vous pouvez exécuter le script CleanupMQM à l'aide de la commande suivante ;
   - Sous Windows, tapez la commande suivante :

     ```
     MQ_INSTALLATION_PATH\mqxr\samples\CleanupMQM.bat
     ```

- Sous AIX et Linux dans une autre fenêtre shell, entrez la commande suivante:

```
MQ_INSTALLATION_PATH/mqxr/samples/CleanupMQM.sh
```

Résultats

Le message `Hello from an MQTT v3 application` que vous avez entré dans la deuxième fenêtre sera publié par cette application et reçu par l'application dans la première fenêtre. L'application dans la première fenêtre l'affiche à l'écran.

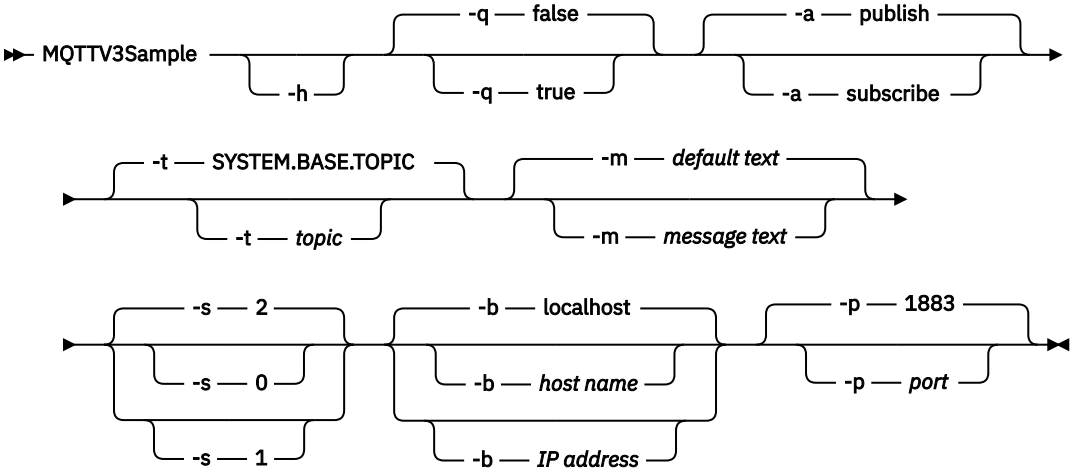
Programme MQTTV3Sample

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.

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:

```
runMQTTV3Sample -a subscribe
```

Pour publier un message sous Windows, utilisez la commande suivante:

```
runMQTTV3Sample
```

Pour plus d'informations sur l'exécution des exemples de script fournis, voir «IBM WebSphere MQ Telemetry exemples de programmes», à la page 491.

Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java

Les étapes de création d'une application client MQTT sont décrites dans un tutoriel. Chaque ligne de code est expliquée. A la fin de la tâche, vous aurez créé un diffuseur de publications MQTT. Vous pouvez parcourir les publications à l'aide de WebSphere MQ Explorer.

Avant de commencer

Installez la fonction WebSphere MQ Telemetry sur un serveur sur lequel IBM WebSphere MQ Version 7.1 ou version ultérieure est installé.

L'application client utilise le package `com.ibm.mq.micro.client.mqttv3` dans IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Le SDK fait partie de l'installation d' IBM WebSphere MQ Telemetry . Le client se connecte à la fonction IBM WebSphere MQ Telemetry pour échanger des messages avec IBM WebSphere MQ.

Vous devez également installer les mises à jour de télémétrie pour IBM WebSphere MQ Explorer Version 7.1 afin d'administrer IBM WebSphere MQ Telemetry. Les mises à jour font partie de l'installation d' IBM WebSphere MQ Telemetry .

Un client MQTT, exécuté sur Java SE, requiert la version 6.0 de Java SE ou ultérieure. IBM Java SE v6.0 fait partie de l'installation IBM WebSphere MQ Version 7.1 . Il se trouve à l'emplacement *WebSphere MQ installation directory*`\java\jre`

Pourquoi et quand exécuter cette tâche

L'exemple est une application de publication, PubSync. PubSync publie `Hello World` sur la rubrique `MQTT Examples`et attend la confirmation que la publication a été distribuée au gestionnaire de files d'attente.

En configurant un abonnement durable à `MQTT Examples` , vous pouvez vérifier que l'application fonctionne.

La procédure utilise Eclipse pour développer, générer et exécuter le client. Vous pouvez télécharger Eclipse depuis le site Web du projet Eclipse à l'adresse www.eclipse.org.

Pour créer l'application, vous pouvez créer les fichiers Java, puis les compiler et les exécuter à l'aide de la ligne de commande.

Dans un nouveau répertoire, créez le chemin de répertoire `.\com\ibm\mq\id`. Créez deux fichiers Java, `Example.java` et PubSync`.java`. Copiez le code de «Exemple de code», à la page 498 dans les fichiers Java.

Compilez le code Java à l'aide de la commande suivante:

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubSync.java com.ibm.mq.id.Example.java
```

Exécutez PubSync à l'aide de la commande suivante:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubSync
```

Procédure

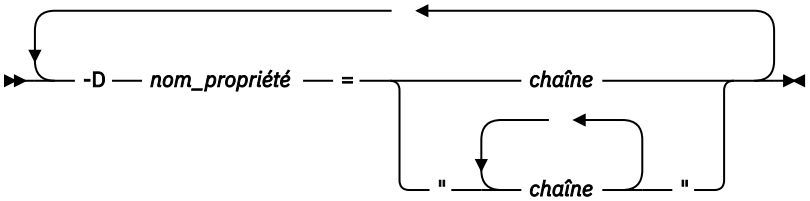
1. Créez un projet Java dans Eclipse.
   a) **Fichier > Nouveau > Projet Java** et entrez un nom de projet. Cliquez sur **Suivant**.

      Vérifiez que la version de l'environnement d'exécution Java est correcte ou ultérieure. Java SE doit être à la version 6.0 ou ultérieure.
   b) Sur la page Paramètres Java, cliquez sur **Bibliothèques > Ajouter des fichiers JAR externes ...**
   c) Accédez au répertoire dans lequel vous avez installé le dossier SDK WebSphere MQ Telemetry. Localisez le dossier SDK\clients\java et sélectionnez tous les fichiers .jar > **> Ouvrir > Terminer**.
2. Installez le Javadoc du client MQTT.

   Avec le Javadoc du client MQTT installé, l'éditeur Java fournit une assistance avec les classes MQTT v3 .
   a) Dans votre projet Java, ouvrez **Explorateur de packages > Bibliothèques référencées**. Cliquez avec le bouton droit de la souris sur com.ibm.micro.client.mqttv3.jar **> Propriétés**.
   b) Dans le navigateur des propriétés, cliquez sur **Emplacement Javadoc**.
   c) Dans la page Emplacement Javadoc, cliquez sur **URL Javadoc > Parcourir ...** et recherchez le dossier *WMQ Installation directory*\mqxr\SDK\clients\java\doc\javadoc **> OK**.
   d) Cliquez sur **Valider ... > OK**

      Vous êtes invité à ouvrir le navigateur pour afficher la documentation.
3. Créez la classe, PubSync, à l'aide de l'assistant Classe Java.
   a) Cliquez avec le bouton droit de la souris sur le projet Java que vous avez créé > **Nouveau** > **Classe**.
   b) Entrez le nom du package, com.ibm.mq.id
   c) Entrez le nom de classe, PubSync
   d) Cochez la case du module de remplacement de méthode, **public static void main (String [ ] args)**
4. Créez un fichier, Example.java , dans le package com.ibm.mq.id. Copiez le code de Figure 89, à la page 500 dans le fichier.

   Tous les paramètres utilisés dans les exemples sont définis en tant que propriétés. Vous pouvez remplacer les valeurs en modifiant les valeurs par défaut dans Example.javaou en fournissant les propriétés sous forme d'options sur la ligne de commande Java à l'aide du paramètre -D :



   L'identificateur client utilisé dans cet exemple, et les exemples «Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java», à la page 500 , sont un nom d'utilisateur suffixé par une chaîne aléatoire.
5. Suivez les étapes pour créer le code ou copiez le code à partir de Figure 88, à la page 499.

   Les étapes qui suivent expliquent le code dans Pubsync.java.
6. Créez un bloc try-catch .

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

Le client MQTT émet `MqttException`, `MqttPersistenceException` ou `MqttSecurityException`. `MqttPersistenceException` et `MqttSecurityException` sont des sous-classes de `MqttException`.

Utilisez la méthode `MqttException.getReasonCode` pour déterminer la raison de l'exception. Si une exception `MqttPersistenceException` ou `MqttSecurityException` est émise, utilisez la méthode `getCause` pour renvoyer l'exception throwable sous-jacente.

7. Créez une nouvelle instance `MqttClient` .

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

Indiquez au client une adresse de serveur, qui sera utilisée ultérieurement pour la connexion à WebSphere MQ. Définissez l'identificateur de client pour nommer le client.

- Vous pouvez éventuellement fournir une implémentation de l'interface `MqttClientPersistence` pour remplacer l'implémentation par défaut. L'implémentation `MqttPersistence` par défaut stocke les messages QoS 1 et 2 en attente de distribution sous forme de fichiers ; voir «Persistance des messages dans les clients MQTT», à la page 554.
- Le port TCP/IP IBM WebSphere MQ par défaut pour MQTT est 1883. Pour SSL, il s'agit de 8883. Dans l'exemple, l'adresse par défaut est `tcp://localhost:1883`.
- En règle générale, il est important de pouvoir identifier un client physique spécifique à l'aide de l'identificateur de client. L'identificateur de client doit être unique sur tous les clients qui se connectent à un serveur ; voir «Identificateur de client», à la page 550. L'utilisation du même identificateur de client qu'une instance précédente indique que l'instance présente est une instance du même client. Si vous dupliquez un identificateur de client dans deux clients en cours d'exécution, une exception est émise dans les deux clients et un client s'arrête.
- La longueur de l'identificateur client est limitée à 23 octets. Une exception est émise si la longueur est dépassée. L'identificateur de client ne doit contenir que des caractères autorisés dans un nom de gestionnaire de files d'attente ; par exemple, aucun trait d'union ni espace.
- Tant que vous n'appelez pas la méthode `MqttClient.connect` , aucun traitement de message n'a lieu.

Utilisez l'objet client pour publier et s'abonner à des rubriques et récupérer des informations sur les publications qui n'ont pas encore été distribuées.

8. Créez une rubrique pour la publication.

```
 MqttTopic topic = client.getTopic(Example.topicString);
```

Une chaîne de rubrique est limitée à 64 ko, ce qui dépasse la longueur maximale d'une chaîne de rubrique IBM WebSphere MQ . Sinon, une chaîne de rubrique suit les mêmes règles que les chaînes de rubrique WebSphere MQ ; voir Chaînes de rubrique. L'exemple définit la chaîne de rubrique `MQTT Examples`.

9. Créez un message de publication.

```
MqttMessage message = new MqttMessage(Example.publication.getBytes());
```

La chaîne `"Hello World"` est convertie en tableau d'octets et utilisée pour créer un `MqttMessage`.

- Une charge de message MQTT est toujours un tableau d'octets. La méthode `getBytes` convertit un objet chaîne en UTF-8. La méthode `MqttMessage` est pratique `toString` pour renvoyer la charge de message sous forme de chaîne. Elle est équivalente à `new string(message.getPayload)`
- Un message de publication est envoyé au gestionnaire de files d'attente avec un en-tête RFH2 et les données de message sont envoyées en tant que message `jms-bytes` .

- L'objet message possède des attributs de qualité de service et des attributs conservés. La qualité de service (QoS) détermine la fiabilité du transfert du message entre le client MQTT et le gestionnaire de files d'attente ; voir «Qualités de service fournies par un client MQTT», à la page 559. L'attribut conservé contrôle si une publication est stockée par le gestionnaire de files d'attente pour les futurs abonnés. Si une publication n'est pas conservée, elle est envoyée uniquement aux abonnés actuels ; voir «Publications conservées et clients MQTT», à la page 561. Les paramètres `MqttMessage` par défaut sont les suivants: "Les messages sont distribués au moins une fois et ne sont pas conservés."

10. Connectez-vous au serveur.

    ```
    client.connect();
    ```

    L'exemple se connecte au serveur à l'aide des options de connexion par défaut. Une fois que vous vous êtes connecté, vous pouvez commencer la publication. Les options de connexion par défaut sont les suivantes:

    - Un petit message "keep-alive" est envoyé toutes les 15 secondes pour empêcher la fermeture de la connexion TCP/IP.
    - La session est démarrée sans vérification de l'achèvement des publications précédentes.
    - L'intervalle entre deux tentatives d'envoi d'un message est de 15 secondes.
    - Aucun message de dernière volonté et testament n'est créé pour la connexion.
    - La SocketFactory standard est utilisée pour créer la connexion.

    Modifiez les options de connexion en créant un objet `ConnectionOptions` et en le transmettant en tant que paramètre supplémentaire à `client.connect`.

11. Publier.

    ```
    MqttDeliveryToken token = topic.publish(message);
    ```

    L'exemple envoie la publication "Hello World" sur la rubrique "Exemples MQTT" au gestionnaire de files d'attente.

    - Lorsque la méthode `publish` est renvoyée, le message est transféré en toute sécurité au client MQTT, mais pas encore au serveur. Si le message comporte QoS 1 ou 2, le message est stocké en local, au cas où le client échouerait avant la fin de la distribution.
    - `publish` renvoie un jeton de distribution, qui est utilisé pour vérifier si un accusé de réception a déjà été reçu du serveur.

12. Attendez l'accusé de réception du serveur.

    ```
    token.waitForCompletion(Example.timeout);
    ```

    L'exemple PubSync attend un accusé de réception du serveur, qui confirme que le message a été distribué.

    - Sans le délai d'attente, le client attendrait indéfiniment. La tâche «Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java», à la page 500 montre comment recevoir des accusés de réception sans attendre à l'aide d'un objet de rappel.

13. Déconnectez le client du serveur.

    ```
    client.disconnect();
    ```

    Le client se déconnecte du serveur et attend la fin des méthodes `MqttCallback` en cours d'exécution. Il attend ensuite jusqu'à 30 secondes pour terminer tout travail restant. Vous pouvez spécifier un délai de mise au repos comme paramètre supplémentaire.

14. Sauvegardez les modifications apportées à `PubSync.java` et `Example.Java`

    Eclipse compile automatiquement Java. Vous êtes maintenant prêt à voir les résultats en exécutant le programme.

Résultats

Pour afficher les publications à l'aide de WebSphere MQ, créez une rubrique, une file d'attente et un abonnement durable, tous appelés "MQTTExampleTopic" à l'aide du script dans Figure 87, à la page 498. Exécutez le client pour publier dans la rubrique MQTT Examples , puis exécutez l'exemple de programme **amqsbcg** pour parcourir les publications dans la file d'attente MQTTExamples .

1. Démarrez un gestionnaire de files d'attente et démarrez son service de télémétrie (MQXR) en cours d'exécution. Vérifiez que l'adresse TCP/IP et le port configurés pour le canal de télémétrie correspondent aux valeurs que vous utilisez dans l'application MQTT.
2. Configurez un abonnement durable en créant le script de commandes mqttexamples.txt et en l'exécutant à l'aide de **runmqsc:**

   ```
   DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
   DEFINE QLOCAL('MQTTExampleQueue') REPLACE
   DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
   ```

   Figure 87. mqttExampleTopic.txt

   Pour exécuter le script sous Windows, entrez la commande suivante:

   ```
   runmqsc queue manager name < mqttExampleTopic.txt
   ```

3. Exécutez le client en tant qu'application Java depuis Eclipseou en exécutant Java dans une fenêtre de commande:

   ```
   java -cp jar_dir\com.mq.micro.client.mqttv3.jar
         com.ibm.mq.id.classname.class
   ```

   Remarque : La fenêtre de commande doit être ouverte dans le répertoire contenant le chemin com\ibm\mq\id.
4. Parcourez les résultats à l'aide de WebSphere MQ Explorer ou exécutez la commande suivante:

   ```
   amqsbcg MQTTExampleQueue queue manager name
   ```

Exemple de code

PubSync.java est une liste complète du code décrit dans Procédure. Modifiez la classe Example dans Figure 89, à la page 500 pour remplacer les paramètres par défaut utilisés dans PubSync.java.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      client.connect();
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
          + " seconds for publication of \"" + message.toString()
          + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName()
          + "\" for client instance: \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 88. PubSync.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figure 89. Example.java

Concepts associés

Applications de publication/abonnement MQTT

Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java

Dans cette tâche, vous suivez un tutoriel pour modifier votre première application de publication. Les modifications permettent à l'application d'envoyer des publications sans attendre les accusés de réception de distribution. Les accusés de réception de distribution sont reçus par une classe de rappel que vous créez.

Avant de commencer

Installez la fonction WebSphere MQ Telemetry sur un serveur sur lequel IBM WebSphere MQ Version 7.1 ou version ultérieure est installé.

L'application client utilise le package `com.ibm.mq.micro.client.mqttv3` dans IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Le SDK fait partie de l'installation d' IBM WebSphere

MQ Telemetry . Le client se connecte à la fonction IBM WebSphere MQ Telemetry pour échanger des messages avec IBM WebSphere MQ.

Vous devez également installer les mises à jour de télémétrie pour IBM WebSphere MQ Explorer Version 7.1 afin d'administrer IBM WebSphere MQ Telemetry. Les mises à jour font partie de l'installation d' IBM WebSphere MQ Telemetry .

Un client MQTT, exécuté sur Java SE, requiert la version 6.0 de Java SE ou ultérieure. IBM Java SE v6.0 fait partie de l'installation IBM WebSphere MQ Version 7.1 . Il se trouve à l'emplacement *WebSphere MQ installation directory*\java\jre

Pourquoi et quand exécuter cette tâche

L'exemple est une application de publication, PubAsync. PubAsync publie `Hello World` sur la rubrique `MQTT Examples`, sans attendre la confirmation que la publication a été distribuée au gestionnaire de files d'attente. Les accusés de réception de distribution sont reçus dans une classe de rappel, `CallBack`.

En configurant un abonnement durable à `MQTT Examples` , vous pouvez vérifier que l'application fonctionne.

La procédure utilise Eclipse pour développer, générer et exécuter le client. Vous pouvez télécharger Eclipse depuis le site Web du projet Eclipse à l'adresse www.eclipse.org.

Les étapes de la procédure modifient l'application PubSync`.java` dans «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494.

Vous pouvez également copier le code, «Exemple de code», à la page 503, dans un nouveau répertoire `.\com\ibm\mq\id`. Créez trois fichiers Java, `Example.java`, `CallBack.javaet` PubAsync`.java`. Compilez les exemples à l'aide de la commande,

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsync.java com.ibm.mq.id.CallBack.java com.ibm.mq.id.Example.java
```

Exécutez PubAsync à l'aide de la commande suivante:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsync.class
```

Procédure

1. Dans le package `com.ibm.mq.id` , créez un fichier, `CallBack.java`. Copiez le code de Figure 92, à la page 504 dans le fichier.

   `Callback.java` implémente l'interface `MqttCallBack` . Dans l'exemple, un constructeur supplémentaire initialise le rappel avec des données d'instance.
2. Dans le package `com.ibm.mq.id` , cliquez avec le bouton droit de la souris sur PubSync`.java` et copiez-le. Collez-le dans le même package, puis renommez-le en PubAsync.
3. Juste avant la ligne de code `client.connect();` , instanciez la classe `CallBack` en transmettant l'identificateur de client.

   ```
   CallBack callback = new CallBack(Example.clientId);
   client.setCallback(callback);
   ```

   - La classe `CallBack` implémente `MqttCallBack`. Une instance de rappel est requise, par identificateur de client. Dans cet exemple, le constructeur transmet l'identificateur de client à enregistrer en tant que données d'instance. Il est utilisé dans le rappel pour identifier l'instance du rappel qui a été démarrée.
   - Vous devez implémenter trois méthodes dans la classe de rappel:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Reçoit une publication à laquelle un abonnement a été souscrit.

public void connectionLost(Throwable cause)
Appelé lorsque la connexion est perdue.

public void deliveryComplete(MqttDeliveryToken token))
Appelé lorsqu'un jeton de distribution est reçu pour un message QoS 1 ou 2 qui a été publié.

- Le rappel est activé par `MqttClient.connect`.

4. Déconnecter le client
   a) Supprimez l'instruction contenant l'expression `token.waitForCompletion` .
   L'unité d'exécution principale se poursuit sans attendre la distribution de la publication.
   b) Vérifiez si le client est déjà déconnecté.
   Le client MQTT se déconnecte suite à une erreur renvoyée à la méthode `lostConnection` dans `MqttCallback`, ou l'application client peut se déconnecter. Testez pour voir s'il existe une connexion ouverte.
   c) Utilisez la constante, `Example.quiesceTimeout`, pour définir la durée maximale de mise au repos du client.

```
if (client.isConnected())
    client.disconnect(Example.quiesceTimeout);
```

Le client se termine lorsqu'une combinaison des trois conditions suivantes est vérifiée:

a. Le rappel a été appelé pour tous les messages qui ont été publiés dans cette session, ou si la session a été redémarrée, dans les sessions précédentes.
b. Les messages sont en cours et l'intervalle de mise au repos a expiré. Par défaut, l'intervalle de mise au repos est de 30 secondes. Vous pouvez modifier le délai de mise au repos en transmettant le nombre de millisecondes à attendre comme paramètre de `client.disconnect`.
c. `client.disconnect` a été appelé après la publication et la mise en file d'attente de certains messages par le client, mais avant leur envoi. Les messages en file d'attente ne sont pas encore en cours. Si la session est redémarrable, les messages sont renvoyés lorsque la session redémarre.

Résultats

Pour afficher les publications à l'aide de WebSphere MQ, créez une rubrique, une file d'attente et un abonnement durable, tous appelés `"MQTTExampleTopic"` à l'aide du script dans Figure 90, à la page 502. Exécutez le client pour publier dans la rubrique `MQTT Examples` , puis exécutez l'exemple de programme **amqsbcg** pour parcourir les publications dans la file d'attente `MQTTExamples` .

1. Démarrez un gestionnaire de files d'attente et démarrez son service de télémétrie (MQXR) en cours d'exécution. Vérifiez que l'adresse TCP/IP et le port configurés pour le canal de télémétrie correspondent aux valeurs que vous utilisez dans l'application MQTT.
2. Configurez un abonnement durable en créant le script de commandes `mqttexamples.txt` et en l'exécutant à l'aide de **runmqsc:**

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Figure 90. mqttExampleTopic.txt

Pour exécuter le script sous Windows, entrez la commande suivante:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Exécutez le client en tant qu'application Java depuis Eclipseou en exécutant Java dans une fenêtre de commande:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Remarque : La fenêtre de commande doit être ouverte dans le répertoire contenant le chemin `com\ibm\mq\id`.

4. Parcourez les résultats à l'aide de WebSphere MQ Explorer ou exécutez la commande suivante:

```
amqsbcg MQTTExampleQueue queue manager name
```

Exemple de code

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubAsync {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      client.connect();
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 91. PubAsync.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 92. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figure 93. Example.java

Création d'un diffuseur de publications asynchrone récupérable pour MQ Telemetry Transport à l'aide de Java

Dans cette tâche, vous suivez un tutoriel pour modifier votre application de publication asynchrone. Les modifications permettent à l'application de terminer la distribution des publications qui n'ont pas fait l'objet d'un accusé de réception lors de la dernière exécution du client.

Avant de commencer

Installez la fonction WebSphere MQ Telemetry sur un serveur sur lequel IBM WebSphere MQ Version 7.1 ou version ultérieure est installé.

L'application client utilise le package `com.ibm.mq.micro.client.mqttv3` dans IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Le SDK fait partie de l'installation d' IBM WebSphere MQ Telemetry . Le client se connecte à la fonction IBM WebSphere MQ Telemetry pour échanger des messages avec IBM WebSphere MQ.

Vous devez également installer les mises à jour de télémétrie pour IBM WebSphere MQ Explorer Version 7.1 afin d'administrer IBM WebSphere MQ Telemetry. Les mises à jour font partie de l'installation d' IBM WebSphere MQ Telemetry .

Un client MQTT, exécuté sur Java SE, requiert la version 6.0 de Java SE ou ultérieure. IBM Java SE v6.0 fait partie de l'installation IBM WebSphere MQ Version 7.1 . Il se trouve à l'emplacement *WebSphere MQ installation directory*\java\jre

Pourquoi et quand exécuter cette tâche

L'exemple est une application de publication, PubAsyncRestartable. PubAsyncRestartable publie Hello World sur la rubrique MQTT Examples, sans attendre la confirmation que la publication a été distribuée au gestionnaire de files d'attente. Les accusés de réception de distribution sont reçus dans une classe de rappel, CallBack. Tous les jetons de distribution pour les publications qui n'ont pas été terminées dans une instance précédente peuvent être examinés. Ils sont également traités par la classe de rappel.

En configurant un abonnement durable à MQTT Examples , vous pouvez vérifier que l'application fonctionne.

La procédure utilise Eclipse pour développer, générer et exécuter le client. Vous pouvez télécharger Eclipse depuis le site Web du projet Eclipse à l'adresse www.eclipse.org.

Les étapes de la procédure modifient l'application PubAsync.java dans «Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java», à la page 500.

Vous pouvez également copier le code, «Exemple de code», à la page 509, dans un nouveau répertoire .\com\ibm\mq\id. Créez trois fichiers Java, Example.java, CallBack.javaet PubAsyncRestartable.java. Compilez les exemples à l'aide de la commande,

```
javac -cp jar_dir\com.mq.micro.client.mqttv3.jar
       com.ibm.mq.id.PubAsyncRestartable.java com.ibm.mq.id.CallBack.java
com.ibm.mq.id.Example.java
```

Exécutez PubAsyncRestartable à l'aide de la commande suivante:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.PubAsyncRestartable.class
```

Procédure

1. Dans le package com.ibm.mq.id , cliquez avec le bouton droit de la souris sur PubAsync.java et copiez-le. Collez-le dans le même package, puis renommez-le en PubAsyncRestartable.
2. Créez un identificateur de client réutilisable.

   ```
   Example.clientId = String.format(
       "%-23.23s",
       (System.getProperty("user.name") + "_" + (System.getProperty(
           "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
   ```

   Figure 94. Identificateur de client réutilisable

   Les applications dans «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 et «Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java», à la page 500 ont utilisé un nouvel identificateur de client pour chaque connexion client. Pour un diffuseur de publications ou un abonné redémarrable, vous devez utiliser le même identificateur de client chaque fois que le client est connecté, mais des clients différents doivent utiliser des identificateurs différents ; voir «Identificateur de client», à la page 550. L'identificateur de client réutilisable est généré à partir du nom d'utilisateur et du nom de la classe. Sa longueur est limitée à 23 octets. Il ne doit comporter que des caractères valides dans les noms d'objet de gestionnaire de files d'attente. Le code supprime les traits d'union qui ont pu être insérés.

3. La valeur QoS du message est définie sur 2 au lieu de la valeur par défaut, 1, pour éviter les messages en double.

```
message.setQos(Example.QoS);
```

   Vous devez remplacer la valeur de `Example.QoS` par 2ou transmettre la propriété `QoS` en tant qu'argument à l'aide de l'option `-DQoS=2` sur la ligne de commande Java.

4. Créez un objet `MqttConnectOptions` et définissez son attribut `cleanSession` sur `false`.

   a) Créez un objet `MqttConnectOptions` .

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

      `conOptions` est un paramètre d'option sur le constructeur `MqttClient` .

   b) Définissez l'attribut clearSession .

```
conOptions.setCleanSession(Example.cleanSession);
```

      Par défaut, le paramètre `Example.cleanSession` est défini sur `true`, ce qui correspond à la valeur par défaut de `MqttConnectionOptions.cleanSession`.

   Lorsque `PubAsyncRestartable` redémarre, il peut démarrer avec une "session propre" et effacer tous les jetons de distribution en attente pour les messages de QoS 1 ou 2.

   Définissez `Example.cleanSession` sur `false` pour conserver tous les jetons de distribution en attente. Les jetons sont traités par la classe `MqttCallBack` lorsque le client est à nouveau connecté.

5. Si la session est en cours de redémarrage, extrayez les jetons de distribution en attente et imprimez leur contenu.

```
if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance "
            + client.getClientId());
```

6. Transmettez le paramètre `conOptions` au constructeur `MqttClient` .

```
client.connect(conOptions);
```

7. Lors de la déconnexion, définissez un intervalle de déconnexion maximal.

```
client.disconnect(Example.timeout);
```

   Pour pouvoir afficher les jetons de distribution en attente en cours de traitement, une instance précédente doit se terminer sans terminer la distribution. Pour exécuter l'exemple avec la possibilité de ne pas accuser réception des publications avant la fin de `PubAsyncRestartable` , définissez `Example.timeout` sur 0.

Résultats

Pour afficher les publications à l'aide de WebSphere MQ, créez une rubrique, une file d'attente et un abonnement durable, tous appelés `"MQTTExampleTopic"` à l'aide du script dans Figure 95, à la page 508. Exécutez le client pour publier dans la rubrique `MQTT Examples` , puis exécutez l'exemple de programme **amqsbcg** pour parcourir les publications dans la file d'attente `MQTTExamples` .

1. Démarrez un gestionnaire de files d'attente et démarrez son service de télémétrie (MQXR) en cours d'exécution. Vérifiez que l'adresse TCP/IP et le port configurés pour le canal de télémétrie correspondent aux valeurs que vous utilisez dans l'application MQTT.
2. Configurez un abonnement durable en créant le script de commandes `mqttexamples.txt` et en l'exécutant à l'aide de **runmqsc:**

```
DEFINE TOPIC('MQTTExampleTopic') TOPICSTR('MQTT Example') REPLACE
DEFINE QLOCAL('MQTTExampleQueue') REPLACE
DEFINE SUB('MQTTExampleSub') DEST('MQTTExampleQueue') TOPICOBJ('MQTTExampleTopic') REPLACE
```

Figure 95. mqttExampleTopic.txt

Pour exécuter le script sous Windows, entrez la commande suivante:

```
runmqsc queue manager name < mqttExampleTopic.txt
```

3. Exécutez le client en tant qu'application Java depuis Eclipseou en exécutant Java dans une fenêtre de commande:

```
java -cp jar_dir\com.mq.micro.client.mqttv3.jar
      com.ibm.mq.id.classname.class
```

Remarque : La fenêtre de commande doit être ouverte dans le répertoire contenant le chemin `com\ibm\mq\id`.

4. Parcourez les résultats à l'aide de WebSphere MQ Explorer ou exécutez la commande suivante:

```
amqsbcg MQTTExampleQueue queue manager name
```

Exemple de code

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
import com.ibm.micro.client.mqttv3.MqttDeliveryToken;
import com.ibm.micro.client.mqttv3.MqttMessage;
import com.ibm.micro.client.mqttv3.MqttTopic;
public class PubAsyncRestartable {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + (System.getProperty(
            "clientId", "PubAsyncRestartable."))).trim()).replace('-', '_');
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      if (!conOptions.isCleanSession()) {
        MqttDeliveryToken tokens[] = client.getPendingDeliveryTokens();
        System.out.println("Starting a previous session for instance \""
            + client.getClientId() + "\" with " + tokens.length
            + " delivery tokens pending");
        for (int i = 0; i < tokens.length; i++) {
          System.out.println("Message \"" + tokens[i].getMessage().toString()
              + "\" with QoS=" + tokens[i].getMessage().getQos()
              + " recovered by instance \"" + client.getClientId()
              + "\" and assigned delivery token \"" + tokens[i].hashCode()
              + "\"");
        }
      } else
        System.out.println("Starting a clean session for instance \""
            + client.getClientId() + "\"");
      client.connect(conOptions);
      System.out.println("Publishing \"" + message.toString()
          + "\" on topic \"" + topic.getName() + "\" with QoS = "
          + message.getQos());
      System.out.println("For client instance \"" + client.getClientId()
          + "\" on address " + client.getServerURI() + "\"");
      MqttDeliveryToken token = topic.publish(message);
      System.out.println("With delivery token \"" + token.hashCode()
          + " delivered: " + token.isComplete());
      if (client.isConnected())
          client.disconnect(Example.quiesceTimeout);
      System.out.println("Disconnected: delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 96. PubAsyncRestartable.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 97. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean          retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figure 98. Example.java

Création d'un abonné pour MQ Telemetry Transport à l'aide de Java

Dans cette tâche, vous suivez un tutoriel pour créer une application d'abonné. L'abonné crée un abonnement à une rubrique et reçoit des publications pour l'abonnement.

Avant de commencer

Installez la fonction WebSphere MQ Telemetry sur un serveur sur lequel IBM WebSphere MQ Version 7.1 ou version ultérieure est installé.

L'application client utilise le package `com.ibm.mq.micro.client.mqttv3` dans IBM WebSphere MQ Telemetry Software Development Toolkit (SDK). Le SDK fait partie de l'installation d' IBM WebSphere MQ Telemetry . Le client se connecte à la fonction IBM WebSphere MQ Telemetry pour échanger des messages avec IBM WebSphere MQ.

Vous devez également installer les mises à jour de télémétrie pour IBM WebSphere MQ Explorer Version 7.1 afin d'administrer IBM WebSphere MQ Telemetry. Les mises à jour font partie de l'installation d' IBM WebSphere MQ Telemetry .

Un client MQTT, exécuté sur Java SE, requiert la version 6.0 de Java SE ou ultérieure. IBM Java SE v6.0 fait partie de l'installation IBM WebSphere MQ Version 7.1 . Il se trouve à l'emplacement *WebSphere MQ installation directory*\java\jre

Pourquoi et quand exécuter cette tâche

L'exemple est une application d'abonné, Subscribe. Subscribe crée une rubrique d'abonnement, MQTT Examples, et attend les publications sur l'abonnement pendant 30 secondes.

Un abonné peut créer un abonnement et attendre des publications. Il peut également recevoir des publications envoyées à un abonnement créé précédemment, pour le même identificateur de client. L'attribut booléen MqttConnectionOptions.cleanSession contrôle si les publications envoyées précédemment sont reçues ou non ; voir «Abonnements», à la page 562.

Vous pouvez utiliser les exemples de programmes de publication pour créer des publications ou utiliser l'explorateur WebSphere MQ pour créer une publication de test sur la rubrique MQTT Examples .

La procédure utilise Eclipse pour développer, générer et exécuter le client. Vous pouvez télécharger Eclipse depuis le site Web du projet Eclipse à l'adresse www.eclipse.org.

Les instructions de la procédure supposent que vous avez déjà créé le package com.ibm.mq.id dans l'une des tâches précédentes et que vous l'avez copié dans les classes Example.java et Callback.java .

Procédure

1. Créez la classe Subscribe dans le package com.ibm.mq.id.
2. Créez un identificateur de client réutilisable.

```
Example.clientId = String.format(
    "%-23.23s",
    (System.getProperty("user.name") + "_" + (System.getProperty(
        "clientId", "Subscribe."))).trim()).replace('-', '_');
```

Figure 99. Identificateur de client réutilisable

Les applications dans «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 et «Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de Java», à la page 500 ont utilisé un nouvel identificateur de client pour chaque connexion client. Pour un diffuseur de publications ou un abonné redémarrable, vous devez utiliser le même identificateur de client chaque fois que le client est connecté, mais des clients différents doivent utiliser des identificateurs différents ; voir «Identificateur de client», à la page 550. L'identificateur de client réutilisable est généré à partir du nom d'utilisateur et du nom de la classe. Sa longueur est limitée à 23 octets. Il ne doit comporter que des caractères valides dans les noms d'objet de gestionnaire de files d'attente. Le code supprime les traits d'union qui ont pu être insérés.

3. Créez un bloc try-catch .

```
try { ...
 } catch (Exception e) {
  e.printStackTrace();
}
```

Le client MQTT émet MqttException, MqttPersistenceException ou MqttSecurityException. MqttPersistenceException et MqttSecurityException sont des sous-classes de MqttException.

Utilisez la méthode MqttException.getReasonCode pour déterminer la raison de l'exception. Si une exception MqttPersistenceException ou MqttSecurityException est émise, utilisez la méthode getCause pour renvoyer l'exception throwable sous-jacente.

4. Créez une nouvelle instance MqttClient .

```
MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
```

Indiquez au client une adresse de serveur, qui sera utilisée ultérieurement pour la connexion à WebSphere MQ. Définissez l'identificateur de client pour nommer le client.

- Vous pouvez éventuellement fournir une implémentation de l'interface `MqttClientPersistence` pour remplacer l'implémentation par défaut. L'implémentation `MqttPersistence` par défaut stocke les messages QoS 1 et 2 en attente de distribution sous forme de fichiers ; voir «Persistance des messages dans les clients MQTT», à la page 554.
- Le port TCP/IP IBM WebSphere MQ par défaut pour MQTT est 1883. Pour SSL, il s'agit de 8883. Dans l'exemple, l'adresse par défaut est `tcp://localhost:1883`.
- En règle générale, il est important de pouvoir identifier un client physique spécifique à l'aide de l'identificateur de client. L'identificateur de client doit être unique sur tous les clients qui se connectent à un serveur ; voir «Identificateur de client», à la page 550. L'utilisation du même identificateur de client qu'une instance précédente indique que l'instance présente est une instance du même client. Si vous dupliquez un identificateur de client dans deux clients en cours d'exécution, une exception est émise dans les deux clients et un client s'arrête.
- La longueur de l'identificateur client est limitée à 23 octets. Une exception est émise si la longueur est dépassée. L'identificateur de client ne doit contenir que des caractères autorisés dans un nom de gestionnaire de files d'attente ; par exemple, aucun trait d'union ni espace.
- Tant que vous n'appelez pas la méthode `MqttClient.connect` , aucun traitement de message n'a lieu.

Utilisez l'objet client pour publier et s'abonner à des rubriques et récupérer des informations sur les publications qui n'ont pas encore été distribuées.

5. Juste avant la ligne de code `client.connect();` , instanciez la classe `CallBack` en transmettant l'identificateur de client.

```
CallBack callback = new CallBack(Example.clientId);
client.setCallback(callback);
```

   - La classe `CallBack` implémente `MqttCallBack`. Une instance de rappel est requise, par identificateur de client. Dans cet exemple, le constructeur transmet l'identificateur de client à enregistrer en tant que données d'instance. Il est utilisé dans le rappel pour identifier l'instance du rappel qui a été démarrée.
   - Vous devez implémenter trois méthodes dans la classe de rappel:

     `public void messageArrived(MqttTopic topic, MqttMessage message)`
     Reçoit une publication à laquelle un abonnement a été souscrit.

     `public void connectionLost(Throwable cause)`
     Appelé lorsque la connexion est perdue.

     `public void deliveryComplete(MqttDeliveryToken token))`
     Appelé lorsqu'un jeton de distribution est reçu pour un message QoS 1 ou 2 qui a été publié.
   - Le rappel est activé par `MqttClient.connect`.
6. Créez un objet `MqttConnectOptions` et définissez son attribut `cleanSession` .
   a) Créez un objet `MqttConnectOptions` .

```
MqttConnectOptions conOptions = new MqttConnectOptions();
```

      `conOptions` est un paramètre d'option sur le constructeur `MqttClient` .

   b) Définissez l'attribut clearSession .

```
conOptions.setCleanSession(Example.cleanSession);
```

      Par défaut, le paramètre `Example.cleanSession` est défini sur `true`, ce qui correspond à la valeur par défaut de `MqttConnectionOptions.cleanSession`.

Si vous utilisez l'objet par défaut `MqttConnectOptions`, ou que vous attribuez à `MqttConnectOptions.cleanSession` la valeur `true` avant de connecter le client, les anciens

abonnements pour le 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 attribuez à `MqttConnectOptions.cleanSession` la valeur `false` avant la connexion, les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour ce 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 ; ce mode dure pendant la totalité de la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez des modes `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimées.

7. Transmettez le paramètre `conOptions` au constructeur `MqttClient` .

```
client.connect(conOptions);
```

8. Créez un abonnement.

```
      client.subscribe(Example.topicString, Example.QoS);
```

L'exemple utilise une méthode `MqttClient.subscribe` qui transmet un filtre de rubrique avec une option `QoS` . La méthode `MqttClient.subscribe` comporte quatre signatures et vous pouvez transmettre des tableaux de filtres d'abonnement ainsi qu'un filtre unique.

L'exemple utilise la chaîne de rubrique utilisée par les exemples de publication comme filtre de rubrique, de sorte qu'il reçoit les publications qu'ils créent.

Chaque fois que vous exécutez l'exemple, `subscribe.java`, il crée un abonnement. Si vous ne modifiez pas Example.topicString, le même abonnement est recréé. Si un abonnement est recréé, il ne génère pas deux abonnements identiques. Un client ne reçoit pas de copies en double des publications correspondant à un abonnement identique.

Les abonnements sont décrits dans «Abonnements», à la page 562et les filtres dans «Chaînes de rubrique et filtres de rubrique dans les clients MQTT», à la page 564.

9. Attendez que certaines publications arrivent, puis déconnectez le client.

```
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
```

Les publications sont reçues par l'implémentation de la méthode `MqttCallback.messageArrived` .

L'application d'abonnement n'a pas publié de messages et n'attend donc pas de jetons de distribution. `client.disconnect` a lieu sans délai.

Exemple de code

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.MqttClient;
import com.ibm.micro.client.mqttv3.MqttConnectOptions;
public class Subscribe {
  public static void main(String[] args) {
    Example.clientId = String.format(
        "%-23.23s",
        (System.getProperty("user.name") + "_" + System.getProperty("clientId",
            "Subscribe."))).trim();
    try {
      MqttClient client = new MqttClient(Example.TCPAddress, Example.clientId);
      CallBack callback = new CallBack(Example.clientId);
      client.setCallback(callback);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setCleanSession(Example.cleanSession);
      client.connect(conOptions);
      System.out.println("Subscribing to topic \"" + Example.topicString
          + "\" for client instance \"" + client.getClientId()
          + "\" using QoS " + Example.QoS + ". Clean session is "
          + Example.cleanSession);
      client.subscribe(Example.topicString, Example.QoS);
      System.out.println("Going to sleep for " + Example.sleepTimeout / 1000
          + " seconds");
      Thread.sleep(Example.sleepTimeout);
      client.disconnect();
      System.out.println("Finished");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 100. Subscribe.java

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class CallBack implements MqttCallback {
  private String instanceData = "";
  public CallBack(String instance) {
    instanceData = instance;
  }
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\" for instance \""
          + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  public void connectionLost(Throwable cause) {
    System.out.println("Connection lost on instance \"" + instanceData
        + "\" with cause \"" + cause.getMessage() + "\" Reason code "
        + ((MqttException)cause).getReasonCode() + "\" Cause \""
        + ((MqttException)cause).getCause() +  "\"");
    cause.printStackTrace();
  }
  public void deliveryComplete(MqttDeliveryToken token) {
    try {
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" received by instance \"" + instanceData + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Figure 101. CallBack.java

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figure 102. Example.java

Concepts associés

Applications de publication/abonnement MQTT

Authentification d'un client Java MQTT à l'aide de JAAS

Apprenez à authentifier un client à l'aide de JAAS. Modifiez l'exemple de programme `JAASLoginModule.java` et l'exemple de programme Java `PubSync.java`. Configurez un canal de télémétrie pour exiger l'authentification JAAS et exécutez le diffuseur de publications modifié, en vérifiant son `nom d'utilisateur` et son `mot de passe` à l'aide de JAAS.

Avant de commencer

Vous devez avoir installé les fichiers JAR du client MQTT v3 , Javadoc, Eclipse, configuré les canaux de télémétrie et codé et exécuté PubSync.java avant d'exécuter cette tâche. Vous disposez d'un espace de travail Eclipse qui inclut une version en cours d'exécution de PubSync.java.

La tâche est écrite pour Windows. Modifiez les chemins de répertoire pour Linux.

Pourquoi et quand exécuter cette tâche

La tâche est basée sur la modification de l'exemple de classe JAASLoginModule dans *WMQ Installation directory*\mqxr\samples\JAASLoginModule.java pour créer MyLogin.java. Dans la tâche, vous modifiez également l'exemple de code PubSync.java dans «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 pour définir un nom d'utilisateur et un mot de passe. A titre de test, MyLogin.java accepte ou rejette de manière aléatoire le nom d'utilisateur et le mot de passe.

Les étapes de la tâche sont écrites comme un exercice de programmation. Vous devez adapter la procédure pour effectuer une authentification réelle dans un environnement de production.

Dans une explication typique de la façon de programmer l'authentification JAAS , il est supposé que le module de connexion authentifie le contexte qui a chargé JAAS. Lorsque le service de télémétrie (MQXR) appelle JAAS, le contexte qui a chargé JAAS est le service de télémétrie (MQXR). Il n'y a aucun intérêt à authentifier le contexte de service de télémétrie (MQXR) ; il s'agit toujours de mqm. A la place, le service de télémétrie (MQXR) configure le nom d'utilisateur et le mot de passe du client pour qu'ils soient disponibles pour la classe du module de connexion. Le nom d'utilisateur et le mot de passe sont transmis au module de connexion à l'aide de deux rappels.

```
javax.security.auth.callback.Callback[] callbacks =
      new javax.security.auth.callback.Callback[2];
callbacks[0] =
      new javax.security.auth.callback.NameCallback("NameCallback");
callbacks[1] =
      new javax.security.auth.callback.PasswordCallback("PasswordCallback", false);
callbackHandler.handle(callbacks);
String username =
      ((javax.security.auth.callback.NameCallback) callbacks[0]).getName();
char[] password =
      ((javax.security.auth.callback.PasswordCallback) callbacks[1]).getPassword();
```

Le nom d'utilisateur et le mot de passe du client sont les seules informations sur le client qui sont disponibles pour le module de connexion.

Procédure

1. Créez deux packages, samples et security.jaas , dans le même projet Java que PubSync.java.

   Le package samples est utilisé uniquement à titre de référence. Modifiez le code dans le package security.jaas .
2. Importez JAASLoginModule.java et JAASPrincipal.java dans les deux packages.

   Si nécessaire, restructurez les instructions de package dans la source Java pour éliminer les erreurs de compilation.
3. Restructurez le nom de classe, JAASLoginModule, dans le package security.jaas en MyLogin
4. Dans MyLogin.java, remplacez une partie du code dans la méthode login pour afficher le fonctionnement du module.
   a) Remplacez le code:

   ```
   // Accept everything.
   if (true)
     loggedIn = true;
   else
     throw new javax.security.auth.login.FailedLoginException("Login failed");
   ```

   b) Avec le code:

   ```
   // login half the users randomly
   PrintWriter pw = new PrintWriter(new FileWriter(System.getProperty("user.dir")
           + "\\MyLogin.log", true));
   pw.println("Called JAASLogin.login at "
           + System.getProperty("publication", "Hello World "
           + String.format("%tc", System.currentTimeMillis())));
   if (Math.random() < 0.5)
     loggedIn = true;
   pw.println("Username: \"" + username + "\", Password: \""
   ```

```
    + String.valueOf(password) + "\" loggedIn: " + loggedIn);
pw.close();
if (!loggedIn)
  throw new javax.security.auth.login.FailedLoginException("Login failed");
principal= new JAASPrincipal(username);
```

La source complète de `MyLogin.java` se trouve dans Figure 105, à la page 520. La source de `JAASPrincipal.java`, avec le nom de package refactorisé en `security.jaas` , se trouve dans Figure 106, à la page 521.

5. Définissez le chemin d'accès aux classes dans `service.env` pour qu'il pointe vers le répertoire contenant le chemin d'accès à `security/jaas/MyLogin.class` et à `security/jaas/JAASPrincipal.class`.

   ```
   CLASSPATH=C:\WMQTelemetryApps\MQTTSecureExamples\bin
   ```

   Pour plus d'informations sur l'utilisation de `service.env` pour transmettre un chemin d'accès aux classes à un service WebSphere MQ , voir Configuration du canal de télémétrie JAAS .

6. Ajoutez une section de module de connexion à `jaas.config`.

   ```
   MyLoginExample {
     security.jaas.MyLogin required debug=true;
     };
   ```

   Voir Configuration du canal de télémétrie JAAS pour plus d'informations sur l'utilisation de `jaas.config` pour définir un module de connexion JAAS .

7. Ajoutez un canal de télémétrie à l'aide de l'assistant **Nouveau canal de télémétrie** dans WebSphere MQ Explorer, en configurant le canal pour exiger l'authentification JAAS . Reportez-vous à la section `MyLoginExample` .

   Par exemple, adaptez les informations que vous entrez dans l'assistant à partir de cette section du fichier `mqxr_win.properties` . Si vous travaillez dans Linux , le fichier est appelé `mqxr_unix.properties`. N'éditez pas directement le fichier de propriétés de télémétrie ; utilisez l'assistant.

   ```
   com.ibm.mq.MQXR.channel/JAASMCAUser: \
   com.ibm.mq.MQXR.Port=1884;\
   com.ibm.mq.MQXR.JAASConfig=MyLoginExample;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

   Remarque : Si vous modifiez l'un des paramètres de canal de télémétrie ou la classe `security.jaas.Mylogin` , vous devez arrêter et redémarrer le service de télémétrie (MQXR). Ce n'est que lorsque vous redémarrez le service que les modifications sont prises en compte.

8. Faites une copie de `PubSync.java` dans le package `com.ibm.mq.id` et nommez la copie `PubSyncJAAS.java`.

   Voir «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 pour la procédure de création de PubSync.java dans le package `com.ibm.mq.id` .

9. Définissez `MqttConnectOptions.username` et `MqttConnectOptions.password` dans le programme `PubSyncJAAS.java` et transmettez `MqttConnectOptions` en tant que paramètre de `MqttClient.connect`.

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setUserName(Example.username);
   conOptions.setPassword(Example.password);
   client.connect(conOptions);
   ```

   Voir le code en italique dans PubSyncJAAS.Java à l'aide des constantes définies dans Example.java.

10. Définissez `Example.TCPAddress` sur l'adresse de socket du canal de télémétrie que vous avez configuré pour utiliser la configuration JAAS , `MyLoginExample`. Par exemple, utilisez 1884 comme numéro de port.

11. Exécutez PubSyncJAAS un certain nombre de fois pour voir le client se connecter et être accepté ou rejeté.

    Une exception est émise chaque fois que la tentative de connexion est rejetée.

Résultats

Figure 103, à la page 519 affiche les résultats de l'exécution de PubSyncJAAS.Java deux fois. Les enregistrements de journal sont affichés dans Figure 104, à la page 519.

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:05 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_61c57a18_4bf7_40d" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Client exception caught
Client is not connected (32104)
    at
com.ibm.micro.client.mqttv3.internal.ExceptionHelper.createMqttException(ExceptionHelper.java:33
)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.internalSend(ClientComms.java:88)
    at com.ibm.micro.client.mqttv3.internal.ClientComms.sendNoWait(ClientComms.java:105)
    at com.ibm.micro.client.mqttv3.MqttTopic.publish(MqttTopic.java:68)
    at com.ibm.mq.id.PubSync.main(PubSync.java:24)
```

```
Waiting for up to 10 seconds for publication of "Hello World Fri Jun 04 08:31:40 BST 2010" with
QoS = 1
On topic "MQTT Example" for client instance: "Admin_1d1599a0_50f5_4ea" on address tcp://
localhost:1884"
With username "Admin" and password "Password"
Delivery token "1731749688" has been received: true
```

Figure 103. Sortie de la console à partir de PubSyncJAAS.java

Le fichier journal MyLogin.log est stocké dans *WMQ Data directory*; par exemple, C:\IBM\MQ\Data\MyLogin.log:

```
Called JAASLogin.login at Hello World Fri Jun 04 08:31:05 BST 2010
Username: "Admin", Password: "Password" loggedIn: false
Called JAASLogin.login at Hello World Fri Jun 04 08:31:40 BST 2010
Username: "Admin", Password: "Password" loggedIn: true
```

Figure 104. MyLogin.log

Exemples

Le code en italique dans Figure 105, à la page 520 correspond à la modification de l'exemple JAASLoginModule.java.

```
package security.jaas;
import java.io.FileWriter;
import java.io.PrintWriter;

public class JAASLogin implements javax.security.auth.spi.LoginModule {
  private javax.security.auth.Subject                  subject;
  private javax.security.auth.callback.CallbackHandler callbackHandler;
  JAASPrincipal                                        principal;
  boolean                                              loggedIn = false;
  public void initialize(javax.security.auth.Subject subject,
      javax.security.auth.callback.CallbackHandler callbackHandler,
      java.util.Map<String, ?> sharedState, java.util.Map<String, ?> options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
  }
  public boolean login() throws javax.security.auth.login.LoginException {
    try {
      javax.security.auth.callback.Callback[] callbacks = new
javax.security.auth.callback.Callback[2];
      callbacks[0] = new javax.security.auth.callback.NameCallback(
          "NameCallback");
      callbacks[1] = new javax.security.auth.callback.PasswordCallback(
          "PasswordCallback", false);

      callbackHandler.handle(callbacks);
      String username = ((javax.security.auth.callback.NameCallback) callbacks[0])
          .getName();
      char[] password = ((javax.security.auth.callback.PasswordCallback) callbacks[1])
          .getPassword();
      // login half the users randomly
      PrintWriter pw = new PrintWriter(new FileWriter(System
          .getProperty("user.dir")
          + "\\mylogin.log", true));
      pw.println("Called JAASLogin.login at "
          + System.getProperty("publication", "Hello World "
              + String.format("%tc", System.currentTimeMillis())));
      if (Math.random() < 0.5)
        loggedIn = true;
      pw.println("Username: \"" + username + "\", Password: \""
          + String.valueOf(password) + "\" loggedIn: " + loggedIn);
      pw.close();
      if (!loggedIn)
        throw new javax.security.auth.login.FailedLoginException("Login failed");
      principal = new JAASPrincipal(username);
    } catch (java.io.IOException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    } catch (javax.security.auth.callback.UnsupportedCallbackException exception) {
      throw new javax.security.auth.login.LoginException(exception.toString());
    }
    return loggedIn;
  }
  public boolean abort() throws javax.security.auth.login.LoginException {
    logout();
    return true;
  }
  public boolean commit() throws javax.security.auth.login.LoginException {
    if (loggedIn) {
      if (!subject.getPrincipals().contains(principal))
        subject.getPrincipals().add(principal);
    }
    return true;
  }
  public boolean logout() throws javax.security.auth.login.LoginException {
    subject.getPrincipals().remove(principal);
    principal = null;
    loggedIn = false;
    return true;
  }
}
```

Figure 105. MyLogin.java

Figure 106, à la page 521 est l'exemple de code JAASLoginPrincipal.java, copié dans le package security.jaas. L'objectif de JAASLoginPrincipal est d'implémenter l'interface

`java.security.Principal` pour conserver un enregistrement des utilisateurs qui ont été correctement connectés par `MyLogin`.

```
package security.jaas;
public class JAASPrincipal implements java.security.Principal,
    java.io.Serializable {
  private static final long serialVersionUID = 1L;
  String                    name;
  public JAASPrincipal(String name) {
    this.name = name;
  }
  public String getName() {
    return name;
  }
  public String toString() {
    return (name);
  }
  public boolean equals(Object object) {
    if (object != null && object instanceof JAASPrincipal
        && name.equals(((JAASPrincipal) object).getName()))
      return true;
    else
      return false;
  }
  public int hashCode() {
    return name.hashCode();
  }
}
```

Figure 106. JAASLoginPrincipal.java

Le code dans PubSync.java qui est modifié pour ajouter un `nom d'utilisateur` et un `mot de passe` est mis en italique dans Figure 107, à la page 521.

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setUserName(Example.username);
      conOptions.setPassword(Example.password);
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("With username \"" + conOptions.getUserName()
              + "\" and password \"" + String.valueOf(conOptions.getPassword()) + "\"");
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Figure 107. PubSyncJAAS.Java

Modifiez les constantes dans Example.java pour qu'elles correspondent à votre configuration. Ignorez les paramètres SSL de cet exemple.

```
package com.ibm.mq.id;
import java.util.Properties;
import java.util.UUID;
public final class Example {
  public static final String          TCPAddress =
    System.getProperty("TCPAddress", "tcp://localhost:1883");
  public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");
  public static final String            username =
    System.getProperty("username", System.getProperty("user.name"));
  public static final char []           password =
    System.getProperty("password", "Password").toCharArray();
  public static String                  clientId =
  String.format("%-23.23s", username  + "_" +
    System.getProperty("clientId",
    (UUID.randomUUID().toString())).trim()).replace('-', '_');
  public static final String         topicString =
    System.getProperty("topicString", "MQTT Example");
  public static final String         publication =
    System.getProperty("publication", "Hello World " +
  String.format("%tc", System.currentTimeMillis()));
  public static final int         quiesceTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final int           sleepTimeout =
    Integer.parseInt(System.getProperty("timeout", "10000"));
  public static final boolean       cleanSession =
    Boolean.parseBoolean(System.getProperty("cleanSession", "false"));
  public static final int                   QoS =
    Integer.parseInt(System.getProperty("QoS", "1"));
  public static final boolean           retained =
    Boolean.parseBoolean(System.getProperty("retained", "false"));
  public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore",
        "C:\\IBM\\MQ\\Data\\ClientKeyStore.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword",
        "password");
    properties.setProperty("com.ibm.ssl.trustStore",
        "C:\\IBM\\MQ\\Data\\ClientTrustStore.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType",
        "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword",
        "password");
    return properties;
  }
}
```

Figure 108. Example.java

Authentification d'une connexion de télémétrie SSL à l'aide de certificats autosignés

Utilisez les certificats autosignés générés à l'aide de **Keytool** pour authentifier une connexion SSL. Vous avez la possibilité d'authentifier le canal de télémétrie ou le canal de télémétrie et les clients qui s'y connectent. Les messages circulant sur la connexion sont chiffrés.

Avant de commencer

Effectuez la tâche «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 avant de commencer pour que PubSync.java fonctionne avec une connexion TCP/IP non sécurisée. Dans cette tâche, vous modifiez PubSync.java pour utiliser une connexion SSL.

Pourquoi et quand exécuter cette tâche

Les étapes de la tâche sont écrites comme un exercice de programmation. Vous devez adapter la procédure pour effectuer une authentification réelle dans un environnement de production.

La tâche est écrite pour Windows. Modifiez les chemins de répertoire pour Linux.

Procédure

1. Effectuez la tâche «Modification de PubSync.java pour utiliser SSL», à la page 523pour modifier PubSync.java afin d'utiliser SSL.
2. Configurez le canal de télémétrie et créez les magasins de clés pour utiliser SSL.

   Authentifiez uniquement le canal de télémétrie ou le canal et les clients qui s'y connectent:

   - Effectuez la tâche «Authentification du canal de télémétrie», à la page 524pour vous connecter à SSL et authentifier le canal de télémétrie.
   - Effectuez la tâche «Authentification du canal de télémétrie et des clients», à la page 525pour vous connecter à SSL, en authentifiant le canal de télémétrie et les clients qui s'y connectent.
3. Arrêtez et redémarrez le service de télémétrie (MQXR) pour prendre en compte les modifications apportées aux configurations de canal de télémétrie.
4. Exécutez le programme client pour voir si la configuration fonctionne.

Modification de PubSync.java pour utiliser SSL

Modifiez le premier exemple de programme de publication pour vous connecter à un canal de télémétrie à l'aide de SSL. Définissez les propriétés SSL utilisées par le programme modifié.

Avant de commencer

Vous devez avoir installé les fichiers JAR du client MQTT v3 , Javadoc, Eclipse, configuré les canaux de télémétrie et codé et exécuté PubSync.java avant d'exécuter cette tâche. Vous disposez d'un espace de travail Eclipse qui inclut une version en cours d'exécution de PubSync.java.

Pourquoi et quand exécuter cette tâche

La tâche utilise le client de publication, PubSync.java, que vous avez créé dans «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 comme base. Seules de petites modifications sont nécessaires pour utiliser SSL ; voir Figure 109, à la page 524 et Figure 110, à la page 524.

Procédure

1. Faites une copie de `PubSync.java` dans le package `com.ibm.mq.id` et nommez la copie `PubSyncSSL.java`.

   Voir «Création de votre première application de publieur MQ Telemetry Transport à l'aide de Java», à la page 494 pour la procédure de création de PubSync.java dans le package `com.ibm.mq.id` .
2. Définissez `Example.SSLAddress` sur l'adresse de socket du canal de télémétrie que vous avez configuré pour utiliser pour la configuration SSL.
3. Modifiez le paramètre d'adresse de socket du constructeur client pour utiliser `Example.SSLAddress`.

   ```
   MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
   ```
4. Définissez `MqttConnectOptions.SSLProperties` dans `PubSyncSSL.java`et transmettez `MqttConnectOptions` en tant que paramètre de `MqttClient.connect`.

   ```
   MqttConnectOptions conOptions = new MqttConnectOptions();
   conOptions.setSSLProperties(Example.getSSLSettings());
   client.connect(conOptions);
   ```

   Consultez le code en italique `PubSyncSSL.Java` à l'aide des constantes définies dans `Example.java`.

Exemples

Les modifications apportées à PubSync.java pour ajouter SSL sont affichées en italique dans Figure 109, à la page 524 .

```
package com.ibm.mq.id;
import com.ibm.micro.client.mqttv3.*;
public class PubSyncSSL {
  public static void main(String[] args) {
    try {
      MqttClient client = new MqttClient(Example.SSLAddress, Example.clientId);
      MqttTopic topic = client.getTopic(Example.topicString);
      MqttMessage message = new MqttMessage(Example.publication.getBytes());
      message.setQos(Example.QoS);
      MqttConnectOptions conOptions = new MqttConnectOptions();
      conOptions.setSSLProperties(Example.getSSLSettings());
      client.connect(conOptions);
      System.out.println("Waiting for up to " + Example.sleepTimeout / 1000
              + " seconds for publication of \"" + message.toString()
              + "\" with QoS = " + message.getQos());
      System.out.println("On topic \"" + topic.getName() + "\" for client instance: \""
              + client.getClientId() + "\" on address " + client.getServerURI() + "\"");
      System.out.println("SSL Properties" + conOptions.getSSLProperties());
      MqttDeliveryToken token = topic.publish(message);
      token.waitForCompletion(Example.sleepTimeout);
      System.out.println("Delivery token \"" + token.hashCode()
          + "\" has been received: " + token.isComplete());
      client.disconnect();
    } catch (Exception e) {
      System.out.println("Client exception caught");
      e.printStackTrace();
    }
  }
}
```

Figure 109. PubSyncSSL.Java

Les modifications apportées à Example.java sont présentées dans le Figure 110, à la page 524.

```
 public static final String          SSLAddress =
    System.getProperty("SSLAddress", "ssl://localhost:8883");

 public static final Properties getSSLSettings() {
    final Properties properties = new Properties();
    properties.setProperty("com.ibm.ssl.keyStore", "C:\\Certificates\\SSClientKey.jks");
    properties.setProperty("com.ibm.ssl.keyStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.keyStorePassword", "password");
    properties.setProperty("com.ibm.ssl.trustStore", "C:\\Certificates\\SSClientTrust.jks");
    properties.setProperty("com.ibm.ssl.trustStoreType", "JKS");
    properties.setProperty("com.ibm.ssl.trustStorePassword", "password");
    return properties;
```

Figure 110. Modifications apportées à Example.java

Authentification du canal de télémétrie

Les clients authentifient le canal de télémétrie pour chiffrer le contenu des messages qui circulent sur le canal et pour s'assurer qu'un client se connecte au canal de télémétrie approprié. Le serveur n'authentifie pas le client.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser un certain nombre d'éditeurs de magasin de clés différents pour créer et gérer des certificats autosignés. La tâche utilise la commande **keytool** de la ligne de commande, qui fait partie de l'environnement d'exécution Java (JRE). Vous pouvez utiliser l'outil d'interface graphique **iKeyman**, qui est fourni avec WebSphere MQ pour parcourir les magasins de clés et générer des clés. Lancez **iKeyman** à l'aide de la commande **strmqikm**.

Procédure

1. Créez un canal de télémétrie, SSLSSOptClients , qui nécessite une connexion SSL à l'aide de l'assistant **Nouveau canal de télémétrie** . Le canal accepte les clients anonymes.

   Adaptez votre configuration de canal à partir de la section de configuration suivante. N'éditez pas directement le fichier de propriétés de télémétrie ; utilisez l'assistant.

   ```
   com.ibm.mq.MQXR.channel/SSLSSOptClients: \
   com.ibm.mq.MQXR.Port=8883;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Générez les clés pour que le client authentifie le canal de télémétrie.

   a) Générez une paire de clés autosignée pour le canal de télémétrie dans un nouveau magasin de clés, SSServerOptKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerOptKey.jks -storepass password -keypass password
   ```

   b) Exportez son certificat public en tant que fichier ASCII, à l'aide de l'option -rfc :

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerOptKey.jks -storepass password -rfc
   ```

   Si vous exécutez la tâche sous Windows, cliquez deux fois sur SSServerPublic.cer pour examiner son contenu.

   c) Importez le certificat public dans un nouveau magasin de clés de confiance client, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

   d) Créez un magasin de clés client vide, SSClientKey.jks.

   Keytool n'a pas de commande permettant de créer un magasin de clés vide. Vous avez deux possibilités :

   i) Exécutez **strmqikm**et créez un magasin de clés, SSClientKey.jks, mais n'ajoutez aucune clé.

   ii) Exécutez l'étape 3a dans «Authentification du canal de télémétrie et des clients», à la page 525, mais n'utilisez pas encore les clés.

Authentification du canal de télémétrie et des clients

Les clients authentifient le canal de télémétrie et le canal de télémétrie authentifie les clients qui s'y connectent. Les messages qui circulent sur le canal sont chiffrés.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser un certain nombre d'éditeurs de magasin de clés différents pour créer et gérer des certificats autosignés. La tâche utilise la commande **keytool** de la ligne de commande, qui fait partie de l'environnement d'exécution Java (JRE). Vous pouvez utiliser l'outil d'interface graphique **iKeyman**, qui est fourni avec WebSphere MQ pour parcourir les magasins de clés et générer des clés. Lancez **iKeyman** à l'aide de la commande **strmqikm**.

Le canal de télémétrie est configuré avec un magasin de clés différent de celui de la tâche, «Authentification du canal de télémétrie», à la page 524. Vous pouvez utiliser le même magasin de clés et omettre l'étape «2», à la page 526 pour ajouter des clés au magasin de clés.

Procédure

1. Créez un canal de télémétrie, SSLSSReqClients , qui nécessite une connexion SSL à l'aide de l'assistant **Nouveau canal de télémétrie** . Le canal accepte uniquement les clients authentifiés.

   Adaptez votre configuration de canal à partir de la section de configuration suivante:

   ```
   com.ibm.mq.MQXR.channel/SSLSSReqClients: \
   com.ibm.mq.MQXR.Port=8884;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\SSServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Générez les clés pour que le client authentifie le canal de télémétrie.

   a) Générez une paire de clés autosignée pour le canal de télémétrie dans un nouveau magasin de clés, SSServerReqKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSServerPrivate
           -dname "CN=mqttserver.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSServerReqKey.jks -storepass password -keypass password
   ```

   b) Exportez son certificat public en tant que fichier ASCII, à l'aide de l'option -rfc :

   ```
   Keytool -export -noprompt -alias SSServerPrivate -file SSServerPublic.cer
           -keystore SSServerReqKey.jks -storepass password -rfc
   ```

   Si vous exécutez la tâche sous Windows, cliquez deux fois sur SSServerPublic.cer pour examiner son contenu.

   c) Importez le certificat public dans un nouveau magasin de clés de confiance client, SSClientTrust.jks:

   ```
   Keytool -import -noprompt -alias SSServerPublic -file SSServerPublic.cer
           -keystore SSClientTrust.jks -storepass password
   ```

3. Générez les clés du canal de télémétrie pour authentifier un client.

   a) Générez une paire de clés autosignée pour le client dans un nouveau magasin de clés, SSClientKey.jks:

   ```
   Keytool -genkey -noprompt -alias SSClientPrivate
           -dname "CN=mqttclient.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore SSClientKey.jks -storepass password -keypass password
   ```

   b) Exportez son certificat public en tant que fichier ASCII, à l'aide de l'option -rfc :

   ```
   Keytool -export -noprompt -alias SSClientPrivate -file SSClientPublic.cer
           -keystore SSClientKey.jks -storepass password -rfc
   ```

   Si vous exécutez la tâche sous Windows, cliquez deux fois sur SSClientPublic.cer pour examiner son contenu.

   c) Importez le certificat public dans le magasin de clés du serveur, SSServerReqKey.jks:

   ```
   Keytool -import -noprompt -alias SSClientPublic -file SSClientPublic.cer
           -keystore SSServerReqKey.jks -storepass password
   ```

   Les canaux de télémétrie utilisent le même magasin pour les clés privées et les certificats de confiance.

Authentification d'une connexion de télémétrie SSL à l'aide d'une chaîne de certificats

Utilisez des certificats signés obtenus auprès d'une autorité de certification ou en implémentant votre propre procédure de certification pour authentifier une connexion SSL. Vous avez la possibilité d'authentifier le canal de télémétrie ou le canal de télémétrie et les clients qui s'y connectent. Les messages circulant sur la connexion sont chiffrés.

Avant de commencer

Effectuez la tâche «Authentification d'une connexion de télémétrie SSL à l'aide de certificats autosignés», à la page 522 avant de commencer pour que `PubSyncSSL.Java` utilise une connexion TCP/IP sécurisée à l'aide de certificats autosignés.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, modifiez les tâches «Authentification du canal de télémétrie», à la page 524et «Authentification du canal de télémétrie et des clients», à la page 525 dans «Authentification d'une connexion de télémétrie SSL à l'aide de certificats autosignés», à la page 522pour utiliser des clés certifiées par une chaîne de certificats.

Vous pouvez obtenir les certificats pour cette tâche auprès d'une autorité de certification ou utiliser des sites Web tels que http://www.openca.org/ pour obtenir des certificats. Les autorités de certification commerciales fournissent généralement des certificats d'essai pour une courte période et sans frais. Cette tâche a été testée à l'aide de certificats obtenus commercialement.

Une autre option consiste à générer votre propre processus de certification et à l'exécuter sur vos propres ordinateurs, à l'aide d'outils provenant de sites Web tels que https://www.openssl.org/.

Les magasins de clés de confiance `cacerts` de l'environnement d'exécution Java ne sont pas utilisés dans cette tâche. Vous pouvez utiliser le magasin de clés de confiance JRE `cacerts` sur le client dans la tâche, «Authentification du canal de télémétrie», à la page 527, au lieu d'utiliser le magasin de clés de confiance spécifié. La chaîne de certificats peut être signée par une autorité de certification bien connue qui possède déjà son certificat racine dans le magasin `cacerts` sur le client. Dans ce cas, n'indiquez pas de magasin de clés de confiance sur le client. Assurez-vous que, si plusieurs environnements d'exécution Java sont installés sur le client, vous gérez le magasin `cacerts` approprié.

Procédure

1. Si vous ne l'avez pas déjà fait, effectuez la tâche «Modification de PubSync.java pour utiliser SSL», à la page 523pour modifier PubSync.java afin d'utiliser SSL.
2. Configurez le canal de télémétrie et créez les magasins de clés pour utiliser SSL.

   Authentifiez uniquement le canal de télémétrie ou le canal et les clients qui s'y connectent:

   - Effectuez la tâche «Authentification du canal de télémétrie», à la page 527pour vous connecter à SSL et authentifier le canal de télémétrie.
   - Effectuez la tâche «Authentification du canal de télémétrie et des clients», à la page 529pour vous connecter à SSL, en authentifiant le canal de télémétrie et les clients qui s'y connectent.
3. Arrêtez et redémarrez le service de télémétrie (MQXR) pour prendre en compte les modifications apportées aux configurations de canal de télémétrie.
4. Exécutez le programme client pour voir si la configuration fonctionne.

Authentification du canal de télémétrie

Les clients authentifient le canal de télémétrie pour chiffrer le contenu des messages qui circulent sur le canal et pour s'assurer qu'un client se connecte au canal de télémétrie approprié. Le serveur n'authentifie pas le client.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser un certain nombre d'éditeurs de magasin de clés différents pour créer et gérer des certificats. La tâche utilise la commande **keytool** de la ligne de commande, qui fait partie de l'environnement d'exécution Java (JRE). Vous pouvez utiliser l'outil d'interface graphique **iKeyman**, qui est fourni avec WebSphere MQ pour parcourir les magasins de clés et générer des clés. Lancez **iKeyman** à l'aide de la commande **strmqikm**.

Procédure

1. Créez un canal de télémétrie, SSLCAOptClients , qui nécessite une connexion SSL à l'aide de l'assistant **Nouveau canal de télémétrie** . Le canal accepte les clients anonymes.

   Adaptez votre configuration de canal à partir de la section de configuration suivante. N'éditez pas directement le fichier de propriétés de télémétrie ; utilisez l'assistant.

   ```
   com.ibm.mq.MQXR.channel/SSLCAOptClients: \
   com.ibm.mq.MQXR.Port=8885;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerOptKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=OPTIONAL;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Générez une clé signée par l'autorité de certification pour que le client authentifie le canal de télémétrie.

   a) Générez une paire de clés autosignée pour le canal de télémétrie dans un nouveau magasin de clés, SSServerOptKey.jks:

   ```
   Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
           -dname "CN=mqttserverOpt.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   L'algorithme de clé est défini sur RSA car certaines autorités de certification l'exigent. Le nom usuel du certificat doit être unique, certaines autorités de certification ne délivrent pas de clés avec des noms communs identiques.

   b) Créer une demande de signature de certificat (CSR) en tant que fichier ASCII

   ```
   Keytool -certreq -noprompt -alias CAServer -file CAServerOptKey.csr
           -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
           -keystore CAServerOptKey.jks -storepass password -keypass password
   ```

   c) Exécutez le logiciel de l'autorité de certification ou connectez-vous à leur site Web. Collez le contenu de CAServerOptKey.csr lorsque vous êtes invité à indiquer le fichier CSR.

   d) L'autorité de certification renvoie un ou deux certificats et un fichier de réponses signé sous forme de fichiers ASCII. Collez le contenu dans deux ou trois fichiers:

   Certificat racine
   Coller dans CARoot.cer

   certificat intermédiaire
   Coller dans CAInter.cer

   Fichier de réponses signé par le serveur
   Coller dans CAServerOpt.rsp

   Le magasin de certificats JRE n'est pas utilisé dans cette tâche. Si vous avez reçu un certificat racine et une réponse signée de l'autorité de certification, utilisez le certificat racine et la réponse signée dans les étapes suivantes. Si vous avez reçu un certificat racine et un certificat intermédiaire, utilisez le certificat intermédiaire et la réponse signée.

   e) Recevez la réponse du serveur signée dans le magasin de clés du serveur à partir duquel vous avez émis la demande de certificat.

La réception de la réponse modifie le certificat autosigné de sorte qu'il soit signé par l'autorité de certification. Si vous examinez le certificat dans le magasin de clés avant et après la réception de la réponse, le signataire change. Si ce n'est pas le cas, une erreur est signalée par l'outil de gestion des clés. Avant d'utiliser le certificat, examinez-le et vérifiez que le signataire est désormais l'autorité de certification.

```
Keytool -import -noprompt -alias CAServer -file CAServerOpt.rsp
        -keystore CAServerOptKey.jks -storepass password
```

Dans certains logiciels de gestion de clés, tels que **iKeyman**, vous recevez plutôt que d'importer des fichiers de réponses.

f) Importez le certificat de l'autorité de certification dans le magasin de clés de confiance du client.

Importez le certificat intermédiaire si vous avez reçu deux certificats de l'autorité de certification ou le certificat racine si vous n'avez reçu qu'un seul certificat.

L'un ou l'autre :

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

Ou:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

Authentification du canal de télémétrie et des clients

Les clients authentifient le canal de télémétrie et le canal de télémétrie authentifie les clients qui s'y connectent. Les messages qui circulent sur le canal sont chiffrés.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser un certain nombre d'éditeurs de magasin de clés différents pour créer et gérer des certificats. La tâche utilise la commande **keytool** de la ligne de commande, qui fait partie de l'environnement d'exécution Java (JRE). Vous pouvez utiliser l'outil d'interface graphique **iKeyman**, qui est fourni avec WebSphere MQ pour parcourir les magasins de clés et générer des clés. Lancez **iKeyman** à l'aide de la commande **strmqikm**.

Le canal de télémétrie est configuré avec un magasin de clés différent de celui de la tâche «Authentification du canal de télémétrie», à la page 527. Vous pouvez utiliser le même magasin de clés et omettre l'étape «2», à la page 529 pour ajouter des clés au magasin de clés.

Procédure

1. Créez un canal de télémétrie, SSLCAReqClients , qui nécessite une connexion SSL à l'aide de l'assistant **Nouveau canal de télémétrie** . Le canal accepte uniquement les clients authentifiés.

   Adaptez votre configuration de canal à partir de la section de configuration suivante. N'éditez pas directement le fichier de propriétés de télémétrie ; utilisez l'assistant.

   ```
   com.ibm.mq.MQXR.channel/SSLCAReqClients: \
   com.ibm.mq.MQXR.Port=8886;\
   com.ibm.mq.MQXR.Backlog=4096;\
   com.ibm.mq.MQXR.KeyFileName=C:\\Certificates\\CAServerReqKey.jks;\
   com.ibm.mq.MQXR.PassPhrase=password;\
   com.ibm.mq.MQXR.ClientAuth=REQUIRED;\
   com.ibm.mq.MQXR.UserName=Admin;\
   com.ibm.mq.MQXR.StartWithMQXRService=true
   ```

2. Générez une clé signée par l'autorité de certification pour que le client authentifie le canal de télémétrie.

a) Générez une paire de clés autosignée pour le canal de télémétrie dans un nouveau magasin de clés, `CAServerReqKey.jks`:

```
Keytool -genkey -noprompt -alias CAServerPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAServerReqKey.jks -storepass password -keypass password
```

L'algorithme de clé est défini sur RSA car certaines autorités de certification l'exigent. Le nom usuel du certificat doit être unique, certaines autorités de certification ne délivrent pas de clés avec des noms communs identiques.

b) Créer une demande de signature de certificat (CSR) en tant que fichier ASCII

```
Keytool -certreq -noprompt -alias CAServer -file CAServerReqKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAServerReqKey.jks -storepass password -keypass password
```

c) Exécutez le logiciel de l'autorité de certification ou connectez-vous à leur site Web. Collez le contenu de `CAServerReqKey.csr` lorsque vous êtes invité à indiquer le fichier CSR.

d) L'autorité de certification renvoie un ou deux certificats et un fichier de réponses signé sous forme de fichiers ASCII. Collez le contenu dans deux ou trois fichiers:

Certificat racine
Coller dans `CARoot.cer`

certificat intermédiaire
Coller dans `CAInter.cer`

Fichier de réponses signé par le serveur
Coller dans `CAServerReq.rsp`

Le magasin de certificats JRE n'est pas utilisé dans cette tâche. Si vous avez reçu un certificat racine et une réponse signée de l'autorité de certification, utilisez le certificat racine et la réponse signée dans les étapes suivantes. Si vous avez reçu un certificat racine et un certificat intermédiaire, utilisez le certificat intermédiaire et la réponse signée.

e) Recevez la réponse du serveur signée dans le magasin de clés du serveur à partir duquel vous avez émis la demande de certificat.

La réception de la réponse modifie le certificat autosigné de sorte qu'il soit signé par l'autorité de certification. Si vous examinez le certificat dans le magasin de clés avant et après la réception de la réponse, le signataire change. Si ce n'est pas le cas, une erreur est signalée par l'outil de gestion des clés. Avant d'utiliser le certificat, examinez-le et vérifiez que le signataire est désormais l'autorité de certification.

```
Keytool -import -noprompt -alias CAServer -file CAServerReq.rsp
        -keystore CAServerReqKey.jks -storepass password
```

Dans certains logiciels de gestion de clés, tels que **iKeyman**, vous recevez plutôt que d'importer des fichiers de réponses.

f) Importez le certificat de l'autorité de certification dans le magasin de clés de confiance du client.

Importez le certificat intermédiaire si vous avez reçu deux certificats de l'autorité de certification ou le certificat racine si vous n'avez reçu qu'un seul certificat.

L'un ou l'autre :

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAClientTrust.jks -storepass password
```

Ou:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAClientTrust.jks -storepass password
```

3. Générez une clé signée par l'autorité de certification pour le canal de télémétrie afin d'authentifier les clients.

a) Générez une paire de clés autosignée pour les clients dans un nouveau magasin de clés, `CAClientKey.jks`:

```
Keytool -genkey -noprompt -alias CAClientPrivate -keyalg RSA
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

L'algorithme de clé est défini sur RSA car certaines autorités de certification l'exigent. Le nom usuel du certificat doit être unique, certaines autorités de certification ne délivrent pas de clés avec des noms communs identiques.

b) Créer une demande de signature de certificat (CSR) en tant que fichier ASCII

```
Keytool -certreq -noprompt -alias CAClient -file CAClientKey.csr
        -dname "CN=mqttserverReq.ibm.com, OU=ID, O=IBM, L=Hursley, S=Hants, C=GB"
        -keystore CAClientKey.jks -storepass password -keypass password
```

c) Exécutez le logiciel de l'autorité de certification ou connectez-vous à leur site Web. Collez le contenu de `CAClientKey.csr` lorsque vous êtes invité à indiquer le fichier CSR.

d) L'autorité de certification renvoie un ou deux certificats et un fichier de réponses signé sous forme de fichiers ASCII. Collez le contenu dans deux ou trois fichiers:

Certificat racine
Coller dans `CARoot.cer`

certificat intermédiaire
Coller dans `CAInter.cer`

Fichier de réponses signé par le client
Coller dans `CAClient.rsp`

Le magasin de certificats JRE n'est pas utilisé dans cette tâche. Si vous avez reçu un certificat racine et une réponse signée de l'autorité de certification, utilisez le certificat racine et la réponse signée dans les étapes suivantes. Si vous avez reçu un certificat racine et un certificat intermédiaire, utilisez le certificat intermédiaire et la réponse signée.

e) Recevez la réponse client signée dans le magasin de clés client à partir duquel vous avez émis la demande de certificat.

La réception de la réponse modifie le certificat autosigné de sorte qu'il soit signé par l'autorité de certification. Si vous examinez le certificat dans le magasin de clés avant et après la réception de la réponse, le signataire change. Si ce n'est pas le cas, une erreur est signalée par l'outil de gestion des clés. Avant d'utiliser le certificat, examinez-le et vérifiez que le signataire est désormais l'autorité de certification.

```
Keytool -import -noprompt -alias CAClient -file CAClient.rsp
        -keystore CAClientKey.jks -storepass password
```

Dans certains logiciels de gestion de clés, tels que **iKeyman**, vous recevez plutôt que d'importer des fichiers de réponses.

f) Importez le certificat de l'autorité de certification dans le magasin de clés du serveur.

Importez le certificat intermédiaire si vous avez reçu deux certificats de l'autorité de certification ou le certificat racine si vous n'avez reçu qu'un seul certificat.

L'un ou l'autre :

```
keytool -import -alias CAInter -file CAInter.cer
        -keystore CAServerReqKey.jks -storepass password
```

Ou:

```
keytool -import -alias CARoot -file CARoot.cer
        -keystore CAServerReqKey.jks -storepass password
```

Création de votre première application de publieur MQ Telemetry Transport à l'aide de C

Les étapes de création d'une application de diffuseur de publications client MQTT sont décrites dans un tutoriel. Chaque ligne de code C est expliquée. A la fin de la tâche, vous aurez créé un diffuseur de publications MQTT.

Avant de commencer

L'application client développée utilise les bibliothèques client client MQTT v3 C. L'application se connecte au démon pour dispositifs WebSphere MQ Telemetry pour publier des messages. Voir Création de votre premier diffuseur de publications pour un exemple de client communiquant avec WebSphere MQ Telemetry.

Pourquoi et quand exécuter cette tâche

L'exemple est une application de publication, `pubsync.c`. Le programme `pubsync.c` publie un message avec le contenu `Hello World!` dans la rubrique `MQTT Exampleet` attend la confirmation que la publication a été distribuée au démon.

Par souci de simplicité, les codes retour de certaines fonctions utilisées ne sont pas testés pour une exécution correcte. Dans le code de production, les codes retour peuvent être vérifiés pour s'assurer que le programme se comporte comme prévu. Une action appropriée doit être effectuée si une erreur imprévue se produit.

En configurant un abonné à `MQTT Example` , vous pouvez vérifier que l'application fonctionne.

Utilisez votre environnement de développement C sélectionné pour développer, générer et exécuter le client. Si vous préférez, vous pouvez copier le code directement à partir des exemples.

Procédure

1. Créez un nouveau fichier source vide, `pubsync.c`
2. Créez un fichier, `settings.h`. Copiez le code de la figure 2 dans le fichier.

   Tous les paramètres utilisés dans le programme sont définis dans `settings.h`. Vous pouvez remplacer les valeurs en modifiant les valeurs dans le fichier.
3. Les étapes qui suivent expliquent le code. Suivez les étapes ou copiez le code de la Figure 1 dans `pubsync.c`.
4. Ajoutez les instructions d'inclusion de fichier d'en-tête pour les bibliothèques standard requises et les fichiers `MQTTClient.h` et `settings.h` .

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "string.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. Démarrez la définition de la fonction `main()` .

   ```
   int main(int argc, char* argv[])
   {
   ```
6. Définissez les variables locales utilisées dans le programme.

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_message pubmsg;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   Remarque : Les options de connexion sont requises par la fonction `MQTTClient_connect` . `MQTTClient_connectOptions_initializer` contient les options par défaut.

7. Créez un client.

```
MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
```

   - `& client` est un pointeur vers un descripteur pour le client nouvellement créé. Lorsque cette fonction est renvoyée avec un code retour 0, elle contient un descripteur pour le nouveau client. L'exemple suppose que l'opération a abouti. Testez le code d'erreur pour une exécution correcte dans le code de production.
   - ADDRESS est l'URI du port MQTT que le démon surveille pour les demandes de connexion client entrantes.
   - `CLIENTID` est le nom utilisé pour identifier le client auprès du démon. Chaque client actif doit avoir un nom unique. Si vous dupliquez un identificateur de client dans deux clients en cours d'exécution, une exception est émise dans les deux clients et un client s'arrête. Le nom est utilisé par le démon pour reconnaître qu'un client tjhat se reconnecte après une déconnexion. Voir Identificateur client.
   - `MQTTCLIENT_PERSISTENCE_NONE` indique que l'état du client est conservé en mémoire et qu'il est perdu en cas de défaillance du système. `MQTTCLIENT_PERSISTENCE_DEFAULT` spécifie la persistance basée sur le système de fichiers, offrant une protection contre les échecs. Pour les applications plus spécialisées, vous pouvez utiliser `MQTTCLIENT_PERSISTENCE_USER`, qui fournit une interface permettant d'implémenter votre propre mécanisme de persistance. Pour plus de détails, voir la documentation d'API pour `MQTTClientPersistence.h`. La question de savoir si la persistance est requise est une question de conception d'application. Pour plus de détails, voir Persistance des messages.
   - Le port TCP/IP du démon par défaut pour MQTT est 1883. Dans l'exemple, l'adresse par défaut est `tcp://localhost:1883`.
   - Tant que vous n'appelez pas la fonction `MQTTClient_connect` , aucun traitement de message n'a lieu.
8. Connectez le client au démon.

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
     printf("Failed to connect, return code %d\n", rc);
     exit(-1);
}
```

   - La fonction `MQTTClient_connect` est appelée, transmettant le descripteur client et un pointeur vers les options de connexion en tant qu'arguments.
   - Le code retour de l'appel `MQTTClient_connect` est testé pour s'assurer que la demande de connexion a abouti.
   - Si le `MQTTClient_connect` échoue, le programme se termine avec le code d'erreur -1.
   - Une fois l'application connectée, vous pouvez commencer à publier et à vous abonner.
   - Un petit message "keep-alive" est envoyé toutes les 20 secondes pour empêcher la fermeture de la connexion TCP/IP. Cette option est définie par `conn_opts.keepAliveInterval`.
   - La session est démarrée sans vérification de l'achèvement des messages en cours restants d'une connexion précédente car `conn_opts.cleansession` est défini sur true. Pour plus de détails, voir Nettoyer les sessions.
   - Aucun message de dernière volonté et testament n'est créé pour la connexion. Pour plus de détails, voir Last will and testament.
9. Remplissez la structure *MQTTClient_message* avec les données permettant de définir la charge de message et ses attributs.

```
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
```

   - PAYLOAD est notre contenu de message.

- L'exemple utilise un contenu de chaîne, mais les contenus MQTT sont des tableaux d'octets. La longueur de la chaîne est requise pour spécifier la taille du contenu.
- L'exemple publie un message QoS=1 . Par conséquent, définissez la valeur en conséquence.
- L'attribut conservé est défini sur false (0) car le message ne doit pas être conservé par le démon. Pour plus de détails, voir Publications conservées.

10. Publiez le message.

```
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
```

- La fonction de publication spécifie le client, la rubrique et le contenu à envoyer au démon.
- TOPIC est défini dans `settings.h` en tant que `MQTT Example`.
- La fonction est également transmise à un *MQTTClient_deliveryToken*. Ce pointeur est rempli avec un jeton représentant le message lorsque la fonction est renvoyée.
- Le message est désormais transféré en toute sécurité au client MQTT, mais pas encore au démon. Si le message contient QoS=1 ou 2, le message est stocké localement, au cas où le client échouerait avant la fin de la distribution.
- Cette fonction renvoie un code d'erreur que vous pouvez tester pour une exécution correcte dans le code de production.

11. Attendez l'accusé de réception du serveur.

```
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
```

- L'exemple `pubsync.c` attend un accusé de réception du serveur, qui confirme que le message a été distribué.
- Les arguments du client et du jeton identifient le message spécifique que le programme attend d'être exécuté.
- TIMEOUT limite la durée pendant laquelle le programme attend la fin de la distribution du message. La tâche Création d'un diffuseur de publications asynchrone pour MQ Telemetry Transport à l'aide de C montre comment recevoir des accusés de réception sans attendre à l'aide des fonctions de rappel.
- Cette fonction renvoie un code d'erreur qui peut être testé pour une exécution correcte dans le code de production.

12. Déconnectez le client du démon.

```
MQTTClient_disconnect(client, 10000);
```

- Le client se déconnecte du serveur et attend que les fonctions de rappel (non utilisées dans cet exemple) soient terminées pour les messages en cours.
- Le second argument indique un délai de mise au repos en millisecondes. L'exemple attend jusqu'à 10 secondes pour terminer tout autre travail qu'il doit effectuer avant de se déconnecter.
- Cette fonction renvoie un code d'erreur qui doit être testé pour une exécution correcte dans le code de production.

13. Libérez de la mémoire utilisée par le client et arrêtez le programme.

```
MQTTClient_destroy(&client);
}
```

Résultats

Pour afficher les publications envoyées par ce client, créez un abonné à la rubrique `MQTT Example` . Pour plus de détails, voir Création d'un abonné pour MQ Telemetry Transport à l'aide de C

Exemple

La Figure 1 est une liste complète du code décrit dans Procédure. Le fichier `settings.h` de la figure 2 permet de modifier les paramètres par défaut utilisés dans `pubsync.c`.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"

int pubsync_main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID,  MQTTCLIENT_PERSISTENCE_NONE, NULL);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
          printf("Failed to connect, return code %d\n", rc);
         exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for up to %d seconds for publication of %s\n"
          "on topic %s for client with ClientID: %s\n",
          TIMEOUT/1000, PAYLOAD, TOPIC, CLIENTID);
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    printf("Message with delivery token %d delivered\n", token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figure 111. `pubsync.c`

```
#define ADDRESS   "tcp://localhost:1883"
#define CLIENTID  "ExampleClientPub"
#define TOPIC     "MQTT Example"
#define PAYLOAD   "Hello World!"
#define QOS      1
#define TIMEOUT   10000L
```

Figure 112. `settings.h`

Création d'un diffuseur asynchrone pour MQ Telemetry Transport à l'aide de C

Les étapes de création d'une application de diffuseur de publications asynchrone de client MQTT sont décrites dans un tutoriel. Chaque ligne de code C est expliquée. A la fin de la tâche, vous aurez créé un diffuseur de publications asynchrone MQTT.

Dans cette tâche, vous suivez un tutoriel pour modifier votre première application de publication. Les modifications permettent à l'application d'envoyer des publications sans attendre les accusés de réception de distribution. Les accusés de réception de distribution sont reçus par une fonction de rappel que vous créez.

Avant de commencer

L'application client développée utilise les bibliothèques client client MQTT v3 C. L'application se connecte au démon pour dispositifs WebSphere MQ Telemetry pour publier des messages. Voir Création de votre premier diffuseur de publications pour un exemple de client communiquant avec WebSphere MQ Telemetry.

Pourquoi et quand exécuter cette tâche

L'exemple est une application de publication, `pubasync.c`. Le programme `pubasync.c` publie un message avec le contenu `Hello World!` dans la rubrique `MQTT Example`, sans attendre la confirmation que la publication a été distribuée au démon. Les accusés de réception de distribution sont reçus dans une fonction de rappel, `MQTTClient_deliveryComplete`.

Par souci de simplicité, les codes retour de certaines fonctions utilisées ne sont pas testés pour une exécution correcte. Dans le code de production, les codes retour peuvent être vérifiés pour s'assurer que le programme se comporte comme prévu. Une action appropriée doit être effectuée si une erreur imprévue se produit.

En configurant un abonné à `MQTT Example` , vous pouvez vérifier que l'application fonctionne.

Utilisez votre environnement de développement C sélectionné pour développer, générer et exécuter le client.

Les étapes de la procédure modifient l'application `pubsync.c` à partir de «Création de votre première application de publieur MQ Telemetry Transport à l'aide de C», à la page 532. Si vous préférez, vous pouvez copier le code directement à partir des exemples.

Procédure

1. Créez un nouveau fichier source vide, `callback.h`.
2. Copiez le code de la figure 2 dans le fichier.
   - `callback.h` déclare les trois méthodes de rappel requises pour l'opération client asynchrone.
   - Une variable, *deliveredtoken*, est également déclarée. Ceci est accessible par le programme principal et le rappel sur les différentes unités d'exécution. Il est donc déclaré volatil. Lors de l'utilisation de rappels, veillez à ce que les variables pertinentes soient accessibles de manière sécurisée par des unités d'exécution.
3. Créez un nouveau fichier source vide, `callback.c`.
4. Copiez le code de la figure 3 dans le fichier.
   - `callback.c` implémente les trois méthodes de rappel utilisées par le client pour les opérations asynchrones, `delivered`, `msgarrvdet` `connlost`.
5. Ajoutez une instruction include pour `callback.h` après les autres inclusions dans `pubasync.c`.

   ```
   #include "callback.h"
   ```

6. Copiez le contenu de `pubsync.c` dans un nouveau fichier, `pubasync.c`.
7. Juste avant l'appel de la fonction `MQTTClient_connect` dans `pubasync.c`, définissez les méthodes de rappel pour le client.

   ```
   MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
   ```

   - Vous devez spécifier trois fonctions de rappel. Ces fonctions sont implémentées dans `callback.c`.
   - `MQTTClient_messageArrived` est appelé lorsqu'un message est envoyé au client en raison d'un abonnement correspondant. Cette valeur doit être vraie lorsque le message reçu a été reçu avec succès par l'application client. Le renvoi de la valeur false indique au client que votre application a rencontré un problème lors de la réception du message.
   - `MQTTClient_connectionLost` est appelé lorsque le client perd sa connexion au serveur.
   - `MQTTClient_deliveryComplete` est appelé lorsqu'un message QoS1 ou QoS2 est arrivé et a été accusé de réception par le serveur. Il n'est pas appelé pour les messages QoS0 . Dans l'exemple, cette fonction sauvegarde le jeton à partir du message distribué dans *deliveredtoken* pour indiquer qu'un message est arrivé.
   - `MQTTClient_setCallbacks` doit être appelé lorsque le client est déconnecté du serveur.

- Le second argument vous permet de transmettre des informations contextuelles aux fonctions de rappel. Cette valeur n'est pas utilisée dans l'exemple. Par conséquent, elle est définie sur NULL.

8. Immédiatement avant l'appel à `MQTTClient_publishMessage`, désélectionnez *deliveredtoken*. `MQTTClient_deliveryComplete` pour définir *deliveredtoken* lorsqu'un jeton est reçu.

```
deliveredtoken = 0;
```

9. Supprimez l'appel `MQTTClient_waitForCompletion` et l'instruction `printf` qui le suit et remplacez-le par une boucle en attente d'une correspondance entre le jeton d'origine et le jeton reçu dans le rappel.

```
while(deliveredtoken != token);
```

   Il s'agit d'un exemple qui ne fait pas face à un certain nombre de situations qui doivent être prises en compte dans la conception du code de production. Il peut s'agir des situations suivantes :

   - Si la distribution n'est pas terminée, un délai d'attente peut être implémenté
   - Plusieurs messages peuvent être en cours. L'exemple de programme ne permet de vérifier qu'un seul jeton de distribution à la fois.

10. Déconnectez le client du démon.

```
MQTTClient_disconnect(client, 10000);
```

    - Le client se déconnecte du serveur et attend que les fonctions de rappel des messages en cours soient terminées.
    - Le second argument indique un délai de mise au repos en millisecondes. L'exemple attend jusqu'à 10 secondes pour terminer tout autre travail qu'il doit effectuer avant de se déconnecter.
    - Cette fonction renvoie un code d'erreur qui doit être testé pour une exécution correcte dans le code de production.

11. Libérez de la mémoire utilisée par le client et arrêtez le programme.

```
MQTTClient_destroy(&client);
}
```

Résultats

Pour afficher la publication envoyée par ce client, créez un abonné à la rubrique `MQTT Example` . Pour plus de détails, voir Création d'un abonné pour MQ Telemetry Transport

Exemple

`pubasync.c`, `callbacks.c` et `callbacks.h` sont des listes complètes du code décrit dans Procédure.

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg;
    MQTTClient_deliveryToken token;
    int rc;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
    }
    pubmsg.payload = PAYLOAD;
    pubmsg.payloadlen = strlen(PAYLOAD);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    deliveredtoken = 0;
    MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
    printf("Waiting for publication of %s\n"
           "on topic %s for client with ClientID: %s\n", PAYLOAD, TOPIC, CLIENTID);
    while(deliveredtoken != token);
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figure 113. pubasync.c

```
MQTTClient_deliveryComplete delivered;
MQTTClient_messageArrived msgarrvd;
MQTTClient_connectionLost connlost;

extern volatile MQTTClient_deliveryToken deliveredtoken;
```

Figure 114. callback.h

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt)
{
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message)
{
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
    }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    free(topicName);
    return 1;
}

void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Figure 115. callback.c

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientPub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Figure 116. settings.h

Création d'un abonné pour MQ Telemetry Transport à l'aide de C

Les étapes de création d'une application d'abonné client MQTT sont décrites dans un tutoriel. Chaque ligne de code C est expliquée. A la fin de la tâche, vous aurez créé un abonné MQTT.

Avant de commencer

L'application client développée utilise les bibliothèques client client MQTT v3 C. L'application se connecte au démon pour dispositifs WebSphere MQ Telemetry pour publier des messages. Voir Création de votre premier diffuseur de publications pour un exemple de client communiquant avec WebSphere MQ Telemetry.

Pourquoi et quand exécuter cette tâche

L'exemple est une application d'abonné, `subscribe.c`. Le programme `subscribe.c` s'abonne à la rubrique `MQTT Example` et attend les publications qui correspondent à l'abonnement jusqu'à ce que l'utilisateur mette fin au programme.

Un abonné crée un abonnement à une rubrique et attend les messages qui correspondent à la rubrique d'abonnement. Les messages publiés alors que le client est déconnecté et qui correspondent à un abonnement créé précédemment par le client peuvent être reçus lorsque le client se reconnecte. Le service ou démon pour dispositifs WebSphere MQ Telemetry (MQXR) reconnaît un client qui a déjà été connecté par l'identificateur client. Pour plus d'informations, voir Identificateur de

client. L'attribut booléen `MQTTClient_connectOptions.cleansession` contrôle si les publications envoyées précédemment sont reçues ou non. Pour plus de détails, voir «Sessions propres», à la page 548.

Par souci de simplicité, les codes retour de certaines fonctions utilisées ne sont pas testés pour une exécution correcte. Dans le code de production, les codes retour peuvent être vérifiés pour s'assurer que le programme se comporte comme prévu. Une action appropriée peut être effectuée si une erreur imprévue se produit.

Vous pouvez utiliser les exemples de programmes de publication décrits précédemment pour envoyer des publications correspondantes au démon pour dispositifs WebSphere MQ Telemetry . Vous pouvez également utiliser l'explorateur WebSphere MQ pour créer des publications de test sur la rubrique `MQTT Example` si vous souhaitez connecter le client à un canal WebSphere MQ Telemetry .

Les instructions de la rubrique Procédure supposent que vous avez déjà créé des fichiers `callback.c`, `callback.h`et `settings.h` dans l'une des tâches précédentes.

Utilisez votre environnement de développement C sélectionné pour développer, générer et exécuter le client. Si vous préférez, vous pouvez copier le code directement à partir des exemples.

Procédure

1. Créez une copie de `settings.h` pour cet exemple et modifiez l'instruction de définition *CLIENTID* comme suit:

   ```
   #define CLIENTID "ExampleClientSub"
   ```

   - Si deux clients avec le même ID tentent de se connecter à un seul serveur, l'un d'eux est déconnecté de force. En règle générale, la nouvelle tentative de connexion aboutit et l'ancienne connexion est déconnectée.
   - La modification de l' `ClientID` vous permet d'utiliser les exemples de publication développés précédemment pour envoyer des messages à cet abonné.
2. Créez un nouveau fichier source vide, `subscribe.c`.
3. Les étapes qui suivent expliquent le code. Suivez les étapes ou copiez le code de Figure 117, à la page 543 dans le fichier `subscribe.c`.
4. Ajoutez les instructions d'inclusion de fichier d'en-tête pour les bibliothèques standard requises et les fichiers `MQTTClient.h` et `settings.h` .

   ```
   #include "stdio.h"
   #include "stdlib.h"
   #include "MQTTClient.h"
   #include "settings.h"
   ```
5. Démarrez la définition de la fonction `main()` .

   ```
   int main(int argc, char* argv[]) {
   ```
6. Définissez les variables locales utilisées dans le programme.

   ```
   MQTTClient client;
   MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
   MQTTClient_deliveryToken token;
   int rc;
   ```

   Les options de connexion sont requises par la fonction `MQTTClient_connect` . `MQTTClient_connectOptions_initializer` contient les options par défaut.
7. Créez un client.

   ```
   MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
   ```

   - `& client` est un pointeur vers un descripteur pour le client nouvellement créé. Lorsque cette fonction est renvoyée avec un code retour 0, le pointeur contient un descripteur pour le nouveau

client. L'exemple suppose que l'opération a abouti. Le code d'erreur peut être testé pour une exécution correcte dans le code de production.

- ADDRESS est l'URI du port MQTT que le démon surveille pour les demandes de connexion client entrantes.
- CLIENTID est le nom utilisé pour identifier le client auprès du démon. Chaque client actif doit avoir un nom unique. Si vous dupliquez un identificateur de client dans deux clients en cours d'exécution, une exception est émise dans les deux clients et un client s'arrête. Le nom est utilisé par le démon pour reconnaître qu'un client se reconnecte à la suite d'une déconnexion. Voir Identificateur de client.
- MQTTCLIENT_PERSISTENCE_NONE indique que l'état du client est conservé en mémoire et qu'il est perdu en cas de défaillance du système. MQTTCLIENT_PERSISTENCE#_DEFAULT spécifie la persistance basée sur le système de fichiers, offrant une protection contre les échecs. Pour les applications plus spécialisées, vous pouvez utiliser MQTTCLIENT_PERSISTENCE_USER, qui fournit une interface permettant d'implémenter votre propre mécanisme de persistance. La question de savoir si la persistance est requise est une question de conception d'application. Pour plus de détails, voir Persistance des messages.
- Le port TCP/IP du démon par défaut pour MQTT est 1883. Dans l'exemple, l'adresse par défaut est `tcp://localhost:1883`.
- Tant que vous n'appelez pas la fonction `MQTTClient_connect` , aucun traitement de message n'a lieu.

8. Connecter le client au démon

```
if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
    printf("Failed to connect, return code %d\n", rc);
    exit(-1);
}
```

- La fonction `MQTTClient_connect` est appelée, transmettant le descripteur client et un pointeur vers les options de connexion en tant qu'arguments.
- Le code retour de l'appel `MQTTClient_connect` est testé pour s'assurer que la demande de connexion a abouti.
- Si l'appel de connexion échoue, le programme se termine avec le code d'erreur -1.
- Une fois l'application connectée, elle peut commencer à publier et à s'abonner.
- Un petit message "keep-alive" est envoyé toutes les 20 secondes pour empêcher la fermeture de la connexion TCP/IP. Cette option est définie par `conn_opts.keepAliveInterval`.
- La session est démarrée sans vérification de l'achèvement des messages en cours restants d'une connexion précédente car *conn_opts.cleansession* est défini sur true. Pour plus de détails, voir Nettoyer les sessions.
- Aucun message de dernière volonté et testament n'est créé pour la connexion. Pour plus de détails, voir Last will and testament

9. Abonnez-vous à la rubrique.

```
MQTTClient_subscribe(client, TOPIC, QOS);
```

- Utilisez la fonction `MQTTClient_subscribe` pour abonner l'application client à la rubrique sélectionnée. Le nom de la rubrique peut inclure des caractères génériques. Pour plus de détails, voir «Chaînes de rubrique et filtres de rubrique dans les clients MQTT», à la page 564.
- Le paramètre QoS détermine la qualité de service maximale appliquée aux messages envoyés à cet abonné. Le serveur envoie les messages à la valeur inférieure de ce paramètre et le paramètre QoS pour le message d'origine.
- Cette fonction renvoie un code d'erreur qui peut être testé pour une exécution correcte dans le code de production.

10. Attendez en boucle que l'utilisateur entre un caractère'Q'à partir du clavier.

```
do {
    ch = getchar();
    } while(ch!='Q' && ch != 'q');
```

Le programme attend maintenant l'arrivée des messages. Dans cet exemple, tous les messages sont traités dans la fonction de rappel `MQTTClient_messageArrived`. Pour plus de détails, voir «Réception de messages», à la page 542.

11. Déconnectez le client du démon.

```
MQTTClient_disconnect(client, 10000);
```

- Le client se déconnecte du serveur et attend que les fonctions de rappel (non utilisées dans cet exemple) soient terminées pour les messages en cours.
- Le second argument indique un délai de mise au repos en millisecondes. L'exemple attend jusqu'à 10 secondes pour terminer tout autre travail qu'il doit effectuer avant de se déconnecter.
- Cette fonction renvoie un code d'erreur qui peut être testé pour une exécution correcte dans le code de production.

12. Libérez de la mémoire utilisée par le client et arrêtez le programme.

```
MQTTClient_destroy(&client);
}
```

Réception de messages

Pourquoi et quand exécuter cette tâche

Lorsque des messages arrivent du serveur, la fonction `MQTTClient_messageArrived` est démarrée. Les étapes qui suivent expliquent le code.

Procédure

1. Démarrez la définition de la fonction de rappel. Cette définition doit correspondre au modèle de fonction `MQTTClient_messageArrived` .

```
int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
```

- `context` permet d'accéder au contexte transmis à la bibliothèque client lorsque la fonction `MQTTClient_setCallbacks` a été appelée. Cette fonction n'est pas utilisée dans l'exemple.
- `topicName` est un pointeur vers la rubrique dans laquelle le message reçu est publié. Si vous vous êtes abonné à l'aide de caractères génériques, ce paramètre identifie la rubrique spécifique utilisée pour le message.
- `topicLen` est la longueur de la chaîne de rubrique. Cette option est fournie pour les utilisateurs qui doivent imbriquer des caractères NULL dans des chaînes de rubrique.
- `message` est un pointeur vers la structure `MQTTClient_message` contenant la charge de message et les attributs.

2. Définissez les variables locales utilisées.

```
int i;
char* payloadptr;
```

Ces variables sont utilisées dans l'exemple pour imprimer le contenu en le répétant.

3. Imprimer un message, en affichant la rubrique et la charge du message

```
printf("Message arrived\n");
printf("     topic: %s\n",topicName);
printf("   message: ");
payloadptr = message->payload;
for(i=0; i<message->payloadlen; i++){
    putchar(*payloadptr++);
```

```
    }
putchar('\n');
```

- L'exemple suppose que le contenu reçu est une séquence de caractères imprimables.
- Un contenu MQTT est un tableau d'octets. L'application est responsable de l'interprétation de leur signification.

4. Libérez la mémoire utilisée pour stocker le message.

```
MQTTClient_freeMessage(&message);
MQTTClient_free(topicName);
```

- Dans l'exemple, tous les messages sont traités dans la fonction de rappel.
- Vérifiez que les fonctions de rappel sont courtes et renvoyez le contrôle à l'unité d'exécution appelante dès que possible.
- Le pointeur de message est transmis pour être traité dans la partie principale du programme.
- Le programme principal doit libérer la mémoire utilisée par le message lorsque le traitement est terminé. `MQTTClient_freeMessage()` est une fonction pratique qui renvoie au système les deux blocs de mémoire utilisés pour contenir la structure *MQTTClient_message* et la charge de message. La mémoire allouée à `topicName` doit être libérée séparément, comme indiqué.

5. Renvoie une valeur true lorsque le rappel a correctement traité le message

```
    return 1;
}
```

- Le renvoi d'une valeur true indique que la bibliothèque client peut traiter le message comme étant correctement distribué.
- Si la fonction de rappel ne peut pas traiter correctement le message, une valeur false est renvoyée. Par exemple, si le rappel place des messages dans une file d'attente pour que le programme principal les traite et que la file d'attente est saturée, le renvoi de la valeur false est approprié.
- Pour les messages QoS1 et QoS2 , le renvoi d'une valeur false indique que le message n'a pas été distribué et que d'autres tentatives de distribution sont effectuées.

Exemple de code

```
#include "stdio.h"
#include "stdlib.h"
#include "MQTTClient.h"
#include "settings.h"
#include "callback.h"

int main(int argc, char* argv[]) {
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    int rc;
    int ch;

    MQTTClient_create(&client, ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);

    MQTTClient_setCallbacks(client, NULL, connlost, msgarrvd, delivered);

    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS) {
        printf("Failed to connect, return code %d\n", rc);
        exit(-1);
        }
    printf("Subscribing to topic %s\nfor client %s using QoS%d\n\n"
           "Press Q<Enter> to quit\n\n", TOPIC, CLIENTID, QOS);

    MQTTClient_subscribe(client, TOPIC, QOS);
    do {
        ch = getchar();
        } while(ch!='Q' && ch != 'q');
    MQTTClient_disconnect(client, 10000);
    MQTTClient_destroy(&client);
}
```

Figure 117. `subscriber.c`

```
#include "MQTTClient.h"

volatile MQTTClient_deliveryToken deliveredtoken;

void delivered(void *context, MQTTClient_deliveryToken dt) {
    printf("Message with token value %d delivery confirmed\n", dt);
    deliveredtoken = dt;
}

int msgarrvd(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    int i;
    char* payloadptr;

    printf("Message arrived\n");
    printf("     topic: %s\n", topicName);
    printf("   message: ");

    payloadptr = message->payload;
    for(i=0; i<message->payloadlen; i++) {
        putchar(*payloadptr++);
        }
    putchar('\n');
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}
void connlost(void *context, char *cause) {
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}
```

Figure 118. `callback.h`

```
#define ADDRESS    "tcp://localhost:1883"
#define CLIENTID   "ExampleClientSub"
#define TOPIC      "MQTT Example"
#define PAYLOAD    "Hello World!"
#define QOS        1
#define TIMEOUT    10000L
```

Figure 119. `settings.h`

Concepts de programmation du client MQTT

Les concepts décrits dans cette section vous aident à comprendre les bibliothèques client Java, JavaScript et C pour la version 3.1 de MQTT protocol. Ces concepts complètent la documentation d'API accompagnant les bibliothèques client.

`com.ibm.micro.client.mqttv3` contient les classes qui fournissent les méthodes publiques pour les bibliothèques client pour le protocole MQTT version 3.1 . Une version du package `com.ibm.micro.client.mqttv3` , ainsi que les packages qui l'accompagnent et qui implémentent le protocole pour Java SE et ME, sont fournis avec l'installation de IBM WebSphere MQ Telemetry. Pour obtenir la version la plus récente des bibliothèques client MQTT (Java, JavaScript) et pour afficher ou télécharger la documentation de l'API, voir "MQTT client programming reference".

Pour développer et exécuter un client MQTT, vous devez copier ou installer ces packages sur l'appareil 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 la version 3.1 de MQTT protocol. Vous pouvez implémenter vos propres clients dans différents langages adaptés à différentes plateformes de dispositif. Voir Format et protocole MQ Telemetry Transport.

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 WebSphere 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 , autant qu'elles le peuvent, 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

L'interface `MqttCallback` est dotée de trois méthodes de rappel ; voir un message d'implémentation dans Callback.java.

`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(MqttDeliveryToken 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 551. 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 des attributs `MqttDeliveryToken` et `MqttMessage` une fois que le message a été publié. Si vous tentez de définir des attributs `MqttMessage` après la publication du message, le résultat est indéfini.

Le client MQTT continue à traiter les accusés de réception de la distribution si le client se reconnecte à la session précédente avec le même `identificateur client`. Voir «Sessions propres», à la page 548. 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(MqttTopic 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 plusieurs manières différentes afin de traiter une publication à la fois :

1. Pour mettre en pause le client d'application jusqu'à la fin de la distribution de la publication, voir Figure 88, à la page 499.
2. Pour réaliser la synchronisation avec la méthode `MqttCallback.deliveryComplete`. `token.waitForCompletion` ne reprend que lorsque `MqttCallback.deliveryComplete` revient au client MQTT. 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

Le Tableau 70, à la page 547 décrit les méthodes du client Java MQTT 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 70. 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.	Aucune.
`MqttDeliveryToken.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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Sessions propres

Le client MQTT et le service de télémétrie (MQXR) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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 WebSphere MQ peut fermer une session ouverte et supprimer toutes les informations de session par la force. 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 l'objet par défaut `MqttConnectOptions`, ou que vous attribuez à `MqttConnectOptions.cleanSession` la valeur `true` avant de connecter le client, les anciens abonnements pour le 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 attribuez à `MqttConnectOptions.cleanSession` la valeur `false` avant la connexion, les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour ce 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 ; ce mode dure pendant la totalité de la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez des modes `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimées.

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 , autant qu'elles le peuvent, 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Identificateur de client

L'identificateur 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. Supposez, par exemple, qu'un dispositif est défaillant et que vous en êtes averti par un client qui appelle le centre d'assistance. Comment le client procède-t-il pour identifier le dispositif, et comment corréler cette identification avec le serveur qui est généralement connecté au client ? Devez-vous consulter une base de données qui mappe chaque dispositif à un identificateur client et à un serveur ? Le nom du dispositif permet-il d'identifier le serveur auquel ce dispositif est connecté ? Lorsque vous parcourez les connexions client MQTT , chaque connexion est libellée avec l'identificateur client. Devez-vous consulter un tableau pour mapper un identificateur client à un dispositif physique ?

L'identificateur client identifie-t-il un dispositif particulier ou une application s'exécutant sur ce client ? Si un client remplace un dispositif défectueux par un nouveau, le nouveau dispositif possède-t-il le même identificateur client que l'ancien ? Allez vous allouer un nouvel identificateur ? Si vous changez un dispositif physique, mais que vous gardez le même identificateur, les publications en suspens et les publications actives sont automatiquement transférées vers le nouveau dispositif.

Comment vous assurer que les identificateurs client sont uniques ? De même qu'un système génère des identificateurs uniques, vous devez faire appel à un processus fiable pour définir l'identificateur sur le client. Le dispositif client est peut-être une "boîte noire", sans aucune interface utilisateur. Est-ce que vous fabriquez le dispositif avec un identificateur client - tel que l'utilisation de l'adresse MAC ? Ou disposez-vous d'une installation logicielle et d'un processus de configuration qui configure le dispositif avant son activation ?

Vous pouvez créer un identificateur client à partir d'une adresse MAC du dispositif 48 bits, pour que l'identificateur soit court et unique. Si la taille de n'est pas une question critique, vous pouvez utiliser les 17 octets restants afin de faciliter la gestion 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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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 la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere 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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Publication Last will and testament (dernières volontés et testament)

Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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 WebSphere MQ, et 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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Persistance des messages dans les clients MQTT

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere 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 MessageSight et IBM WebSphere 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 WebSphere 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 supprimé du client uniquement lorsqu'il n'est plus nécessaire de garantir la distribution "au moins une fois", QoS=1ou "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 MessageSight et IBM WebSphere MQ en tant que message persistant. S'il est marqué comme QoS=0, il est mis en file d'attente dans IBM MessageSight et IBM WebSphere MQ en tant que message non persistant. Dans IBM WebSphere 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 messages 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 client MQTT pour 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`. Le client dispose également d'une implémentation par défaut pour Java ME, `MqttDefaultMIDPPersistence`.

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` ou `MqttDefaultMIDPPersistence`.

`MqttPersistable`
`MqttClientPersistence` extrait et insère les objets `MqttPersistable` à l'aide d'une clé de protection. Vous devez fournir une implémentation de `MqttPersistable` ainsi qu'une implémentation de `MqttClientPersistence` si vous n'utilisez pas la classe `MqttDefaultFilePersistence` ou `MqttDefaultMIDPPersistence`.

`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 aussi instancier `MqttDefaultFilePersistence` et placer les fichiers dans un répertoire par défaut. Le nom du répertoire est *`client identifier`*`-tcp` *`hostname portnumber`*. `"\", "\\", "/", ":"` et `" "` sont supprimés de la chaîne de nom de répertoire.

Le chemin d'accès au répertoire est la valeur de la propriété système `rcp.data`. Si `rcp.data` n'est pas défini, le chemin est la valeur de la propriété système `usr.data`.

`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.

`MqttDefaultMIDPPersistence`
`MqttDefaultMIDPPersistence` a un constructeur par défaut et aucun paramètre. Il utilise le package `javax.microedition.rms.RecordStore` pour stocker les messages.

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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ 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 des messages autorisée par le protocole MQTT est 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 WebSphere 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` et `qos`, mais ce contenu peut changer à l'avenir.

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.

Le client MQTT peut également créer un objet de message temporaire pour vous 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 file d'attente ou IBM MessageSight est responsable de la création 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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Qualités de service fournies par un client MQTT

Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere 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 protocole MQTT ne nécessite pas que les serveurs envoient les publications avec `QoS=0` au 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. En conséquence, le destinataire peut recevoir le même message à plusieurs reprises, et 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 le cadre de la seconde paire de transmissions, l'expéditeur dit au destinataire qu'il peut terminer le traitement du message, `"PUBREL"`. Si l'expéditeur ne reçoit pas d'accusé de réception du message `"PUBREL"`, le message `"PUBREL"` est renvoyé jusqu'à la réception de l'accusé de réception. L'expéditeur supprime le message qu'il a enregistré lors de la réception de l'accusé de réception au 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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)

Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Publications conservées et clients MQTT

Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere MQ.

Publications conservées et clients MQTT

Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

Utilisez la méthode `MqttMessage.setRetained` pour indiquer si la publication sur une rubrique est conservée ou non.

Pour supprimer une publication conservée dans IBM WebSphere MQ, exécutez la commande **CLEAR TOPICSTR** MQSC.

Si vous créez une publication avec une charge utile null, la publication vide est envoyée aux abonnements. Les autres courtiers MQTT peuvent ne pas envoyer de publication vide aux abonnés.

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.

Lorsque vous créez ou mettez à jour une publication conservée, envoyez la publication avec un QoS ou 1 ou 2. Si vous l'envoyez avec un QoS de 0, IBM WebSphere MQ crée une publication conservée non permanente. La publication n'est pas conservée en cas d'arrêt du gestionnaire de files d'attente.

Utilisez les publications conservées pour enregistrer la dernière valeur d'une mesure. Les nouveaux abonnés à la rubrique conservée reçoivent immédiatement la valeur de mesure la plus récente. 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.

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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications

Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere 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 WebSphere 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 WebSphere MQ peut supprimer un abonnement. Répertorie les abonnements à l'aide de WebSphere MQ Explorer ou à l'aide de commandes **runmqsc** ou PCF. Tous les abonnements de client MQTT sont nommés. Ils reçoivent un nom au format suivant: *ClientIdentifier*:*Topic name*

Si vous utilisez l'objet par défaut `MqttConnectOptions`, ou que vous attribuez à `MqttConnectOptions.cleanSession` la valeur `true` avant de connecter le client, les anciens abonnements pour le 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 attribuez à `MqttConnectOptions.cleanSession` la valeur `false` avant la connexion, les abonnements créés par le client sont ajoutés à tous les abonnements qui existaient pour ce 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 ; ce mode dure pendant la totalité de la session. Pour modifier son paramètre, vous devez déconnecter et reconnecter le client. Si vous passez des modes `cleanSession=false` à `cleanSession=true`, tous les abonnements précédents pour le client et toutes les publications qui n'ont pas été reçues sont supprimées.

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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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 WebSphere 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 WebSphere 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 de rubrique 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 WebSphere 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 client MQTT pour 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 WebSphere MQ par les clients MQTT doivent suivre la convention de nom des gestionnaires de files d'attente. En particulier, les chaînes de rubrique ne peuvent pas contenir de traits d'union.
2. La longueur maximale est différente. Les chaînes de rubrique IBM WebSphere 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 WebSphere Message Broker, un niveau de rubrique null, `'...//...'` n'était pas valide. Les niveaux de rubrique Null sont pris en charge par IBM WebSphere MQ.

Contrairement à la publication/l'abonnement IBM WebSphere 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 WebSphere 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 WebSphere MQ est décrite dans Schéma de caractères génériques basés sur des rubriques. La syntaxe des filtres de sujet que vous pouvez construire avec un client MQTT est décrite dans la classe `MqttClient` de la documentation d'API du client MQTT pour 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 filtre de rubrique est presque identique. La seule différence réside dans la manière dont différents courtiers MQTT interprètent un filtre de sujet. Dans WebSphere Message Broker V6, un caractère générique multi-niveau pouvait uniquement être utilisé à la fin du filtre de rubrique. Dans IBM WebSphere MQ, un caractère générique à plusieurs niveaux peut être utilisé à n'importe quel niveau de l'arborescence de rubriques ; par exemple, `USA/#/Dutchess County`.

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 , autant qu'elles le peuvent, 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) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT 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

Jetons de distribution

Publication Last will and testament (dernières volontés et testament)
Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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

Publications
Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT peut créer des publications à envoyer à IBM WebSphere MQet s'abonner à des rubriques sur IBM WebSphere MQ MQ pour recevoir des publications.

Qualités de service fournies par un client MQTT
Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

Publications conservées et clients MQTT
Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

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.

Concepts de programmation client C

Les différences entre le client C et Java pour la version 3.1 de MQ Telemetry Transport sont décrites dans cette rubrique. La rubrique complète les concepts client et les informations de référence C.

La rubrique est organisée de la même manière que «Concepts de programmation du client MQTT», à la page 544. Chaque en-tête correspond à une rubrique dans *WebSphere(r) MQ Telemetry Transport client programming concepts*. Les sections décrivent les différences entre le client C et le client Java. Les petites différences dans les signatures entre les méthodes Java et les fonctions C ne sont pas décrites.

Le client C est le plus souvent utilisé pour implémenter un adaptateur léger entre un dispositif de télémétrie et le démon pour dispositifs WebSphere MQ Telemetry . Le démon est couramment utilisé comme concentrateur de réseau entre des dispositifs de télémétrie très légers et le service de télémétrie (MQXR).

Le démon pour dispositifs WebSphere MQ Telemetry est également un client C et les différences dans son comportement par rapport au service de télémétrie (MQXR) sont décrites. Le démon ne fournit pas d'implémentation de JAAS ou SSL pour les clients qui s'y connectent.

`mqttclient.dll` et `mqttclient.lib` sont les bibliothèques Windows 32 bits qui contiennent les fonctions client pour l'implémentation C du protocole MQ Telemetry Transport version 3.1 . Les bibliothèques Linux 32 bits sont `libmqttclient.so` et `libmqttclient.a`. Deux fichiers d'en-tête contiennent la fonction et d'autres déclarations requises par les applications client: `MQTTClient.h` et `MQTTClientPersistence.h`. Ces fichiers sont fournis avec l'installation de WebSphere MQ Telemetry.

Pour développer et exécuter un client MQ Telemetry Transport, vous devez copier ces fichiers sur le dispositif client. Contrairement aux clients WebSphere MQ , vous n'avez pas besoin d'installer un environnement d'exécution client distinct.

Consultez les conditions de licence associées à la fonction WebSphere MQ Telemetry qui régit la connexion des clients MQ Telemetry Transport à WebSphere MQ et au démon pour dispositifs WebSphere MQ Telemetry .

Le client C est une implémentation de référence de la version 3.1 de MQ Telemetry Transport. Vous pouvez implémenter vos propres clients dans différents langages adaptés à différentes plateformes de dispositif. Pour plus de détails, reportez-vous à la rubrique Format et protocole MQ Telemetry Transport.

Identificateur du client MQTT

«Identificateur de client», à la page 550	L'identificateur 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.

- Aucune différence.

Publications

«Publications», à la page 557	Les publications sont des instances de `MqttMessage` qui sont associées à une chaîne de rubrique. Le client MQTT

- La fonction de rappel n'est pas appelée pour les publications avec "fire and forget", `QoS=0`, qualité de service.

Jetons de distribution

«Jetons de distribution», à la page 551	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.

- Un jeton de distribution est un `int`. Il a un typedef de `MQTTClient_deliveryToken`
- La fonction de rappel n'est pas appelée pour les publications avec "fire and forget", `QoS=0`, qualité de service.

Publications conservées

«Publications conservées et clients MQTT», à la page 561	Si vous créez un abonnement à une rubrique ayant des publications conservées, la dernière publication conservée dans la rubrique vous est immédiatement envoyée.

- Les messages conservés ne sont sauvegardés dans le démon que si la persistance est configurée ; voir Sauvegarde des messages et des abonnements conservés.

  Pour WebSphere MQ, la qualité de service détermine si un message conservé est sauvegardé définitivement. Si un client est connecté au service de télémétrie, les messages conservés avec "fire and forget", la qualité de service `QoS=0` sont supprimés si le gestionnaire de files d'attente est arrêté.

Abonnements

«Abonnements», à la page 562	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.

- Les abonnements durables ne sont sauvegardés dans le démon que si la persistance est configurée ; voir Sauvegarde des messages et des abonnements conservés.
- Les publications peuvent être reçues de manière synchrone. Appelez la fonction `MQTTClient_receive` .

Rappels et synchronisation

«Rappels et synchronisation dans les applications client MQTT», à la page 545	Le modèle de programmation du client MQTT utilise les unités d'exécution de manière extensive. Les unités d'exécution découplent une application client MQTT , autant qu'elles le peuvent, 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`.

- L'opération de synchronisation dans le client C est modale. L'appel de `MQTTClient_setCallback` place le client en mode asynchrone.
- En mode synchrone, le client d'application doit volontairement céder le contrôle afin que le client MQTT puisse traiter les accusés de réception et émettre des commandes ping MQTT pour maintenir le réseau actif. Contrôle des rendements en appelant `MQTTClient_receive` ou `MQTTClient_yield`.

Chaînes de rubrique et filtres

«Chaînes de rubrique et filtres de rubrique dans les clients MQTT», à la page 564	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 WebSphere MQ.

- Le démon pour dispositifs WebSphere MQ Telemetry gère le caractère générique à plusieurs niveaux # différemment de WebSphere MQ v7. /# doit être les deux derniers caractères de la chaîne de

filtrage pour que # se comporte comme un caractère générique. Dans WebSphere MQ v7, `../#/..` est une utilisation valide du caractère générique multiniveau. Le démon pour dispositifs WebSphere MQ Telemetry traite le caractère générique multi-niveau de la même manière que WebSphere MQ Broker v6.

Qualité de service

«Qualités de service fournies par un client MQTT», à la page 559	Un client MQTT fournit trois qualités de service pour distribuer les publications à WebSphere MQ et au client MQTT : "au plus une fois", "au moins une fois" et "exactement une fois". Lorsqu'un client MQTT envoie une demande à WebSphere MQ pour créer un abonnement, la demande est envoyée avec la qualité de service "au moins une fois".

- Aucune différence.

Persistance des messages

«Persistance des messages dans les clients MQTT», à la page 554	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.

- En raison des différences de liaison de langage, définissez le mécanisme de persistance des messages dans le client C comme suit. Appelez le client MQTT C avec l'une des trois options définies comme quatrième paramètre à `MQTTClient_create`:

  `MQTTCLIENT_PERSISTENCE_DEFAULT`
  Persistance basée sur un fichier, dont les détails sont spécifiques à la plateforme client.

  `MQTTCLIENT_PERSISTENCE_NONE`
  Les données sont uniquement conservées en mémoire et perdues lorsque le client s'arrête. Le démon pour dispositifs WebSphere MQ Telemetry ne prend en charge que cette option.

  `MQTTCLIENT_PERSISTENCE_USER`
  Vous pouvez développer des fonctions pour implémenter votre propre mécanisme de persistance. Transmettez une structure, `MQTTClient_persistence` , contenant des pointeurs vers vos fonctions sur l'appel `MQTTClient_create` . Pour plus de détails, consultez les informations de référence du client MQTT C.

Sessions propres

«Sessions propres», à la page 548	Le client MQTT et le service de télémétrie (MQXR) conservent des informations sur l'état de la session. Les informations d'état de session permettent de garantir la distribution "au moins une fois" et "une seule fois", et la réception "une seule fois" des publications. L'état de la session comprend également les abonnement créés par le client MQTT. Vous pouvez choisir d'exécuter le client MQTT sans conserver les informations d'état entre les sessions. Modifiez le mode de nettoyage de session en définissant `MqttConnectOptions.cleanSession` avant de vous connecter.

- Aucune différence.

Dernière volonté et testament

«Publication Last will and testament (dernières volontés et testament)», à la page 553	Si la connexion d'un client MQTT s'arrête inopinément, vous pouvez configurer WebSphere 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.

- Aucune différence.

Traitement des erreurs de programme

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:

- Codes anomalie pour toutes les autres plateformes WebSphere MQ

Concevez vos programmes pour gérer tous les codes retour qui peuvent survenir à chaque appel.

Interruptions du système

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:

Systèmes UNIX, Linux et Windows
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 335). 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. Avec les systèmes WebSphere MQ sur les systèmes UNIX, Linuxet Windows , un appel MQGET ou MQPUT dans votre application échoue au moment du remplissage de tous les fichiers journaux, avec le message MQRC_RESOURCE_PROBLEM. Pour plus d'informations sur les fichiers journaux sur les systèmes AIX, HP-UX, Linux, Solaris et Windows , voir Administration.

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 UNIX, Linuxet 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.

Sur les systèmes WebSphere MQ for Windows, UNIX et Linux , le nombre d'annulations survit toujours aux redémarrages du gestionnaire de files d'attente.

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 12.

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 413.

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 12 .

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 WebSphere 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 573.

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 WebSphere MQ for z/OS utilisant 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* correspond au *ApplId* du système CICS suivi 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 de tout code anomalie MQRC_ * pouvant 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 WebSphere MQ telles que la connexion à un gestionnaire de files d'attente et la génération de rapports d'exception.

WebSphere MQ La multidiffusion a été conçue pour être aussi transparente que possible pour l'utilisateur et tout en étant 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 WebSphere MQ existantes ne nécessitent pas de réécriture substantielle pour utiliser la multidiffusion. Toutefois, il peut y avoir des limitations (voir «Multidiffusion et interface de file d'attente de messages», à la page 574 pour plus d'informations) et des problèmes de sécurité à prendre en compte (voir Sécurité multidiffusion pour plus d'informations).

Multidiffusion et interface de file d'attente de messages

Utilisez ces informations pour comprendre les principaux concepts MQI et la façon dont ils sont liés à la multidiffusion WebSphere MQ .

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` .

WebSphere MQ Multicast prend en charge l'utilisation de 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 23.

Le tableau suivant répertorie tous les principaux concepts MQI et la manière dont ils sont liés à la multidiffusion:

Tableau 71. 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	Rejeté	2005 (07D5) (RC2005): MQRC_BUFFER_LENGTH_ERROR
Regroupement	Rejeté	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Segmentation	Rejeté	2443 (098B) (RC2443): MQRC_SEGMENTATION_NOT_ALLOWED
Listes de diffusion	Rejeté	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
	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)	Rejeté	2072 (0818) (RC2072) : MQRC_SYNCPOINT_NOT_AVAILABLE
Navigation dans les messages	Rejeté	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Messages de verrouillage	Rejeté	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
Exploration avec marque	Rejeté	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Transmettre un contexte	Rejeté	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

Tableau 71. Concepts MQI et leurs relations avec la multidiffusion (suite)

Concept MQI	Action lors d'une tentative d'utilisation de la multidiffusion	Code raison
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
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	Rejeté	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 WebSphere MQ Opération d'égal à égal en continu multidiffusion ; bien que WebSphere MQ se connecte à un gestionnaire de files d'attente WebSphere 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 client automatisée

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 multidiffusion WebSphere MQ .

WebSphere MQ La multidiffusion 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 WebSphere MQ est requise, la conversion est effectuée localement sur chaque client.

Les données sont converties sur le client pour le trafic de multidiffusion WebSphere 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 432 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 WebSphere MQ les gestionnaires d'événements de multidiffusion et la génération de rapports WebSphere MQ les exceptions de multidiffusion.

WebSphere MQ La multidiffusion facilite 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 WebSphere MQ standard.

Un événement de multidiffusion individuel peut entraîner l'appel de plusieurs événements WebSphere 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 WebSphere MQ .

La constante WebSphere MQ `MQCBDO_EVENT_CALL` permet aux applications d'enregistrer un rappel pour recevoir uniquement des événements WebSphere 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

WebSphere 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 WebSphere 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

WebSphere MQ Les exceptions de multidiffusion incluent certaines 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 72. 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	N/A
MQMCEV_VERSION_CONFLIT	Réception de paquets de version de protocole plus récents	N/A
FIABILITÉ_MQMCEV_	Différents modes de fiabilité de l'émetteur et du récepteur	N/A
MQMCEV_FERME_TRANS	La transmission de rubrique est fermée par 1 source	N/A
Erreur MQMCEV_STREAM_ERROR	Erreur détectée sur le flux	N/A
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	N/A
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	N/A

Tableau 72. Descriptions des codes d'événement de multidiffusion (suite)

Code d'événement	Description	Données supplémentaires
MQMCEV_REPAIR_DELAY	Le récepteur de multidiffusion n'a pas reçu de paquet de réparation pour un NAK en attente	N/A
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
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	N/A
MQMCEV_REPAIR_ALERT_OFF	Le débit d'envoi des paquets de réparation de l'émetteur est inférieur à la normale	N/A
MQMCEV_SHM_DEST_INUTILISABLE	La région de mémoire partagée utilisée par une destination de rubrique de l'émetteur a été détectée comme étant inutilisable	N/A
MQMCEV_PORT_SHM_INUTILISABLE	Le port de mémoire partagée utilisé par une instance de récepteur a été détecté comme étant inutilisable	N/A
MQMCEV_CCT_GETTIME_FAILED	Echec de l'obtention de l'heure à partir du temps de cluster coordonné	N/A
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	

Tableau 72. Descriptions des codes d'événement de multidiffusion (suite)

Code d'événement	Description	Données supplémentaires
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 de RMM
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 de RMM

Utilisation de .NET

Les classes WebSphere MQ pour .NET permettent à un programme écrit dans l'infrastructure de programmation .NET de se connecter à WebSphere MQ en tant que client WebSphere MQ MQI ou de se connecter directement à un serveur WebSphere MQ .

Si vous disposez d'applications qui utilisent Microsoft .NET Framework et que vous souhaitez tirer parti des fonctions de WebSphere MQ, vous devez utiliser les classes WebSphere MQ pour .NET.

L'interface orientée objet WebSphere MQ .NET est différente de l'interface MQI car elle utilise des méthodes d'objets plutôt que des instructions MQI.

L'interface de programmation d'application WebSphere MQ 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 WebSphere 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 WebSphere 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();
```

WebSphere MQ classes for .NET est un ensemble de classes qui permettent aux applications .NET d'interagir avec WebSphere MQ. Ils représentent les différents composants de WebSphere MQ utilisés par votre application, tels que les gestionnaires de files d'attente, les files d'attente, les canaux et les messages. Pour plus de détails sur ces classes, voir Classes et interfaces WebSphere MQ .NET.

Avant de pouvoir compiler des applications que vous écrivez, vous devez avoir installé .NET Framework. Pour obtenir des instructions sur l'installation des classes WebSphere MQ pour .NET et .NET Framework, voir «Installation des classes WebSphere MQ pour .NET», à la page 583.

Concepts associés

Présentation technique

«Options de connexion», à la page 582
Il existe trois modes de connexion des classes WebSphere MQ pour .NET à un gestionnaire de files d'attente. Déterminez le type de connexion qui convient le mieux à vos besoins.

«Utilisation de WebSphere MQ classes for .NET», à la page 594
Cette collection de rubriques explique comment configurer votre système pour qu'il exécute les exemples de programme afin de vérifier votre installation WebSphere MQ classes for .NET et comment exécuter vos propres programmes.

«Résolution des problèmes WebSphere MQ .NET», à la page 597
Si un programme n'aboutit pas, exécutez l'un des exemples d'application et suivez les conseils donnés dans les messages de diagnostic.

«Ecriture et déploiement de programmes WebSphere MQ .NET», à la page 597
Pour utiliser WebSphere MQ classes for .NET afin d'accéder aux files d'attente WebSphere MQ , vous pouvez écrire des programmes dans n'importe quel langage pris en charge par .NET contenant des appels qui placent des messages dans des files d'attente WebSphere MQ et en extraire des messages.

«IBM WebSphere MQ canal personnalisé pour Microsoft Windows Communication Foundation (WCF)», à la page 617
Le canal personnalisé Microsoft Windows Communication Foundation (WCF) pour IBM WebSphere MQ envoie et reçoit des messages entre les clients et les services WCF.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Développement d'applications», à la page 7
IBM WebSphere MQ fournit plusieurs façons de développer des applications pour envoyer et recevoir les 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.

Initiation aux classes WebSphere MQ pour .NET

Les classes WebSphere MQ pour .NET permettent à un programme écrit dans l'infrastructure de programmation .NET de se connecter à WebSphere MQ en tant que client WebSphere MQ MQI ou de se connecter directement à un serveur WebSphere MQ .

Options de connexion

Il existe trois modes de connexion des classes WebSphere MQ pour .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 WebSphere MQ classes for .NET en tant que client WebSphere MQ MQI, vous pouvez l'installer avec le client WebSphere MQ MQI, soit sur la machine du serveur WebSphere MQ , soit sur une machine distincte. Une connexion de liaisons client peut utiliser des transactions XA ou non XA

Connexion de liaisons de serveur

Lorsqu'elles sont utilisées en mode liaisons de serveur, les classes WebSphere MQ for .NET utilisent l'API du gestionnaire de files d'attente au lieu de communiquer via un réseau. Cela offre de meilleures performances pour les applications WebSphere MQ que l'utilisation de connexions réseau.

Pour utiliser la connexion de liaisons, vous devez installer les classes WebSphere MQ pour .NET sur le serveur WebSphere MQ .

Connexion client gérée

Une connexion établie dans ce mode se connecte en tant que client WebSphere MQ à un serveur WebSphere MQ s'exécutant sur la machine locale ou distante.

Les classes WebSphere MQ pour .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 597.

Installation des classes WebSphere MQ pour .NET

WebSphere MQ classes for .NET, y compris les exemples, est installé avec WebSphere MQ. Il existe un prérequis pour Microsoft .NET Framework.

La version la plus récente de WebSphere MQ classes for .NET est installée par défaut dans le cadre de l'installation standard de WebSphere MQ dans la fonction *Java and .NET Messaging and Web Services* . Pour obtenir des instructions d'installation, voir Installation d'un serveur IBM WebSphere MQ sur Windows ou Installation d'un client IBM WebSphere MQ sur des systèmes Windows .

Dans un environnement à installations multiples, si vous avez précédemment installé les classes WebSphere MQ pour .NET en tant que module de prise en charge, vous ne pouvez pas installer WebSphere MQ sauf si vous avez d'abord désinstallé le module de prise en charge. La fonction WebSphere MQ classes for .NET installée avec WebSphere MQ contient la même fonctionnalité que le module de prise en charge.

Des exemples d'application, y compris des fichiers source, sont également fournis ; voir «Modèles d'application», à la page 594.

Pour exécuter WebSphere MQ classes for .NET sur des plateformes 32 bits ou 64 bits, vous devez avoir installé Microsoft .NET Framework V2.0 ou version ultérieure.

Remarque : Si Microsoft .NET Framework v2.0 ou version ultérieure n'est pas installé avant l'installation de WebSphere MQ V7.0.1, l'installation du produit WebSphere MQ se poursuivra sans erreur, mais les classes WebSphere MQ pour .NET ne seront pas disponibles. Si .NET Framework est installé après l'installation de WebSphere MQ 7.0.1, les assemblages WebSphere .NET doivent être enregistrés en exécutant le script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , où *WMQInstallDir* est le répertoire dans lequel WebSphere MQ 7.0.1 est installé. Ce script installe les assemblages requis dans le cache d'assemblage global (GAC). Un ensemble de fichiers `amqi*.log` enregistrant les actions effectuées sont créés dans le répertoire `%TEMP%` .

Pour plus d'informations sur l'utilisation du canal personnalisé WebSphere MQ pour Microsoft WCF avec .NET 3, voir: «IBM WebSphere MQ canal personnalisé pour Microsoft Windows Communication Foundation (WCF)», à la page 617

Transactions distribuées 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.

WebSphere MQ Les classes .NET prennent déjà en charge les transactions réparties dans les connexions en mode liaisons de serveur et non gérées. Dans ces modes, les classes WebSphere MQ .NET délègue tous ses appels au client de transaction étendue C, qui gère le traitement des transactions pour le compte de .NET.

WebSphere MQ.NET .NET prennent désormais en charge les transactions réparties en mode géré où WebSphere MQ .NET Classes utilise 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 WebSphere MQ. L'application WebSphere MQ .NET peut insérer et extraire des messages à l'aide de la programmation de transaction implicite .NET ou du 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.

WebSphere MQ.NET utilise Microsoft Distributed Transaction Coordinator (MS DTC) comme gestionnaire de transactions, qui coordonne et gère la transaction entre plusieurs gestionnaires de ressources. WebSphere MQ est utilisé comme gestionnaire de ressources.

WebSphere 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», à la page 585
- 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, WebSphere 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

- WebSphere MQ .NET Le client entièrement géré ne prend pas en charge l'élément imbriqué TransactionScope
- Le client non géré WebSphere MQ .NET ne prend pas en charge l'élément imbriqué TransactionScope

Transactions dépendantes de System.Transactions

- WebSphere MQ .NET client entièrement géré prend en charge la fonction de transactions dépendantes fournie par System.Transactions.
- Le client non géré WebSphere MQ .NET ne prend pas en charge la fonction de transactions dépendantes fournie par System.Transactions.

Exemples de produits

Les nouveaux 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 588

Transactions réparties en mode géré

WebSphere MQ Les classes .NET utilisent l'espace de nom System.Transactions pour la prise en charge des transactions distribuées en mode géré. En mode géré, MS DTC coordonne et gère les transactions réparties sur tous les serveurs inclus dans une transaction.

Les classes WebSphere 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 WebSphere 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 WebSphere MQ .NET insère des messages à l'aide d'un 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

WebSphere MQ.NET prennent en charge les connexions non gérées (client) en 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 WebSphere 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. Les classes WebSphere MQ .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 (MTDDC). MTDDC coordonne les transactions qui couvrent des magasins de données ou des ressources distincts. Pour fonctionner, chaque magasin de données doit implémenter son propre gestionnaire de ressources propriétaire.

WebSphere MQ devient compatible avec MTDDC en implémentant une interface (interface de gestionnaire de ressources propriétaire) dans laquelle il gère le mappage des appels DTC XA à des appels WebSphere MQ(X/Open). WebSphere MQ joue le rôle de gestionnaire de ressources.

Lorsqu'un composant tel que COM + demande l'accès à un WebSphere MQ, le composant 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 WebSphere 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 code défaut émet ensuite des appels à MQMTS qui, à son tour, émet des appels à WebSphere MQ pour valider ou annuler la transaction.

Ecriture d'une application WebSphere MQ .NET à l'aide d'un client non géré

Pour s'exécuter 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. .NET Framework 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 WebSphere 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 MTDDC 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 WebSphere 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 `WebSphere 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 des transactions

Cette section décrit le processus de récupération des transactions dans WebSphere MQ .NET XA à l'aide du mode géré.

Présentation

Dans le traitement des transactions réparties, les transactions peuvent être terminées avec succès. Toutefois, il peut exister des scénarios dans lesquels une transaction peut échouer pour de nombreuses raisons. Il peut s'agir d'une panne système, d'une panne matérielle, d'une erreur réseau, de données incorrectes ou non valides, d'erreurs d'application ou de 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.

Modèle de récupération

Dans le modèle de récupération des transactions de Microsoft .NET, le gestionnaire de transactions (System.Transactions, ou le coordinateur de transactions distribuées Microsoft (MS DTC), ou les deux), initie, coordonne et contrôle la récupération des transactions. Les gestionnaires de ressources basés sur le protocole OLE Tx (protocole XA de Microsoft) fournissent les options permettant de configurer le code défaut pour piloter, coordonner et contrôler la récupération 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.

Récupération à l'aide du coordinateur de transaction distribuée Microsoft (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 WebSphere MQ .NET

Cette section décrit comment les transactions distribuées peuvent être récupérées avec les classes WebSphere 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. Les classes WebSphere MQ .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 WebSphere MQ .NET est un processus en deux étapes.

1. Consignation des informations de reprise de transaction.
   - 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.
2. Récupération de transactions à l'aide d'une application de surveillance WmqDotnetXAMonitor.
   - WmqDotnetXAMonitor est une application gérée par .NET qui traite les messages dans SYSTEM.DOTNET.XARECOVERY.QUEUE et récupère les transactions incomplètes

   Si l'agent MCA ne parvient pas à insérer 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.)

SYSTEM.DOTNET.XARECOVERY.QUEUE

Il s'agit d'une file d'attente système qui contient les informations de reprise des transactions incomplètes. Cette file d'attente est créée lorsqu'un gestionnaire de files d'attente est créé.

Remarque : Vous ne devez pas supprimer SYSTEM.DOTNET.XARECOVERY.QUEUE .

Application WMQDotnetXAMonitor

L'application WebSphere MQ .NET XA Monitor surveille un gestionnaire de files d'attente donné et récupère les transactions incomplètes, le cas échéant. Les éléments suivants sont considérés comme des transactions incomplètes et sont récupérés:

transactions incomplètes

- 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 WebSphere MQ est arrêté.
- Si la transaction est préparée, mais que le gestionnaire de transactions est arrêté.

  L'application de surveillance doit être exécutée à partir du même système que votre application client WebSphere MQ .NET. Si des applications s'exécutent sur plusieurs systèmes se connectant au même gestionnaire de files d'attente, l'application de surveillance doit être exécutée à partir de tous les systèmes. Bien que chaque machine client dispose d'une application de moniteur en cours d'exécution pour récupérer l'application, chaque moniteur doit être en mesure d'identifier le message qui correspond à la transaction que le DTC MS local du moniteur en cours coordonnait pour qu'il puisse le réinscrire et le terminer.

Cas d'utilisation de la reprise de transaction pour WebSphere MQ .NET

Les différents scénarios d'utilisation sont les suivants:

- **WebSphere MQ Application utilisant un code défaut unique et une instance de gestionnaire de files d'attente unique:** Dans ce scénario, lorsque vous vous connectez au gestionnaire de files d'attente et exécutez l'unité de travail (UoW) sous la transaction, et si la transaction échoue et devient incomplète, l'application de surveillance récupère la transaction et la termine.

Dans ce scénario, une seule instance de l'application de surveillance est en cours d'exécution, car un seul gestionnaire de files d'attente est associé aux transactions.

- **Plusieurs applications WebSphere MQ utilisant un code défaut unique et une instance de gestionnaire de files d'attente unique:** Dans ce scénario, il existe plusieurs applications WMQ 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 de surveillance les récupère et termine les transactions appartenant à toutes les applications.

  Dans ce scénario, une seule application de surveillance s'exécute, car un gestionnaire de files d'attente est utilisé dans les transactions.

- **Plusieurs applications WebSphere MQ , plusieurs codes d'accès client, différentes instances de gestionnaire de files d'attente:** Dans ce scénario, il existe plusieurs applications WMQ 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 scénario, une seule instance de l'application de surveillance sera exécutée par client (utilisateur ou ordinateur) car chaque client possède son propre gestionnaire de files d'attente utilisé dans les transactions.

- **Plusieurs applications WebSphere MQ , plusieurs codes d'accès client, plusieurs mêmes instances de gestionnaire de files d'attente:** Dans ce scénario, il existe plusieurs applications WMQ sous différents codes d'accès client (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 scénario, une seule instance de l'application de surveillance 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 WebSphere MQ , code défaut unique, différentes instances de gestionnaire de files d'attente:** Dans ce scénario, il existe plusieurs applications WMQ sous un code défaut unique (c'est-à-dire, sur un ordinateur, plusieurs applications WMQ 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 scénario, 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 de surveillance n'est pas en cours d'exécution en arrière-plan, vous pouvez la démarrer.

Utilisation de l'application WMQDotnetXAMonitor

L'application du moniteur XA doit être exécutée manuellement. Il peut être démarré à tout moment. Vous pouvez le démarrer lorsque vous voyez les messages dans 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 WebSphere MQ .NET.

Commande de démarrage de l'application de surveillance

```
WmqDotnetXAMonitor.exe -m <QueueManagerName> -n <ConnectionName> -c <Channel> -i
```

Où

- **n** -Nom de connexion au format hôte (port). Le nom de connexion 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 de surveillance exécute la récupération pour chacun des noms de connexion spécifiés dans la liste séparée par des virgules.
- **c** -Nom du canal.
- **m** -Nom du gestionnaire de files d'attente. Facultatif
- **i** -Fin de la branche heuristique. Facultatif

L'application de surveillance 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, le moniteur XA parcourt la file d'attente à la recherche de messages et vérifie si le message répond aux critères de transaction incomplets.
3. Si l'un des messages répond aux critères de transaction incomplète, le moniteur l'extrait et extrait les informations de reprise de transaction.
4. Il détermine ensuite si les informations de reprise sont liées au code défaut MS local. Si oui, elle procède à la récupération de la transaction. Sinon, il revient pour parcourir le message suivant.
5. Il effectue ensuite des appels au gestionnaire de files d'attente pour récupérer la transaction incomplète.

Paramètres du fichier de configuration de l'application WmqDotNETXAMonitor
Pour surveiller l'application, des entrées peuvent également être fournies à l'aide du fichier de configuration de l'application. Un exemple de fichier de configuration d'application est fourni avec WebSphere MQ .NET. Ce fichier peut être modifié en fonction de vos besoins.

Le fichier de configuration d'application a la priorité la plus élevée lors de la prise en compte des valeurs d'entrée. Si des valeurs d'entrée sont fournies à la fois sur la ligne de commande et dans le fichier de configuration d'application, les valeurs de la configuration d'application sont prises en compte.

Exemple de fichier de configuration d'application.

```
<?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>
```

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/Rollback
Time|ProcessId|ThreadId|Recovery Completed for TransactionId= xxxxx
Time|ProcessId|ThreadId|Current QueueDepth = n
Time|ProcessId|ThreadId|Current MessageId = xxxx
Time|ProcessId|ThreadId|Current Incomplete Transaction being recovered = xxxxx
Time|ProcessId|ThreadId|Actual Outcome of the transaction(as per DTC)= Commit/Rollback
Time|ProcessId|ThreadId|  Recovery Completed for TransactionId= xxxxx
```

Utilisation de WebSphere MQ classes for .NET

Cette collection de rubriques explique comment configurer votre système pour qu'il exécute les exemples de programme afin de vérifier votre installation WebSphere MQ classes for .NET et comment exécuter vos propres programmes.

Configuration de votre gestionnaire de files d'attente pour l'acceptation des connexions client TCP/IP

Pour configurer un gestionnaire de files d'attente afin qu'il accepte les demandes de connexion entrantes provenant des clients:

1. Définissez un canal de connexion serveur:
   a. Démarrez le gestionnaire de files d'attente.
   b. Définissez un exemple de canal appelé NET.CHANNEL[3]:

      ```
      DEF CHL('NET.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
      DESCR('Sample channel for WebSphere MQ classes for .NET')
      ```

2. Démarrez un programme d'écoute:

   ```
   runmqlsr -t tcp [-m qmnqme] [-p portnum]
   ```

   Remarque : Les crochets indiquent des paramètres facultatifs ; *nom_qm* n'est pas requis pour le gestionnaire de files d'attente par défaut et le numéro de port *numéro_port* n'est pas requis si vous utilisez la valeur par défaut (1414).

Modèles d'application

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.

Cinq exemples d'application 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

[3] Dans cet exemple, nous ne prenons pas en compte les implications en matière de sécurité. Pour un système de production, envisagez d'utiliser SSL ou un exit de sécurité. Pour plus d'informations, voir Sécurité .

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 quel langage pris 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 WebSphere 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.

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 WebSphere 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

Pour générer les exemples d'application, un fichier de traitement par lots a été fourni pour chaque langue.

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 sur Microsoft Visual Studio 2003/.NET SDKv1.1, remplacez la commande de compilation:

```
cl /clr:oldsyntax MQ_INSTALLATION_PATH\bin mmqwrld.cpp
```

par

```
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
```

Résolution des problèmes WebSphere MQ .NET

Si un programme n'aboutit pas, exécutez l'un des exemples d'application et suivez les conseils donnés dans les messages de diagnostic.

Ces exemples d'application sont décrits dans «Utilisation de WebSphere MQ classes for .NET», à la page 594.

Si les problèmes persistent et que vous devez contacter l'équipe de maintenance IBM , vous pouvez être invité à activer la fonction de trace.

Traçage du modèle d'application

Pour des instructions sur l'utilisation de la fonction de trace, voir «Traçage des programmes WebSphere MQ .NET», à la page 617.

Messages d'erreur

Le message d'erreur commun suivant peut s'afficher:

Exception non gérée de type'System.IO.FileNotFoundException's'est produite dans un module inconnu

Si cette erreur se produit pour amqmdnet.dll ou amqmdxcs.dll, assurez-vous que les deux sont enregistrés dans le'Global Assembly Cache'ou créez un fichier de configuration qui pointe vers les assemblages amqmdnet.dll et amqmdxcs.dll . Vous pouvez examiner et modifier le contenu du cache d'assemblage à l'aide de mscorcfg.msc, qui est fourni avec l'infrastructure .NET.

Si .NET Framework n'était pas disponible lors de l'installation de WebSphere MQ , il se peut que les classes ne soient pas enregistrées dans le cache d'assemblage global. Vous pouvez réexécuter manuellement le processus d'enregistrement à l'aide de la commande

```
amqidnet -c MQ_INSTALLATION_PATH\bin\amqidotn.txt -l logfile.txt
```

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Les informations relatives à cette installation sont consignées dans le fichier journal spécifié (`logfile.txt` dans cet exemple).

Ecriture et déploiement de programmes WebSphere MQ .NET

Pour utiliser WebSphere MQ classes for .NET afin d'accéder aux files d'attente WebSphere MQ , vous pouvez écrire des programmes dans n'importe quel langage pris en charge par .NET contenant des appels qui placent des messages dans des files d'attente WebSphere MQ et en extraire des messages.

La documentation WebSphere MQ contient uniquement des informations sur les langages C#, C++ et Visual Basic.

Cette collection de rubriques fournit des informations pour vous aider à écrire des applications afin d'interagir avec les systèmes WebSphere MQ . Pour plus de détails sur les classes individuelles, voir The WebSphere MQ .NET classes and interfaces.

Différences de connexion

La façon dont vous programmez pour WebSphere MQ .NET comporte des dépendances sur les modes de connexion que vous souhaitez utiliser.

Connexions client gérées

Lorsque des classes WebSphere MQ pour .NET sont utilisées en tant que client géré, il existe un certain nombre de différences par rapport à un client WebSphere MQ MQI standard.

Les fonctions suivantes ne sont pas disponibles pour un client géré:

- Compression de canal
- Prise en charge de SSL
- 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** , un canal de connexion serveur à un client géré .NET peut prendre plus de temps à se fermer que les canaux de connexion serveur à d'autres clients.

Si vous avez défini *NMQ_MQ_LIB* sur `managed` afin d'utiliser les diagnostics d'incident WebSphere 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 géré. Net 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 font uniquement partie du package client à partir de la version 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, WebSphere MQ .NET émet une exception.

Lorsque NMQ_MQ_LIB est défini sur "managed", le client utilise les tests de diagnostic d'incident WebSphere MQ gérés, la conversion de données .NET et d'autres fonctions de niveau inférieur WebSphere MQ gérées.

Toutes les autres valeurs de NMQ_MQ_LIB entraînent le processus .NET à utiliser des tests de diagnostic et de conversion de données WebSphere MQ non gérés, ainsi que d'autres fonctions de niveau inférieur WebSphere MQ non gérées (en supposant qu'un client ou serveur WebSphere MQ MQI est installé sur le système).

WebSphere 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 niveau inférieur ne sont pas entièrement gérés (en supposant qu'un client ou serveur WebSphere MQ MQI 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 WebSphere MQ est installé, il se connecte en tant que serveur.
6. Si un client WebSphere MQ MQI est installé, il se connecte en tant que client non XA.
7. Sinon, il se connecte en tant que client géré.

Fichiers de configuration pour WebSphere MQ classes for .NET

Une application client .NET peut utiliser un fichier de configuration client WebSphere MQ MQI 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 WebSphere MQ classes for .NET peut utiliser un fichier de configuration client de la même manière que n'importe quel autre client WebSphere MQ MQI. Ce fichier est généralement appelé mqclient.ini, mais vous pouvez spécifier un nom de fichier différent. Pour plus d'informations sur le fichier de configuration client, voir Configuration d'un client à l'aide d'un fichier de configuration WebSphere MQ MQI client configuration file.

Seuls les attributs suivants d'un fichier de configuration client WebSphere MQ MQI sont pertinents pour WebSphere MQ classes for .NET. Si vous spécifiez d'autres attributs, cela n'a aucun effet.

stanza	**Attribut**
Canaux	CCSID
Canaux	Répertoire ChannelDefinition
Canaux	Fichier ChannelDefinition
Canaux	Paramètres ServerConnection
ClientExit-Chemin	ExitsDefaultPath

stanza	Attribut
ClientExit-Chemin	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
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 WebSphere 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 utilisés 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 .NET Framework, mais les noms de section, les clés et les valeurs mentionnés dans cette documentation sont spécifiques à Websphere MQ.

Le format du fichier de configuration de l'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 montre 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.

Pour plus d'informations, voir la documentation Microsoft .

Exemple de fragment de code

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 a WebSphere MQ message, writing some text in UTF format
      MQMessage hello_world = new MQMessage();
      hello_world.WriteUTF("Hello World!");

      // Specify the message options
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the defaults,
```

```
                                                              // same as MQPMO_DEFAULT

      // Put the message on the queue
      system_default_local_queue.Put(hello_world, pmo);



      // Get the message back again

      // First define a WebSphere MQ message buffer to receive the message
      MQMessage retrievedMessage =new MQMessage();
      retrievedMessage.MessageId =hello_world.MessageId;

      // Set the get message options
      MQGetMessageOptions gmo =new MQGetMessageOptions(); //accept the defaults
                                                          //same as MQGMO_DEFAULT

      // Get the message off the queue
      system_default_local_queue.Get(retrievedMessage,gmo);

      // Prove we have the message by displaying the UTF message text
      String msgText = retrievedMessage.ReadUTF();
      Console.WriteLine("The message is: {0}", msgText);

      // Close the queue
      system_default_local_queue.Close();

      // Disconnect from the queue manager
      qMgr.Disconnect();
    }

    //If an error has occurred,try to identify what went wrong.

    //Was it a WebSphere MQ error?
    catch (MQException ex)
    {
      Console.WriteLine("A WebSphere MQ error occurred: {0}", ex.ToString());
    }

    catch (System.Exception ex)
    {
      Console.WriteLine("A System error occurred: {0}", ex.ToString());
    }

    return 0;
  }//end of start
}//end of sample
```

Opérations sur les gestionnaires de files d'attente

Cette section explique comment se connecter et se déconnecter d'un gestionnaire de files d'attente à l'aide de WebSphere MQ classes for .NET.

Configuration de l'environnement WebSphere MQ

Avant d'utiliser la connexion client pour vous connecter à un gestionnaire de files d'attente, vous devez configurer l'environnement WebSphere MQ .

Remarque : Cette étape n'est pas nécessaire lors de l'utilisation de WebSphere MQ classes for .NET en mode liaisons de serveur.

L'interface de programmation .NET vous 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 MQEnvironment .NET class

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 WebSphere MQ sur le port 1414. Pour spécifier un port différent, utilisez le code suivant:

```
MQEnvironment.Port = nnnn;
```

Connexion à un gestionnaire de files d'attente

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();
```

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 WebSphere 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 ClasseMQTopic .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 procédurale, où MQPUT et MQGET placent et obtiennent des tableaux d'octets, les classes WebSphere MQ pour .NET placent et obtiennent 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* .

Pour plus d'informations, voir ClasseMQMessage .NET .

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 WebSphere MQ et JMS. Pour plus d'informations sur les propriétés de message dans WebSphere 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
- Int
- Int2
- Int4
- Int8
- Long
- Objet
- Court
- String

Par exemple, pour obtenir la propriété WebSphere MQ myproperty (chaîne de caractères), utilisez l'appel `message.GetStringProperty('myproperty')`. Vous pouvez éventuellement transmettre un descripteur de propriété, qui sera terminé par WebSphere MQ .

Traitement des erreurs

Traiter les erreurs provenant des classes WebSphere MQ pour .NET à l'aide des 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 WebSphere MQ sont différents de zéro. Cela simplifie la logique du programme de sorte que vous n'ayez pas à vérifier les codes retour après chaque appel à WebSphere 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 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* ouvert automatiquement 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 multiprocessus. WebSphere 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 WebSphere MQ .NET garantit que, pour une connexion particulière (instance d'objetMQQueueManager), tous les accès au gestionnaire de files d'attente cible WebSphere MQ sont synchronisés. Le comportement par défaut est qu'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 les classes .NET pour WebSphere MQ. 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 section CHANNELS, utilisez les attributs de répertoire ChannelDefinitionpour indiquer le répertoire dans lequel se trouve la table et le fichier ChannelDefinitionpour indiquer 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 le répertoire ChannelDefinitionpour spécifier le répertoire dans lequel se trouve la table et le fichier ChannelDefinitionpour indiquer 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 section CHANNELS, utilisez les attributs de répertoire ChannelDefinitionpour indiquer le répertoire dans lequel se trouve la table et le fichier ChannelDefinitionpour indiquer 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.

Comment une application .NET détermine la définition de canal à utiliser

Dans l'environnement client WebSphere 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 d'application .NET, nom de section CHANNELS, 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 WebSphere 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 WebSphere 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

V 7.5.0.6 A partir de la Version 7.5.0, Fix Pack 6, l'ID utilisateur et le mot de passe qui sont spécifiés avec l'application client .NET gérée sont définis dans la classe IBM WebSphere MQ .NET MQChannelDefinition qui est transmise à l'exit de sécurité du 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 423).

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 608 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 d'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 608 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 WebSphere 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 SSL (Secure Sockets Layer)

La section suivante ne s'applique pas au client géré.

WebSphere Les classes MQ pour les applications client .NET prennent en charge le chiffrement SSL (Secure Sockets Layer). SSL 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.

activation de SSL

SSL est pris en charge uniquement pour les connexions client. Pour activer SSL, 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 SSL, 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 SSL avec un CipherSpec de NULL_MD5:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "NULL_MD5";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

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 SSL aboutisse, le magasin de clés du client doit être configuré avec une chaîne de certificats racine d'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 SSL, qui contient un *nom distinctif* (DN).

Une application client WebSphere 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, le premier devant 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 Sécurité.

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 SSL

Les codes anomalie suivants peuvent être émis par les classes WebSphere MQ pour .NET lors de la connexion à un gestionnaire de files d'attente à l'aide de SSL:

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 SSL.

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.

Utilisation de .NET Monitor

Pour plus d'informations, voir Fonctions disponibles uniquement avec l'installation principale sous Windows .

.NET Monitor est une application similaire à un moniteur de déclenchement WebSphere MQ . 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. .NET Monitor 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 .NET Monitor, 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 .NET Monitor a déjà lancé une transaction.

Le composant reçoit des 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 à .NET Monitor.

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.

Lorsque le composant reçoit des 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 WebSphere MQ.

Exemples de fragments de code

Cette rubrique contient deux exemples de composants qui obtiennent un message de .NET Monitor 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 des programmes WebSphere 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 WebSphere MQ est installé.

Pour générer une application C# à l'aide de WebSphere 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 WebSphere 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 WebSphere 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.

Traçage des programmes WebSphere MQ .NET

Dans WebSphere MQ .NET, vous démarrez et contrôlez la fonction de trace comme dans les programmes WebSphere MQ à l'aide de l'interface MQI.

Toutefois, les paramètres -i et -p de la commande strmqtrc, qui vous permettent de spécifier des identificateurs de processus et d'unité d'exécution, ainsi que des processus nommés, n'ont aucun effet.

Vous devez normalement utiliser la fonction de trace uniquement à la demande du service IBM .

Pour plus d'informations sur les commandes de trace, voir Utilisation de la trace sous Windows .

IBM WebSphere MQ canal personnalisé pour Microsoft Windows Communication Foundation (WCF)

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

Concepts associés

«Introduction à l'utilisation du canal personnalisé WebSphere MQ pour WCF avec .NET 3», à la page 617
Présentation des informations disponibles pour les programmeurs utilisant le canal personnalisé WebSphere MQ pour Windows Communication Foundation (WCF) avec .NET 3.

«Utilisation des canaux personnalisés WebSphere MQ pour WCF», à la page 621
Présentation des informations disponibles pour les programmeurs utilisant les canaux personnalisés WebSphere MQ V7 pour Windows Communication Foundation (WCF).

«Utilisation des exemples WCF», à la page 639
Les exemples Windows Communication Foundation (WCF) fournissent des exemples simples de la manière dont le canal personnalisé WebSphere MQ peut être utilisé.

«Identification des incidents sur le canal personnalisé WCF pour WebSphere MQ», à la page 645
Vous pouvez utiliser la trace WebSphere MQ pour collecter des informations détaillées sur les différentes parties du code WebSphere MQ . Lorsque vous utilisez Windows Communication Foundation (WCF), une sortie de trace distincte est générée pour la trace de canal personnalisé WCF intégrée à la trace d'infrastructure WCF Microsoft .

Introduction à l'utilisation du canal personnalisé WebSphere MQ pour WCF avec .NET 3

Présentation des informations disponibles pour les programmeurs utilisant le canal personnalisé WebSphere MQ pour Windows Communication Foundation (WCF) avec .NET 3.

Qu'est-ce que le canal personnalisé WebSphere MQ pour WCF?

Le canal personnalisé pour WebSphere 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.

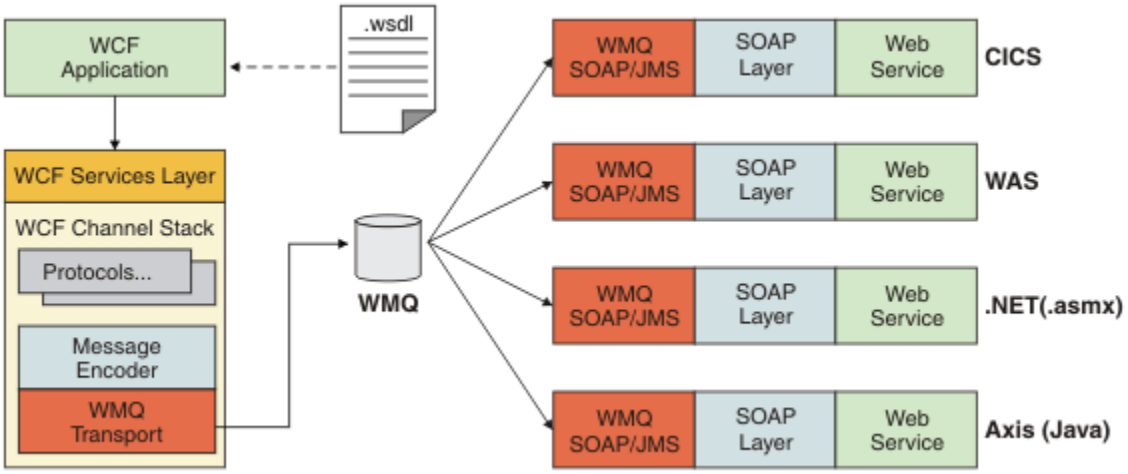
Le canal personnalisé pour WebSphere MQ est un canal de transport et, en tant que tel, il doit être associé à un encodeur de message et à des protocoles facultatifs requis par l'application à l'aide d'une liaison personnalisée WCF. Ainsi, les applications qui ont été développées pour utiliser WCF peuvent utiliser le canal personnalisé pour WebSphere MQ pour envoyer et 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 de WebSphere MQ. Pour la liste complète des fonctions prises en charge, voir «Fonctions et capacités des canaux personnalisés WCF», à la page 621.

Quand et pourquoi utiliser le canal personnalisé WebSphere MQ pour WCF?

Le canal personnalisé WebSphere MQ peut être utilisé 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 WebSphere MQ dans le modèle de programmation unifié WCF.

Un scénario de modèle d'utilisation typique du canal personnalisé WebSphere MQ pour WCF est une interface vers des services Web hébergés sur WebSphere MQ (SOAP/JMS)

Les messages sont transmis à l'aide du format de message SOAP sur JMS de WebSphere MQ, ce qui permet aux clients et services WCF d'appeler ou d'être appelés par d'autres applications ou environnements d'hébergement WebSphere MQ compatibles avec ce format, y compris les services Web et les clients exécutés dans WebSphere Application Server, CICS, Axis v1 (Java), et .asmx (.NET), comme illustré dans le diagramme suivant:



Pour plus de détails sur SOAP sur JMS, voir: «Transport WebSphere MQ pour SOAP», à la page 978

Voici un exemple de scénario typique du diagramme :

1. Un service Web hébergé dans WebSphere Application Server et exposé sur WebSphere MQ à l'aide de la prise en charge de SOAP sur JMS dans WebSphere Application Server

2. Le document WSDL décrivant le service peut ensuite être utilisé par l'outil WCF pour générer un proxy client et une configuration qui créent ensuite une pile de canaux WCF appropriée, y compris le canal personnalisé.
3. L'application client peut ensuite utiliser le proxy pour démarrer le service Web de la même manière que n'importe quel autre service Web.

Le canal est généralement utilisé avec un encodeur de message WCF text/SOAP, mais le canal peut être apparié avec d'autres encodeurs de message WCF si nécessaire. L'utilisation d'autres encodeurs peut également fournir une intégration limitée avec des applications WebSphere MQ natives qui ne prennent pas en charge SOAP sur JMS, mais ce n'est pas le rôle principal du canal.

Les principaux avantages de l'utilisation du canal personnalisé dans un environnement WCF sont les suivants:

- Appel asynchrone: prise en charge des opérations client de déclenchement et d'oubli où le client est découplé de la disponibilité du service et des fonctions, telles que le réacheminement des réponses et le multi-accès.
- Caractéristiques de mise à l'échelle fiable: La messagerie basée sur la file d'attente permet d'ajouter de la capacité de manière prévisible à un système.
- Qualité de service: les messages sont tangibles et traçables, et peuvent être facilement gérés et administrés.

Configuration logicielle requise et instructions d'installation pour le canal personnalisé WebSphere MQ pour WCF

Cette rubrique décrit la configuration logicielle requise et les informations d'installation pour le canal personnalisé WebSphere MQ pour WCF.

Le canal personnalisé WebSphere MQ pour WCF peut uniquement se connecter aux gestionnaires de files d'attente WebSphere MQ V7 ou version ultérieure.

Configuration logicielle requise pour le canal personnalisé WCF pour WebSphere MQ

Ces informations répertorient la configuration logicielle requise pour le canal personnalisé WCF pour WebSphere MQ.

Environnement d'exécution

- Microsoft .NET Framework v3.0 ou version ultérieure doit être installé sur la machine hôte.
- *Java et .NET Messaging and Web Services* est installé par défaut dans le cadre du programme d'installation WebSphere MQ 7.0.1 . Installe les assemblages .NET nécessaires pour le canal personnalisé dans le cache d'assemblage global.

Remarque : Si Microsoft .NET Framework v2.0 ou version ultérieure n'est pas installé avant d'installer WebSphere MQ V7.0.1, l'installation du produit WebSphere MQ se poursuit sans erreur, mais le canal personnalisé WebSphere MQ n'est pas disponible. Si .NET Framework est installé après l'installation de WebSphere MQ 7.0.1, le canal personnalisé WebSphere MQ doit être activé en exécutant le script *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , où *WMQInstallDir* correspond au répertoire dans lequel WebSphere MQ 7.0.1 est installé. Ce script installe les assemblages requis dans le cache d'assemblage global (GAC). Un ensemble de fichiers `amqi*.log` enregistrant 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 v3.0 ou une version ultérieure à partir d'une version antérieure, par exemple à partir de .NET v2.0.

Environnement de développement

- Microsoft Visual Studio 2008 ou Windows Software Development Kit for .NET 3.0 ou version ultérieure.

- Microsoft .NET Framework V3.5 ou version ultérieure doit être installé sur la machine hôte afin de générer les exemples de fichiers de solution.

Remarque : Si Microsoft .NET Framework v2.0 ou version ultérieure n'est pas installé avant d'installer WebSphere MQ V7.0.1, l'installation du produit WebSphere MQ se poursuit sans erreur, mais le canal personnalisé WebSphere MQ n'est pas disponible. Si .NET Framework est installé après l'installation de WebSphere MQ 7.0.1, le canal personnalisé WebSphere MQ doit être activé en exécutant le script *WMQInstallDir*\bin\amqiRegisterdotNet.cmd , où *WMQInstallDir* correspond au répertoire dans lequel WebSphere MQ 7.0.1 est installé. Ce script installe les assemblages requis dans le cache d'assemblage global (GAC). Un ensemble de fichiers amqi*.log enregistrant 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 v3.0 ou une version ultérieure à partir d'une version antérieure, par exemple à partir de .NET v2.0.

WebSphere MQ Custom Channel for WCF: Qu'est-ce qui est installé?

Le canal personnalisé pour WebSphere 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 de WebSphere MQ 7.0.1 .

Canal personnalisé WebSphere MQ pour WCF

Le canal personnalisé WebSphere MQ pour WCF est installé par défaut dans le cadre de l'installation de WebSphere MQ 7.0.1 ; 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 vers WebSphere MQ 7.0.1 à partir d'une version antérieure, la mise à jour installe le canal personnalisé WebSphere 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 Java and .NET Messaging and Web Services contient le fichier IBM.XMS.WCF.dll et le fichier IBM.XMS.WCF.dll est l'assemblage de canal personnalisé principal, qui contient les classes d'interface WCF. Ce fichier est installé dans le cache d'assemblage global (GAC) et est également disponible dans le répertoire suivant: *MQ_INSTALLATION_PATH*\bin où *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ 7.0.1 est installé.

Les classes de clé requises pour l'utilisation du canal personnalisé se trouvent dans l'espace de nom *: IBM.XMS.WCF* et:

Nom de la liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement
Importateur de liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter
Configuration de liaison de transport	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig

Exemples de canal personnalisé WebSphere MQ

Les exemples fournissent des exemples simples de la façon dont le canal personnalisé WebSphere MQ pour WCF peut être utilisé. Les exemples et les fichiers associés se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\tools\wcf\samples\ , où *MQ_INSTALLATION_PATH* est le répertoire d'installation de WebSphere MQ. Pour plus d'informations sur les exemples de canal personnalisé WebSphere MQ , voir: «Utilisation des exemples WCF», à la page 639

svcutil.exe.config

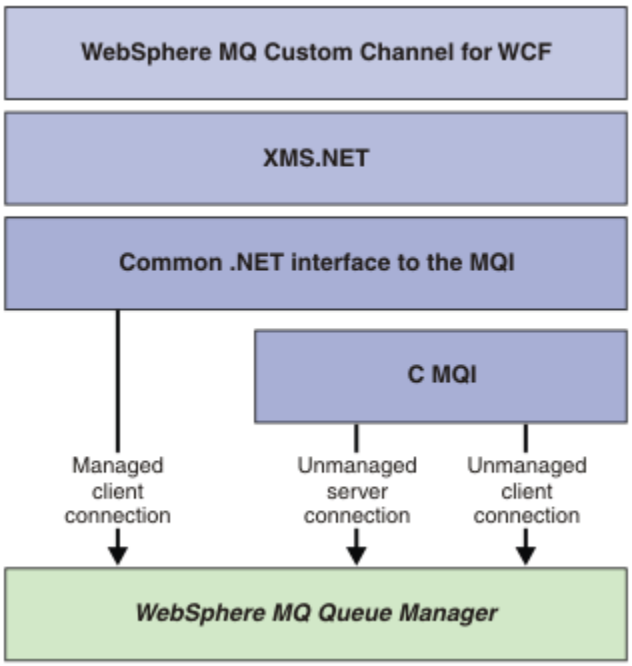
Le svcutil.exe.config est un exemple des paramètres de configuration requis pour permettre à l'outil de génération de proxy client Microsoft WCF svcutil de reconnaître le canal personnalisé. Le fichier svcutil.exe.config se trouve dans le répertoire *MQ_INSTALLATION_PATH*\tools\wcf\docs\examples\ , où *MQ_INSTALLATION_PATH* est le

répertoire d'installation de WebSphere 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 636.

Architecture WCF

Le canal personnalisé WebSphere MQ pour WCF est intégré par-dessus l'API `IBM Message Service Client for .NET (XMS .NET)`.

L'architecture WCF est illustrée dans le diagramme suivant:



Tous les composants requis sont installés par défaut avec l'installation WebSphere MQ V7.0.1 .

Les trois connexions sont les suivantes: Connexions client gérées, Connexions serveur non gérées et Connexions client non gérées. Pour plus d'informations sur ces connexions, voir «Options de connexion WCF», à la page 626.

Utilisation des canaux personnalisés WebSphere MQ pour WCF

Présentation des informations disponibles pour les programmeurs utilisant les canaux personnalisés WebSphere MQ V7 pour Windows Communication Foundation (WCF).

Microsoft Windows Communication Foundation soutient la prise en charge des services Web et de la messagerie dans Microsoft .NET Framework 3. WebSphere MQ V7 peut désormais être utilisé en tant que 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 conformément à l'implémentation SOAP sur JMS de WebSphere MQ V7. Les applications peuvent ensuite communiquer avec les services hébergés par WCF ou par l'infrastructure de service WebSphere SOAP sur JMS. Pour plus de détails sur SOAP sur JMS, voir: «Transport WebSphere MQ pour SOAP», à la page 978

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 WebSphere MQ peut utiliser dans les canaux personnalisés Microsoft Windows Communication Foundation (WCF).

Le canal personnalisé WebSphere 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 WebSphere 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 WebSphere MQ pour WCF est utilisé, par exemple, pour envoyer des messages dans deux directions de manière asynchrone. La même instance de 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

connectionFactory
Le paramètre connectionFactory est obligatoire. Pour la syntaxe de ce paramètre, voir Syntaxe d'URI et paramètres pour le déploiement de service Web .

initialContextFactory
Le paramètre initialContextFactory est obligatoire et doit être défini sur "com.ibm.mq.jms.Nojndi" pour la compatibilité avec WebSphere Application Server et d'autres produits (voir «Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP», à la page 1037).

Distribution garantie de canal personnalisé WCF

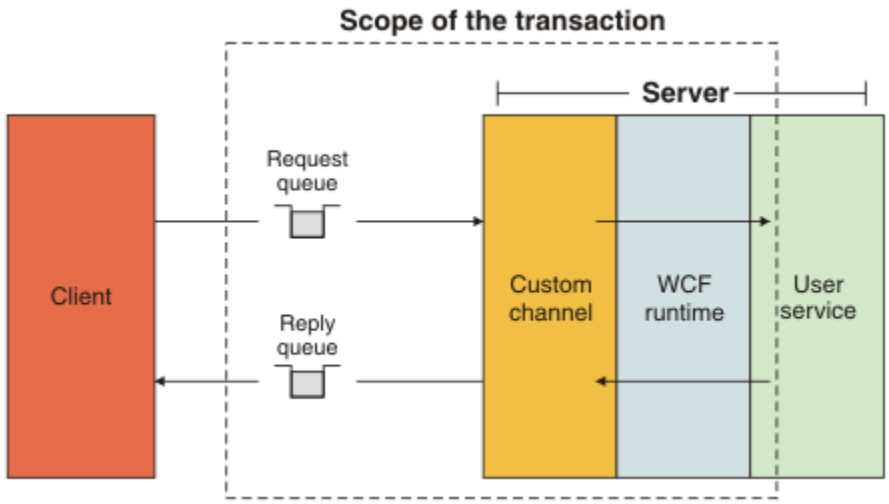
La livraison assurée garantit qu'une demande de service ou une réponse est activée et qu'elle n'est pas perdue.

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. Pour plus d'informations sur la définition des propriétés d'URI, voir: Syntaxe d'URI et paramètres pour le déploiement de service Web.

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 WebSphere 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 624

- 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 632
- 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 WebSphere MQ pour WCF prend en charge l'utilisation de SSL uniquement pour les connexions client non gérées au gestionnaire de files d'attente.

SSL peut être spécifié de l'une des deux manières suivantes:

- Indiquez SSL directement sur l'URI SOAP sur JMS. Pour une description complète des options SSL, voir SSL et WebSphere MQ transport for SOAP
- Indiquez SSL 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 WebSphere 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.

Vous ne pouvez pas spécifier la table de définition de canal directement dans l'URI SOAP sur JMS. 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 .

Concepts associés

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 623

- 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:

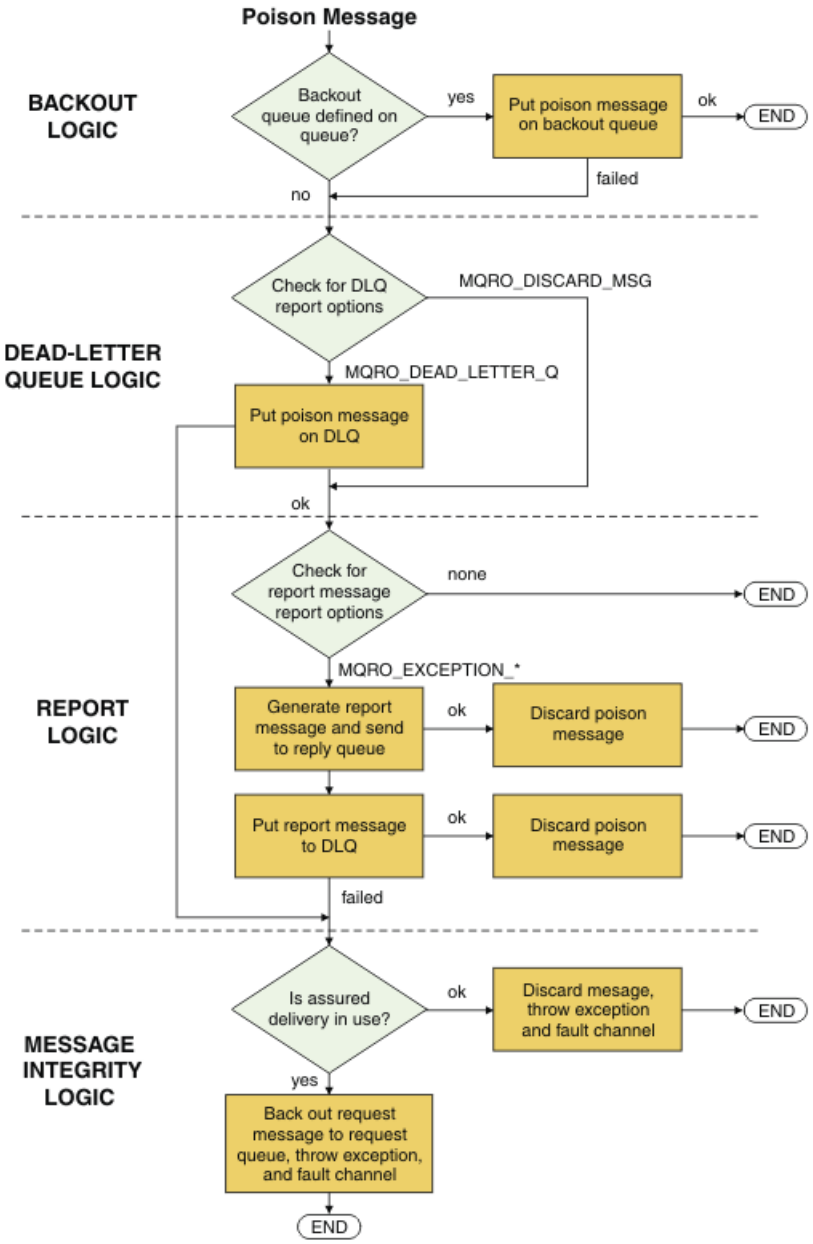
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

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 d'un message incohérent:



Options de connexion WCF

Il existe trois modes de connexion d'un canal personnalisé WebSphere 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 597

Pour plus d'informations sur l'architecture WCF, voir «Architecture WCF», à la page 621

Connexion client non gérée

Une connexion établie dans ce mode se connecte en tant que client WebSphere MQ à un serveur WebSphere MQ s'exécutant sur la machine locale ou sur une machine distante.

Pour utiliser le canal personnalisé WebSphere MQ pour WCF en tant que client WebSphere MQ , vous pouvez l'installer avec le client WebSphere MQ MQI, soit sur le serveur WebSphere MQ , soit sur une machine distincte.

Connexion à un serveur non géré

Lorsqu'il est utilisé en mode liaisons de serveur, le canal personnalisé WebSphere 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 WebSphere MQ que l'utilisation de connexions réseau.

Pour utiliser la connexion de liaisons, vous devez installer le canal personnalisé WebSphere MQ pour WCF sur le serveur WebSphere MQ .

Connexion client gérée

Une connexion établie dans ce mode se connecte en tant que client WebSphere MQ à un serveur WebSphere MQ s'exécutant sur la machine locale ou sur une machine distante.

Les classes de canal personnalisé WebSphere 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 597.

Création et configuration du canal personnalisé WebSphere MQ pour WCF

Les canaux personnalisés WebSphere MQ V7 pour WCF fonctionnent de la même manière que les canaux WCF de transport proposés par Microsoft. Le canal personnalisé WebSphere MQ pour WCF peut être créé de deux manières.

Pourquoi et quand exécuter cette tâche

Le canal personnalisé WebSphere MQ s'intègre à WCF en tant que canal de transport WCF et doit donc 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 WebSphere 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 WebSphere 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 630
- 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'outil d'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 73. Nouvelles zones d'élément de liaison

Zone	Valeur
Nom	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
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 74. 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 628
2. Apportez les modifications requises aux valeurs par défaut des propriétés, comme décrit dans: «Options de configuration de liaison WCF», à la page 632

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 628 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 75. 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: «WebSphere MQ Canal personnalisé pour le format d'adresse d'URI de noeud final WCF», à la page 631
Liaison en cours	`customBinding`
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 76. 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: «WebSphere MQ Canal personnalisé pour le format d'adresse d'URI de noeud final WCF», à la page 631
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 WebSphere 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 WebSphere 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 639

Ces définitions peuvent être créées de deux manières:

- Administrativement, en fournissant les détails dans un fichier de configuration d'application, comme décrit ici: «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 627
- A l'aide d'un programme directement à partir du code de l'application, comme décrit dans l'exemple suivant.

Etape 1: Création d'une instance de l'élément de liaison de transport du canal

Ajoutez le code suivant à votre application:

```
SoapJmsIbmTransportBindingElement transportBindingElement = new
SoapJmsIbmTransportBindingElement();
```

Etape 2: Définition des propriétés de liaison

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;
```

Etape 3: Créez une liaison personnalisée qui associe le canal de transport à un encodeur de message

Créez une liaison personnalisée en ajoutant le code suivant à votre application:

```
Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
transportBindingElement);
```

Etape 4: Création de l'URI SOAP/JMS

L'URI SOAP/JMS qui décrit les détails de connexion WebSphere MQ requis pour accéder au service doit être fourni en tant qu'adresse de noeud final. Cela 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 l'adresse de noeud final, voir «WebSphere MQ Canal personnalisé pour le format d'adresse d'URI de noeud final WCF», à la page 631

WebSphere MQ Canal personnalisé pour le format d'adresse d'URI de noeud final WCF

Un identificateur URI (Universal Resource Identifier) fournit l'emplacement et les détails de connexion pour spécifier un service Web. Ce format d'URI permet un contrôle complet des paramètres et des options spécifiques à SOAP/ WebSphere MQlors de l'accès aux services cible.

Un service Web est spécifié à l'aide d'un identificateur URI (Universal Resource Identifier). Cette section indique le format d'URI pris en charge dans WebSphere MQ transport for SOAP. Ce format d'URI permet un contrôle complet des paramètres et des options spécifiques à SOAP/WebSphere MQlors de l'accès aux services cible. Ce format est compatible avec WebSphere Application Server (WAS) et avec CICS facilitant l'intégration de WebSphere 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 *valeur* est une valeur appropriée, et l'élément name=*valeur* peut être répété autant de fois que nécessaire, la deuxième occurrence et les occurrences suivantes étant précédées d'une perluète (&).

Pour plus d'informations sur la définition des propriétés d'URI, voir: Syntaxe d'URI et paramètres pour le déploiement de service Web

Les noms de paramètre sont sensibles à la casse, tout comme les noms des objets WebSphere 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 des paramètres supplémentaires 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 d'axe, `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
```

Options de configuration de liaison WCF

Cette rubrique décrit comment les options de configuration peuvent être appliquées aux informations de liaison des canaux personnalisés et répertorie les options disponibles.

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 également être spécifiés dans le fichier de configuration de l'application, par exemple: `app.config`. Le fichier de configuration est généré par **svcutil**, par exemple:

```
<customBinding>

...

     <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>

...

</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

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.

Nom de la propriété	Application client ou de service	Valeur d'administration	Valeur programmatique	Description
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.
Appels MaxConcurrent	Client	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.

Nom de la propriété	Application client ou de service	Valeur d'administration	Valeur programmatique	Description
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 623). 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 WebSphere MQ pour WCF et les services WCF qui l'utilisent peuvent être hébergés par les méthodes suivantes:

- Auto-hébergement
- Windows Service

Le canal personnalisé IBM WebSphere MQ pour WCF ne peut pas être hébergé dans le service Windows Process Activation.

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 dernières informations, 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 627

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 WebSphere MQ 7 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 630

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 WebSphere 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 les exposer 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'utilitaire de métadonnées Microsoft ServiceModel (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 WebSphere 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 WebSphere 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 WebSphere MQ , mais également exposer 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 WebSphere 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 637

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 WebSphere 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
```

Où *MQ_INSTALLATION_PATH* est le répertoire d'installation de WebSphere MQ.

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 627

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 WebSphere MQ 7 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 630

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* sont la liaison et l'adresse de noeud final configurées à l'étape «1», à la page 638 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 manière dont le canal personnalisé WebSphere MQ peut être utilisé.

Pour générer les exemples de projets, le SDK Microsoft .NET 3.5 ou Microsoft Visual Studio 2008 est requis.

Exemple WCF client et serveur unidirectionnel simple

Cet exemple illustre le canal personnalisé WebSphere 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 636

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 WebSphere MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS, voir *WebSphere MQ Transport for SOAP* dans la documentation du produit WebSphere 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 WebSphere MQ.

4. Exécutez le client une 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 WebSphere 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 WebSphere 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 636

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 WebSphere MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS, voir *WebSphere MQ Transport for SOAP* dans la documentation du produit WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ et qu'il soit accessible à partir d'un gestionnaire de files d'attente local. Pour plus d'informations sur l'installation et la configuration de l'environnement, voir: «Installation de WebSphere MQ Web transport for SOAP», à la page 988

Lorsque l'environnement d'hébergement de service .NET SOAP sur JMS est correctement installé et configuré dans WebSphere 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 WebSphere MQ , par exemple: `C:\Program Files\IBM\WebSphere MQ`
2. Vérifiez que le compilateur Java `javac` est disponible dans la variable PATH.
3. Copiez le fichier `axis.jar` du répertoire `prereqs/axis` du CD d'installation WebSphere vers le répertoire de production WebSphere MQ , par exemple: `C:\Program Files\IBM\WebSphere MQ\java\lib\soap`
4. Ajoutez à la variable PATH: *MQ_INSTALLATION_PATH*`\Java\lib` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere MQ est installé, par exemple: `C:\Program Files\IBM\WebSphere 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 WebSphere MQ est installé, par exemple: `C:\Program Files\IBM\WebSphere 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 WebSphere MQ est installé.
   - Pour AXIS:*MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere 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 d'un client WCF à l'exemple de service .NET SOAP sur JMS fourni dans WebSphere 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 637

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 WebSphere MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS, voir *WebSphere MQ Transport for SOAP* dans la documentation du produit WebSphere MQ .

Procédure

Exécutez le client une 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 WebSphere 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 WebSphere 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 WebSphere MQ et qu'il soit accessible à partir d'un gestionnaire de files d'attente local. Pour plus d'informations sur l'installation et la configuration de l'environnement, voir: «Installation de WebSphere MQ Web transport for SOAP», à la page 988

Lorsque l'environnement d'hébergement de service .NET SOAP sur JMS est correctement installé et configuré dans WebSphere 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 WebSphere MQ , par exemple: `C:\Program Files\IBM\WebSphere MQ`
2. Vérifiez que le compilateur Java `javac` est disponible dans la variable PATH.
3. Copiez le fichier `axis.jar` du répertoire `prereqs/axis` du CD d'installation WebSphere vers le répertoire d'installation WebSphere MQ .
4. Ajoutez à la variable PATH: *MQ_INSTALLATION_PATH*`\Java\lib` où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere MQ est installé, par exemple: `C:\Program Files\IBM\WebSphere 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 WebSphere MQ est installé, par exemple: `C:\Program`

Files\IBM\WebSphere 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 WebSphere MQ est installé.
   - Pour AXIS:*MQ_INSTALLATION_PATH*\Tools\soap\samples\runivt Dotnet2AxisClient hold où *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere 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 entre un client WCF et l'exemple de service Axis Java SOAP sur JMS fourni dans WebSphere 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 637

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 WebSphere MQ.

Procédure

Exécutez le client une 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 WebSphere 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.

Référence associée

«Traitement des différents noms d'élément de réponse SOAP», à la page 652
WCF s'attend à ce que le nom d'une valeur renvoyée soit dans un format spécifique par défaut, mais un service risque de ne pas renvoyer un élément avec son nom dans le format attendu.

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 (WAS) 6. Un service de demande-réponse est également fourni.

Avant de commencer

Cet exemple requiert l'utilisation de la configuration WebSphere MQ suivante:

Tableau 77. WebSphere 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 nécessite également qu'un environnement d'hébergement WebSphere Application Server V6 soit correctement installé et configuré. WebSphere Application Server V6 utilise une connexion en mode liaisons pour se connecter à WebSphere MQ par défaut. Par conséquent, WebSphere Application Server V6 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 de 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 WebSphere 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 WebSphere MQ.

Pourquoi et quand exécuter cette tâche

L'exemple illustre une connexion d'un client WCF à l'exemple de service WebSphere Application Server SOAP sur JMS fourni dans les exemples WCF inclus dans WebSphere MQ V7, 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 WebSphere 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 636

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 *MQ_INSTALLATION_PATH*\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear , où *MQ_INSTALLATION_PATH* correspond au répertoire d'installation de WebSphere MQ. Pour plus d'informations sur le formatage de l'URI de noeud final JMS, voir Syntaxe d'URI et paramètres pour le déploiement de service Web .

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 et WebSphere Studio*. Pour en savoir plus sur le développement de services Web SOAP sur JMS compatibles avec le canal personnalisé WCF WebSphere MQ , consultez l'article correspondant à l'adresse suivante: 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 WebSphere 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.

Identification des incidents sur le canal personnalisé WCF pour WebSphere MQ

Vous pouvez utiliser la trace WebSphere MQ pour collecter des informations détaillées sur les différentes parties du code WebSphere MQ . Lorsque vous utilisez Windows Communication Foundation (WCF), une sortie de trace distincte est générée pour la trace de canal personnalisé WCF intégrée à la trace d'infrastructure WCF Microsoft .

L'activation complète de la trace pour le canal personnalisé WCF génère deux fichiers de sortie:

1. Trace de canal personnalisé WCF intégrée à la trace d'infrastructure WCF Microsoft .
2. Trace de canal personnalisé WCF intégrée à XMS .NET.

En disposant de deux sorties de trace, les problèmes peuvent être suivis à chaque interface à l'aide des outils appropriés, par exemple:

- Détermination des problèmes WCF à l'aide des outils Microsoft appropriés.
- WebSphere MQ Problèmes du client MQI avec le format de trace XMS .

Pour simplifier l'activation de la trace, la pile de trace .NET 3 TraceSource et XMS .NET sont toutes deux contrôlées à l'aide d'une interface unique, comme décrit dans: «Noms de fichier de trace et de configuration de trace WCF», à la page 646.

Hiérarchie d'exceptions de canal personnalisé WCF

Les types d'exception émis par le canal personnalisé sont cohérents avec WCF et sont généralement une exception TimeoutException ou CommunicationException (ou une sous-classe de CommunicationException).

Des informations supplémentaires sur la condition d'erreur, lorsqu'elles sont disponibles, sont fournies à l'aide d'exceptions liées ou internes. Les exceptions suivantes sont des exemples typiques, et chaque couche de l'architecture du canal ajoute une exception liée supplémentaire, par exemple: CommunicationsException a une exception liée XMSException, qui a une exception liée MQException:

1. System.serviceModel.CommunicationsExceptions
2. IBM.XMS.XMSException
3. IBM.WMQ.MQException

Les informations clés sont capturées et fournies dans la collecte de données de l'exception CommunicationException la plus élevée dans la hiérarchie. Cette capture et cette mise à disposition de données évitent aux applications d'avoir à se lier à chaque couche de l'architecture du canal afin d'interroger les exceptions liées, ainsi que toute information supplémentaire qu'elles pourraient contenir. Les noms de clé suivants sont définis:

- IBM.XMS.WCF.ErrorCode: Code du message d'erreur de l'exception de canal personnalisé en cours.
- IBM.XMS.ErrorCode: Message d'erreur de la première exception XMS dans la pile.
- IBM.WMQ.ReasonCode: Code anomalie WebSphere MQ sous-jacent.
- IBM.WMQ.CompletionCode: code achèvement WebSphere MQ sous-jacent.

Noms de fichier de trace et de configuration de trace WCF

Lorsque la trace est entièrement activée, elle génère deux fichiers de sortie, un pour le diagnostic des problèmes WCF et un fichier détaillé pour le matériel de diagnostic de trace interne. Pour simplifier l'activation de la trace, les piles de trace .NET 3 TraceSource et XMS .NET utilisent une seule interface.

Deux méthodes de trace différentes sont disponibles pour le canal personnalisé WCF, les deux méthodes de trace sont activées indépendamment ou ensemble. Chaque méthode produit son propre fichier de trace, de sorte que lorsque les deux méthodes de trace ont été activées, deux fichiers de sortie de trace sont générés.

Pour que la configuration et l'activation restent aussi simples que possible, la même interface est utilisée pour contrôler les deux méthodes de trace. Le fichier `app.config` doit être édité pour inclure la configuration de trace appropriée, comme décrit dans la section suivante. Les utilisateurs peuvent ensuite ajouter leurs propres sections équivalentes pour combiner la sortie avec la trace de leur propre application.

Le traçage de canal personnalisé WCF n'est pas activé par défaut. Vous devez d'abord créer un programme d'écoute de trace, puis définir le niveau de trace requis pour la source de trace sélectionnée dans le fichier `app.config` .

Configuration du canal personnalisé WCF avec la trace de l'infrastructure WCF

Ajoutez la section de code suivante à la section `<system.diagnostics><sources>` dans le fichier `app.config` :

```
<source name="IBM.XMS.WCF" switchValue="Verbose,ActivityTracing">
    <listeners>
```

```
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

L'élément de code précédent crée la trace de canal à l'aide de TraceSource.NET 3. Tous les appels des fichiers de configuration associés aux fichiers exécutables sont contrôlés par cet élément de code.

Configuration du canal personnalisé WCF avec la trace XMS .NET

La configuration de la trace XMS .NET requiert l'ajout d'une section de code à la section `<system.diagnostics><sources>` dans le fichier `app.config` . Toutefois, l'élément de code est ajouté à l'élément extensible `<source>` présenté dans la section Configuration du canal personnalisé WCF avec la trace de l'infrastructure WCF . Par conséquent, bien que le code de trace de l'infrastructure WCF doive être présent pour que la trace XMS .NET fonctionne, la trace de l'infrastructure WCF peut être désactivée si elle n'est pas requise, comme décrit dans la section Activation de la trace WCF .

```
<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing"
 xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path"
 xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced">
    <listeners>
        <remove name="Default"/>
        <add name="NewListener"/>
    </listeners>
</source>
```

Variables de configuration de trace WCF

Tableau 78. Variables de configuration de trace WCF

Variable	Description
nom	Indiquez le nom suivant: `IBM.XMS.WCF`
switchValue	switchValue contrôle le niveau de trace. Lorsque switchValue est défini sur `Off`, l'infrastructure WCF TraceSource n'est pas générée. Toute autre valeur, telle que `Verbose`, génère TraceSource. Pour obtenir des informations détaillées sur le niveau de trace de Microsoft, consultez votre documentation WCF ou accédez à la page Web Microsoft WCF Tracing: https://msdn.microsoft.com/en-us/library/ms733025(vs.85).aspx

Tableau 78. Variables de configuration de trace WCF (suite)

Variable	**Description**

xmsTraceSpecification =*ComponentName=type=état*	*ComponentName* est le nom de la classe que vous souhaitez tracer. Vous pouvez utiliser un caractère générique * dans ce nom. Exemple :
	`*=all=enabled`
	indique que vous souhaitez tracer toutes les classes, et
	`IBM.XMS.impl.*=all=enabled`
	indique que vous n'avez besoin que d'une trace d'API.
	type peut être l'un des types de trace suivants:
	• Tous
	• débogage
	• événement
	• EntryExit
	state peut être activé ou désactivé.

xmsTraceFilePath= "*nom_fichier*"	Si vous ne spécifiez pas de xmsTraceFilePath, ou si leFilePath xmsTraceest présent mais qu'il contient une chaîne vide, le fichier de trace est placé dans le répertoire en cours. Pour stocker le fichier de trace dans un répertoire nommé, indiquez le nom du répertoire dans leFilePath xmsTrace, par exemple: `xmsTraceFilePath="c:\somepath"`
xmsTraceFileSize= "*taille*"	Taille maximale allouée au fichier de trace. Lorsqu'un fichier atteint cette taille, il est archivé et renommé. La valeur maximale par défaut est de 20 Ko, qui est spécifiée comme suit: `xmsTraceFileSize="20000000".`
xmsTraceFileNumber= "*numéro*"	Nombre de fichiers de trace à conserver. La valeur par défaut est 4 (un fichier actif et trois fichiers archive). Le nombre minimal autorisé est deux.

xmsTraceFormat="*format*"	Il existe deux niveaux de format xmsTrace: `basic` et `advanced`. Le format de trace par défaut est de base si vous ne spécifiez pas de format xmsTraceou si le format xmsTraceest présent mais qu'il contient une chaîne vide. Les fichiers de trace sont générés dans ce format si vous spécifiez:
	`xmsTraceFormat="basic"`
	Si vous avez besoin d'une trace compatible avec les outils d'analyse de trace, vous devez spécifier:
	`traceFormat="advanced"`

Activation de la trace WCF

Il existe quatre combinaisons pour activer et désactiver les deux méthodes de trace différentes. Les quatre combinaisons nécessitent d'éditer les valeurs des sections de code décrites dans les sections précédentes.

Il existe également une variable d'environnement qui peut être définie ; pour plus d'informations, voir «Activation de la trace WCF avec la variable d'environnement WCF_TRACE_ON», à la page 650.

Ce tableau et les valeurs affichées dépendent des éléments de code précédemment indiqués qui ont déjà été ajoutés au fichier `app.config` .

Tableau 79. Combinaisons d'activation de trace WCF.

Type de trace	Valeur modifiée	Exemple
Trace XMS activée. WCF TraceSource activé	`switchValue` n'est pas défini sur `Off`	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Trace XMS activée. WCF TraceSource désactivé	Le `switchValue` est défini sur `Off` et un `xmsTraceSpecification` a été indiqué	`<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=enabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`
Trace XMS désactivée. WCF TraceSource activé	Il existe deux façons d'atteindre ce résultat: • La variable `switchValue` n'est pas définie sur `Off` et un`xmsTraceSpecification` n'a pas été ajouté • La variable `switchValue` n'est pas définie sur `Off` et le `xmsTraceSpecification` a été défini sur `disabled`	`<source name="IBM.XMS.WCF" switchValue="Verbose, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>`

Tableau 79. Combinaisons d'activation de trace WCF. (suite)

Type de trace	Valeur modifiée	Exemple
Trace XMS désactivée. WCF TraceSource désactivé	Il existe trois façons d'atteindre ce résultat: • Aucun élément <source> dans le fichier app.config • La variable switchValue est définie sur Off et unxmsTraceSpecification n'a pas été ajouté • La variable switchValue est définie sur Off et la variable xmsTraceSpecification a été définie sur disabled	<source name="IBM.XMS.WCF" switchValue="Off, ActivityTracing" xmsTraceSpecification="*=all=disabled" xmsTraceFilePath="path" xmsTraceFileSize="2000000" xmsTraceFileNumber="4" xmsTraceFormat="advanced"> <listeners> <remove name="Default"/> <add name="NewListener"/> </listeners> </source>

Activation de la trace WCF avec la variable d'environnement WCF_TRACE_ON

Outre les méthodes décrites précédemment pour activer la trace WCF, la trace XMS .NET peut également être activée à l'aide de la variable d'environnement WCF_TRACE_ON.

La définition de la variable d'environnement WCF_TRACE_ON sur une valeur non nulle équivaut à la définition de xmstraceSpecification sur *=all=enabled, par exemple: "set WCF_TRACE_ON=true"

Toutefois, si xmstraceSpecification est explicitement défini dans le fichier app.config , la variable d'environnement WCF_TRACE_ON est remplacée.

Fichiers de sortie de trace WCF et noms de fichier

Les fichiers de trace XMS sont généralement nommés à l'aide du nom de base et du format d'ID de processus xms_trace_*pid*.log, où *pid* est l'ID de processus.

Comme les fichiers de trace XMS peuvent toujours être générés en parallèle avec les fichiers de trace de canal personnalisé WCF, les fichiers de trace de canal personnalisé WCF intégrés aux fichiers de sortie de trace XMS .NET ont le format suivant pour éviter toute confusion: wcfxms_trace_*pid*.log, où *pid* est l'ID de processus.

Le fichier de sortie de trace est créé dans le répertoire de travail en cours par défaut, mais cette destination peut être redéfinie si nécessaire.

WCF XMS First Failure Support Technology (FFST)

Vous pouvez collecter des informations détaillées sur les différentes parties du code WebSphere MQ à l'aide de la trace WebSphere MQ . XMS FFST dispose de ses propres fichiers de configuration et de sortie pour le canal personnalisé WCF.

Les fichiers de trace XMS FFST sont traditionnellement nommés à l'aide du nom de base et du format d'ID de processus xmsffdc*pid_date*.txt, où *pid* correspond à l'ID de processus et *date* à l'heure et à la date.

Comme les fichiers de trace XMS FFST peuvent toujours être produits en parallèle avec les fichiers XMS FFST du canal personnalisé WCF, les fichiers de sortie XMS FFST du canal personnalisé WCF ont le format suivant pour éviter toute confusion: wcfffdc*pid_date*.txt, où *pid* correspond à l'ID de processus et *date* à l'heure et à la date.

Ce fichier de sortie de trace est créé dans le répertoire de travail en cours par défaut, mais cette destination peut être redéfinie si nécessaire.

Le canal personnalisé WCF avec l'en-tête de trace XMS .NET est similaire à l'exemple suivant:

```
************* Start Display XMS WCF Environment *************
  Product Name  :- value
  WCF Version   :- value
  Level    :- value
*************  End Display XMS WCF Environment  *************
```

Les fichiers de trace FFST sont formatés de manière standard, sans formatage spécifique au canal personnalisé.

Informations de version WCF

Les informations de version WCF facilitent l'identification des problèmes et sont incluses dans les métadonnées d'assemblage du canal personnalisé.

Le canal personnalisé WebSphere MQ pour les métadonnées de version WCF peut être extrait de l'une des trois manières suivantes:

- Utilisation de l'utilitaire WebSphere MQ dspmqver. Pour plus d'informations sur l'utilisation de dspmqver, voir: dspmqver
- A l'aide de la boîte de dialogue des propriétés de l'Explorateur Windows : dans l'Explorateur Windows , cliquez avec le bouton droit de la souris sur **IBM.XMS.WCF.dll** > **Propriétés** > **Version**.
- A partir des informations d'en-tête des canaux FFST ou des fichiers de trace. Pour plus d'informations sur les informations d'en-tête FFST , voir «WCF XMS First Failure Support Technology (FFST)», à la page 650

Conseils et astuces WCF

Les conseils et astuces suivants ne sont pas dans un ordre significatif et peuvent être ajoutés lorsque de nouvelles versions de la documentation sont publiées. Il s'agit de sujets qui peuvent vous faire gagner du temps s'ils sont pertinents pour le travail que vous effectuez.

Externalisation des exceptions à partir de l'hôte de service WCF

Pour les services hébergés à l'aide de l'hôte de service WCF, toutes les exceptions non traitées émises par le service, les éléments internes WCF ou la pile de canaux ne sont pas externalisées par défaut. Pour être informé de ces exceptions, un gestionnaire d'erreurs doit être enregistré.

Le code suivant fournit un exemple de définition du comportement du service de gestionnaire d'erreurs qui peut être appliqué en tant qu'attribut d'un service:

```
using System.ServiceModel.Dispatcher;
using System.Collections.ObjectModel;
....
    public class ErrorHandlerBehaviorAttribute : Attribute, IServiceBehavior, IErrorHandler
    {
        //
        // IServiceBehavior Interface
        //
        public void AddBindingParameters(ServiceDescription serviceDescription,
           ServiceHostBase serviceHostBase, Collection<ServiceEndpoint> endpoints,
           BindingParameterCollection bindingParameters)
        {
        }
        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
            ServiceHostBase serviceHostBase)
        {
            foreach (ChannelDispatcher channelDispatcher in serviceHostBase.ChannelDispatchers)
{
                channelDispatcher.ErrorHandlers.Add(this);
            }
        }
        public void Validate(ServiceDescription serviceDescription, ServiceHostBase
serviceHostBase)
        {
        }

        //
        // IErrorHandler Interface
        //
```

```
        public bool HandleError(Exception e)
        {
            // Process the exception in the required way, in this case just outputting to the
console
            Console.Out.WriteLine(e);

            // Always return false to allow any other error handlers to run
            return false;
        }
        public void ProvideFault(Exception error, MessageVersion version, ref Message fault)
        {
        }
    }
```

Traitement des différents noms d'élément de réponse SOAP

WCF s'attend à ce que le nom d'une valeur renvoyée soit dans un format spécifique par défaut, mais un service risque de ne pas renvoyer un élément avec son nom dans le format attendu.

WCF a la convention d'attendre que la valeur renvoyée soit nommée au format suivant: *methodName*`Result` où *methodName* est le nom de l'opération de service. Par exemple, pour un service appelé `getQuote`, WCF attend que la réponse soit appelée: `getQuoteResult`.

Toutefois, le service peut renvoyer un élément dont le nom n'est pas conforme à ce format.

Lors de l'exécution de l'outil scvutil pour générer un client proxy, si le WSDL spécifie un nom différent, l'interface de proxy ajoute des paramètres pour indiquer à WCF le nom à rechercher. Exemple :

```
[System.ServiceModel.OperationContractAttribute(Action = "", ReplyAction = "*")]
[System.ServiceModel.XmlSerializerFormatAttribute(Style =
System.ServiceModel.OperationFormatStyle.Rpc,
                                                  Use =
System.ServiceModel.OperationFormatUse.Encoded)]
[return: System.ServiceModel.MessageParameterAttribute(Name = "getQuoteReturn")]
float getQuote(string in0);
```

Si vous créez votre propre interface (par exemple, en ajoutant une méthode de demande-réponse à une interface de proxy existante), vous devez vous assurer d'ajouter les mêmes paramètres à l'interface si le service renvoie un nom différent. Si vous ne le faites pas, le problème le plus courant est qu'un appel à la méthode de service renvoie toujours une valeur null ; si un objet est renvoyé, la méthode renvoie la valeur null, mais si une valeur numérique telle qu'un entier est renvoyée, la méthode renvoie la valeur 0.

Utilisation de C++

WebSphere MQ fournit des classes C++ équivalentes aux objets WebSphere 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.

A partir de WebSphere MQ version 7.0, les améliorations apportées aux interfaces de programmation WebSphere MQ ne seront pas appliquées aux classes C + +.

WebSphere MQ C++ offre les fonctions suivantes:

- Initialisation automatique des structures de données WebSphere 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.
- IMS Transmission et réception de l'en-tête de pont.
- Transmission et réception de l'en-tête de message de référence.
- Déclencher la réception des messages.
- Transmission et réception de l'en-tête de pont CICS .
- 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 WebSphere MQ de l'interface MQI procédurale (par exemple, en utilisant C) qui possèdent 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.

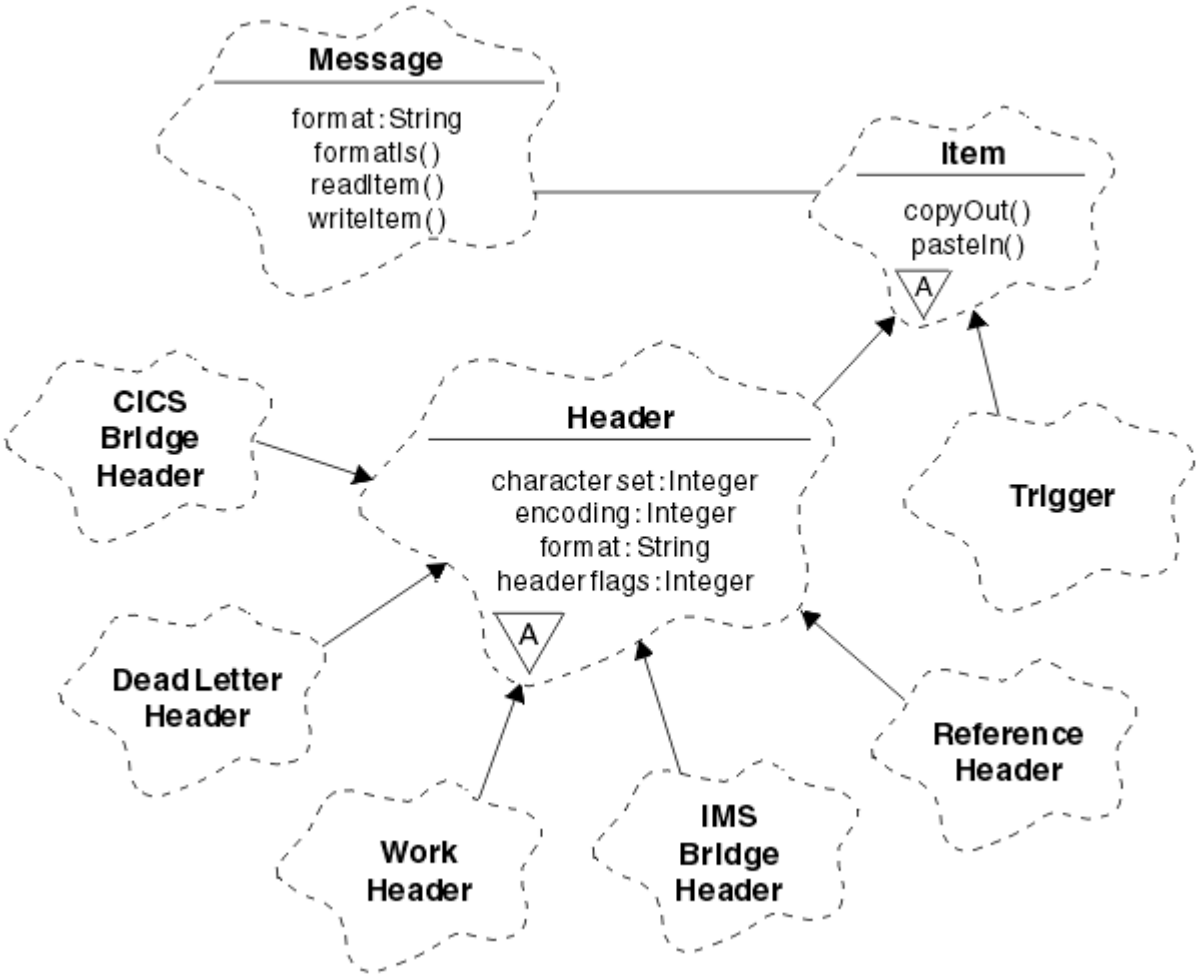


Figure 120. WebSphere MQ Classes C++ (gestion des éléments)

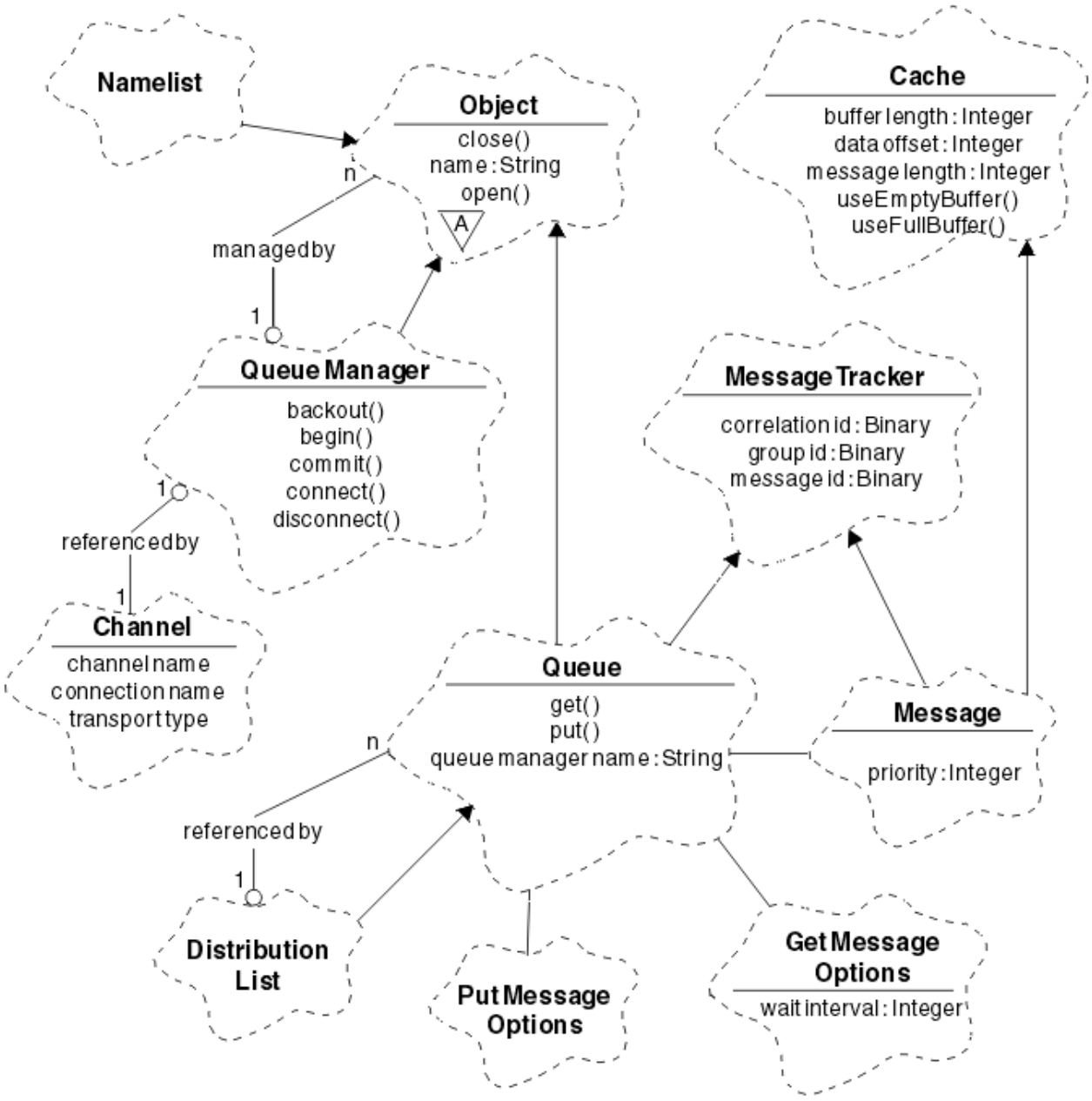


Figure 121. WebSphere MQ Classes C++ (gestion des files d'attente)

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 121, à la page 654) et les classes de traitement d'élément (voir Figure 120, à la page 653):

- 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. Les 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 sous la hiérarchie de classe ImqObject (voir Classe C++ImqObject) 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 ImqMessage Classe C++) 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 665) 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 655
Quatre exemples de programmes sont fournis pour illustrer l'obtention et l'insertion de messages.

«Remarques sur le langage C++», à la page 659
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 663
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.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Développement d'applications», à la page 7
IBM WebSphere MQ fournit plusieurs façons de développer des applications pour envoyer et recevoir les 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.

Référence associée

«Génération de programmes C++ WebSphere MQ», à la page 670
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 WebSphere MQ .

Références croisées C++ et MQI

Classes C++ WebSphere MQ

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 80, à la page 656.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Tableau 80. 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
HP-UX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/ah (voir la remarque «2», à la page 656)
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/as
Linux	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\ exemples	*MQ_INSTALLATION_PATH*\tools\cplus\ exemples \bin\vn (voir la remarque «3», à la page 656)

Remarques :

1. Les programmes générés à l'aide du compilateur ILE C++ pour IBM i se trouvent dans la bibliothèque QMQM. Les fichiers d'inclusion se trouvent dans /QIBM/ProdData/mqm/inc.
2. Les programmes générés à l'aide du compilateur HP ANSI C++ se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/samp/bin/ah. Pour plus d'informations, reportez-vous à la section «Génération de programmes C++ sous HP-UX», à la page 671.
3. Les programmes générés à l'aide de Microsoft 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++ sous Windows», à la page 677.

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 WebSphere 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 WebSphere 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)

Exemple de code

```
extern "C" {
#include <stdio.h>
}

#include <imqi.hpp> // WebSphere MQ C++

#define EXISTING_QUEUE "SYSTEM.DEFAULT.LOCAL.QUEUE"

#define BUFFER_SIZE 12

static char gpszHello[ BUFFER_SIZE ] = "Hello world" ;
int main ( int argc, char * * argv ) {
  ImqQueueManager manager ;
  int iReturnCode = 0 ;

  // Connect to the queue manager.
  if ( argc > 2 ) {
    manager.setName( argv[ 2 ] );
  }
  if ( manager.connect( ) ) {
    ImqQueue * pqueue = new ImqQueue ;
    ImqMessage * pmsg = new ImqMessage ;

    // Identify the queue which will hold the message.
    pqueue -> setConnectionReference( manager );
    if ( argc > 1 ) {
      pqueue -> setName( argv[ 1 ] );


      // The named queue can be a model queue, which will result in
      // the creation of a temporary dynamic queue, which will be
      // destroyed as soon as it is closed. Therefore we must ensure
      // that such a queue is not automatically closed and reopened.
      // We do this by setting open options which will avoid the need
      // for closure and reopening.
      pqueue -> setOpenOptions( MQOO_OUTPUT | MQOO_INPUT_SHARED |
                                MQOO_INQUIRE );
    } else {
      pqueue -> setName( EXISTING_QUEUE );

      // The existing queue is not a model queue, and will not be
      // destroyed by automatic closure and reopening. Therefore we
      // will let the open options be selected on an as-needed basis.
      // The queue will be opened implicitly with an output option
      // during the "put", and then implicitly closed and reopened
      // with the addition of an input option during the "get".
    }

    // Prepare a message containing the text "Hello world".
    pmsg -> useFullBuffer( gpszHello , BUFFER_SIZE );
    pmsg -> setFormat( MQFMT_STRING );

    // Place the message on the queue, using default put message
    // Options.
    // The queue will be automatically opened with an output option.
    if ( pqueue -> put( * pmsg ) ) {
      ImqString strQueue( pqueue -> name( ) );

      // Discover the name of the queue manager.
```

```
      ImqString strQueueManagerName( manager.name( ) );
      printf( "The queue manager name is %s.\n",
              (char *)strQueueManagerName );

      // Show the name of the queue.
      printf( "Message sent to %s.\n", (char *)strQueue );

      // Retrieve the data message just sent ("Hello world" expected)
      // from the queue, using default get message options. The queue
      // is automatically closed and reopened with an input option
      // if it is not already open with an input option. We get the
      // message just sent, rather than any other message on the
      // queue, because the "put" will have set the ID of the message
      // so, as we are using the same message object, the message ID
      // acts as in the message object, a filter which says that we
      // are interested in a message only if it has this
      // particular ID.

      if ( pqueue -> get( * pmsg ) ) {
        int iDataLength = pmsg -> dataLength( );

        // Show the text of the received message.
        printf( "Message of length %d received, ", iDataLength );

        if ( pmsg -> formatIs( MQFMT_STRING ) ) {
          char * pszText = pmsg -> bufferPointer( );

          // If the last character of data is a null, then we can
          // assume that the data can be interpreted as a text
          // string.
          if ( ! pszText[ iDataLength - 1 ] ) {
            printf( "text is \"%s\".\n", pszText );
          } else {
            printf( "no text.\n" );
          }

        } else {
          printf( "non-text message.\n" );
        }
      } else {
        printf( "ImqQueue::get failed with reason code %ld\n",
                pqueue -> reasonCode( ) );
        iReturnCode = (int)pqueue -> reasonCode( );
      }

    } else {
      printf( "ImqQueue::open/put failed with reason code %ld\n",
              pqueue -> reasonCode( ) );
      iReturnCode = (int)pqueue -> reasonCode( );
    }

    // Deletion of the queue will ensure that it is closed.
    // If the queue is dynamic then it will also be destroyed.
    delete pqueue ;
    delete pmsg ;

  } else {
    printf( "ImqQueueManager::connect failed with reason code %ld\n"
            manager.reasonCode( ) );
    iReturnCode = (int)manager.reasonCode( );
  }

  // Destruction of the queue manager ensures that it is
  // disconnected.  If the queue object were still available
  // and open (which it is not), the queue would be closed
  // prior to disconnection.

  return iReturnCode ;
}
```

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 d'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.

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 sur 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 WebSphere MQ en langage C + +.

Ces fichiers d'en-tête sont récapitulés dans le tableau suivant.

Tableau 81. 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 WebSphere 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.

Les méthodes qui définissent les données de type caractère (**char ***) utilisent toujours une copie des données, mais certaines méthodes peuvent tronquer la copie, car certaines limites sont imposées par WebSphere MQ.

La classe ImqString (voir ImqString C++ class) encapsule la classe **char *** traditionnelle 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 la 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++ WebSphere MQ sont indépendantes de la plateforme WebSphere MQ . Ils peuvent donc offrir des fonctions qui ne sont pas prises en charge sur certaines plateformes.

Si vous tentez d'utiliser une fonction sur une plateforme sur laquelle elle n'est pas prise en charge, la fonction est détectée par WebSphere MQ , mais pas par les liaisons de langage C + +. WebSphere 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 WebSphere 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 à la passerelle WebSphere 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 le pont CICS en C++

Exemple de code de programme pour l'écriture d'un message sur le pont CICS .

Les messages envoyés à WebSphere MQ for z/OS à l'aide de la passerelle CICS requièrent un en-tête spécial. L'en-tête de pont CICS 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 WebSphere 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++ WebSphere 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 WebSphere MQ .

Les compilateurs de chaque plateforme et version prise en charge de WebSphere MQ sont répertoriés sur la page de la configuration système requise pour WebSphere MQ à l'adresse Configuration système requise pour IBM WebSphere MQ.

La commande dont vous avez besoin pour compiler et lier votre programme C++ WebSphere MQ dépend de votre installation et de vos exigences. Les exemples qui suivent présentent des commandes de compilation et de liaison standard pour certains des compilateurs utilisant l'installation par défaut de WebSphere MQ sur un certain nombre de plateformes.

Génération de programmes C++ sous AIX

Générez des programmes C++ WebSphere MQ sous AIX à l'aide du compilateur XL C Enterprise Edition .

Client

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Application 32 bits sans unités d'exécution

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia -limqb23ia -lmqic
```

Application à unités d'exécution 32 bits

```
xlC_r -o imqsputc_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Application 64 bits sans unités d'exécution

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Application à unités d'exécution 64 bits

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

serveur

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Application 32 bits sans unités d'exécution

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia -limqb23ia -lmqm
```

Application à unités d'exécution 32 bits

```
xlC_r -o imqsput_32_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Application 64 bits sans unités d'exécution

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Application à unités d'exécution 64 bits

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

Génération de programmes C++ sous HP-UX

Générez des programmes WebSphere MQ C++ sur HP-UX à l'aide des compilateurs aC+ + ou aCC .

Sous HP-UX Itanium, WebSphere MQ prend en charge uniquement l'environnement d'exécution Standard. Utilisez le compilateur aCC .

- libimqi23bh.sl fournit les classes C++ WebSphere MQ pour l'environnement d'exécution Standard.
- Pour la compatibilité avec les versions antérieures, un lien symbolique est fourni entre libimqi23ah.sl et libimqi23bh.sl.

IA64 (IPF)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: IA64 (IPF)

Application 32 bits sans unités d'exécution

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqic
```

Application à unités d'exécution 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqic_r -lpthread
```

Application 64 bits sans unités d'exécution

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqic
```

Application à unités d'exécution 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsputc_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqic_r
-lpthread
```

Serveur: IA64 (IPF)

Application 32 bits sans unités d'exécution

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh -lmqm
```

Application à unités d'exécution 32 bits

```
aCC -Wl,+b,: +e -D_HPUX_SOURCE -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -L/usr/lib/hpux32 -limqi23bh_r -lmqm_r -lpthread
```

Application 64 bits sans unités d'exécution

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64 imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh -lmqm
```

Application à unités d'exécution 64 bits

```
aCC +DD64 +e -D_HPUX_SOURCE -o imqsput_64_r imqsput.cpp
-IMQ_INSTALLATION_PATH/inc -LMQ_INSTALLATION_PATH/lib64 -L/usr/lib/hpux64 -limqi23bh_r
-lmqm_r
-lpthread
```

Génération de programmes C++ sur Linux

Générez des programmes WebSphere MQ C++ sous Linux à l'aide du compilateur GNU g + +.

Système p

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: System p

Application 32 bits sans unités d'exécution

```
g++ -m32 -o imqsputc_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl
-limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m32 -o imqsputc_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r
-limqb23gl_r -lmqic_r
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -o imqsputc_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -o imqsputc_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r
```

Serveur: System p

Application 32 bits sans unités d'exécution

```
g++ -m32 -o imqsput_32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl
-limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m32 -o imqsput_r32 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r
-limqb23gl_r -lmqm_r
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -o imqsput_64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -o imqsput_r64 imqsput.cpp -fsigned-char -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r
```

System z

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: System z

Application 32 bits sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m31 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqc23gl_r -limqb23gl_r -lmqic_r
-lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64  -Wl,-rpath=/usr/lib64
-limqc23gl_r -limqb23gl_r -lmqic_r -lpthread
```

Serveur: System z

Application 32 bits sans unités d'exécution

```
g++ -m31 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib  -Wl,-rpath=/usr/lib
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m31 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64
-limqs23gl_r -limqb23gl_r -lmqm_r -lpthread
```

System x (32 bits)

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: System x (32 bits)

Application 32 bits sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,
-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl -limqb23gl -lmqic
```

Application à unités d'exécution 32 bits

```
g++ -m32 -fsigned-char -o imqsputc_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl -limqb23gl
-lmqic
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsputc_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqc23gl_r -limqb23gl_r
-lmqic_r -lpthread
```

Serveur: System x (32 bits)

Application 32 bits sans unités d'exécution

```
g++ -m32 -fsigned-char -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 32 bits

```
g++ -m32 -fsigned-char -o imqsput_32_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATHlib -Wl,-rpath=MQ_INSTALLATION_PATH/lib -LMQ_INSTALLATION_PATH/lib
-Wl,-rpath=MQ_INSTALLATION_PATH/lib -Wl,-rpath=/usr/lib -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Application 64 bits sans unités d'exécution

```
g++ -m64 -fsigned-char -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl -limqb23gl -lmqm
```

Application à unités d'exécution 64 bits

```
g++ -m64 -fsigned-char -o imqsput_64_r imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -LMQ_INSTALLATION_PATH/
lib64
-Wl,-rpath=MQ_INSTALLATION_PATH/lib64 -Wl,-rpath=/usr/lib64 -limqs23gl_r -limqb23gl_r
-lmqm_r -lpthread
```

Génération de programmes C++ sous Solaris

Générez des programmes C++ WebSphere MQ sous Solaris à l'aide du compilateur Sun ONE.

SPARC

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: SPARC

Application 32 bits

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Application 64 bits

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Serveur: SPARC

Application 32 bits

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Application 64 bits

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Client: x86-64

Application 32 bits

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Application 64 bits

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Serveur: x86-64

Application 32 bits

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
```

```
-LMQ_INSTALLATION_PATH/lib -RMQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Application 64 bits

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -IMQ_INSTALLATION_PATH/inc
-LMQ_INSTALLATION_PATH/lib64 -RMQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

Génération de programmes C++ sous Windows

Générez des programmes C++ WebSphere MQ sous Windows à l'aide du compilateur Microsoft Visual Studio C + +.

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 WebSphere MQ est installé.

Client

```
cl -MD imqsput.cpp /Feimqsputc.exe imqb23vn.lib imqc23vn.lib
```

serveur

```
 cl -MD imqsput.cpp /Feimqsput.exe imqb23vn.lib imqs23vn.lib
```

Utilisation des classes WebSphere MQ pour Java

Les classes WebSphere MQ for Java vous permettent d'utiliser WebSphere MQ dans un environnement Java. Une application Java peut utiliser des classes WebSphere MQ pour Java ou des classes WebSphere MQ pour JMS pour accéder aux ressources WebSphere MQ .

Les classes WebSphere MQ pour Java permettent à une application Java de:

- Connexion à WebSphere MQ en tant que client WebSphere MQ
- Connexion directe à un gestionnaire de files d'attente WebSphere MQ

Les classes WebSphere MQ pour Java encapsulent l'interface MQI (Message Queue Interface), l'API WebSphere MQ native.

Les classes WebSphere MQ pour Java utilisent un modèle d'objet similaire aux interfaces C++ et .NET pour WebSphere MQ.

Pourquoi utiliser des classes WebSphere MQ pour Java?

Si les points suivants sont importants dans votre installation, envisagez d'utiliser les classes WebSphere MQ pour Java:

- Les classes WebSphere MQ pour Java encapsulent l'interface MQI (Message Queue Interface), l'API WebSphere MQ native.
  - Si vous êtes familiarisé avec l'utilisation de l'interface MQI dans les langages procéduraux, vous pouvez transférer ces connaissances dans l'environnement Java.
  - Vous pouvez exploiter toute la gamme des fonctions de WebSphere MQ, en plus de celles disponibles via JMS.

- Les classes WebSphere MQ pour Java utilisent un modèle d'objet similaire aux interfaces C++ et .NET pour WebSphere MQ. Si vous connaissez bien ces interfaces, vous pouvez transférer ces connaissances dans l'environnement Java.

Remarque : La reconnexion automatique du client n'est pas prise en charge par WebSphere MQ classes for Java.

Initiation à WebSphere MQ classes for Java

Cette collection de rubriques fournit une présentation des classes WebSphere MQ pour Java et de leurs utilisations.

Que sont les classes WebSphere MQ pour Java?

Les classes WebSphere MQ pour Java vous permettent d'utiliser WebSphere MQ dans un environnement Java.

Les classes WebSphere MQ pour Java permettent à une application Java de:

- Connexion à WebSphere MQ en tant que client WebSphere MQ
- Connexion directe à un gestionnaire de files d'attente WebSphere MQ

Les classes WebSphere MQ pour Java encapsulent l'interface MQI (Message Queue Interface), l'API WebSphere MQ native.

Les classes WebSphere MQ pour Java utilisent un modèle d'objet similaire aux interfaces C++ et .NET pour WebSphere MQ.

Pourquoi utiliser des classes WebSphere MQ pour Java?

Une application Java peut utiliser des classes WebSphere MQ pour Java ou des classes WebSphere MQ pour JMS pour accéder aux ressources WebSphere MQ . L'utilisation des classes WebSphere MQ pour Java présente un certain nombre d'avantages.

Si les points suivants sont importants dans votre installation, envisagez d'utiliser les classes Websphere MQ pour Java:

- Les classes WebSphere MQ pour Java encapsulent l'interface MQI (Message Queue Interface), l'API WebSphere MQ native.
  - Si vous êtes familiarisé avec l'utilisation de l'interface MQI dans les langages procéduraux, vous pouvez transférer ces connaissances dans l'environnement Java.
  - Vous pouvez exploiter toute la gamme des fonctions de WebSphere MQ, en plus de celles disponibles via JMS.
- Les classes WebSphere MQ pour Java utilisent un modèle d'objet similaire aux interfaces C++ et .NET pour WebSphere MQ. Si vous connaissez bien ces interfaces, vous pouvez transférer ces connaissances dans l'environnement Java.

Options de connexion pour WebSphere MQ classes for Java

WebSphere Les classes MQ pour Java peuvent se connecter en mode client ou liaisons.

Les options programmables permettent à WebSphere MQ classes for Java de se connecter à WebSphere MQ de l'une des manières suivantes:

- En tant que client WebSphere MQ MQI utilisant le protocole TCP/IP (Transmission Control Protocol/ Internet Protocol)
- En mode liaisons, connexion directe à WebSphere MQ à l'aide de l'interface JNI (Java Native Interface)

Les clients ne peuvent pas être exécutés sur z/OS, mais les clients d'autres plateformes peuvent se connecter à un gestionnaire de files d'attente WebSphere MQ for z/OS si la fonction Client Attach Facility est installée.

Les sections suivantes décrivent plus en détail les options de connexion en mode client et en mode liaison.

Connexion client

Pour se connecter à un gestionnaire de files d'attente en mode client, une application WebSphere MQ classes for Java peut s'exécuter sur le même système que le gestionnaire de files d'attente ou sur un autre système. Dans chaque cas, WebSphere MQ classes for Java se connecte au gestionnaire de files d'attente via TCP/IP.

Une application WebSphere 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 plus d'informations sur la façon d'écrire des applications pour utiliser des connexions en mode client, voir «WebSphere Classes MQ pour les modes de connexion Java», à la page 693.

Connexion par liaisons

Lorsqu'elles sont utilisées en mode liaisons, les classes WebSphere MQ for Java utilisent l'interface JNI (Java Native Interface) pour appeler directement dans 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 classes WebSphere MQ pour les applications Java que la connexion en mode client, en évitant le coût des communications TCP/IP.

Les applications qui utilisent les classes WebSphere MQ pour 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 les classes WebSphere MQ pour l'application Java, doit être configuré pour charger les classes WebSphere MQ pour les bibliothèques Java ; pour plus d'informations, voir Les classes WebSphere MQ pour les bibliothèques Java .

Pour plus d'informations sur l'écriture d'applications afin d'utiliser des connexions en mode liaisons, voir «WebSphere Classes MQ pour les modes de connexion Java», à la page 693.

Prérequis pour WebSphere MQ classes for Java

Pour utiliser WebSphere MQ classes for Java, vous avez besoin de certains autres produits logiciels.

Pour obtenir les informations les plus récentes sur les prérequis pour les classes WebSphere MQ for Java, voir le fichier README WebSphere MQ .

Pour développer des classes WebSphere MQ pour les applications 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 sur la page de la configuration système requise pour WebSphere MQ à l'adresse Configuration système requise pour IBM WebSphere MQ.

Pour exécuter des classes WebSphere MQ pour les applications Java, vous avez besoin des composants logiciels suivants:

- Un gestionnaire de files d'attente WebSphere MQ , pour les applications qui se connectent à un gestionnaire de files d'attente
- Un environnement JRE (Java Runtime Environment) pour chaque système sur lequel vous exécutez des applications. Un environnement d'exécution Java approprié est fourni avec WebSphere MQ.

Si vous avez besoin de connexions SSL 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 WebSphere MQ pour les applications Java si IPv6 est pris en charge par votre machine virtuelle Java (JVM) et l'implémentation TCP/IP sur votre système d'exploitation.

Installation et configuration des classes WebSphere MQ pour Java

Cette section décrit les répertoires et les fichiers créés lors de l'installation de WebSphere MQ classes for Java et explique comment configurer WebSphere MQ classes for Java après l'installation.

Ce qui est installé pour WebSphere MQ classes for Java

La version la plus récente de WebSphere MQ classes for Java est installée avec WebSphere 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 WebSphere MQ , voir:

Installation d'un serveur WebSphere MQ
Installation d'un client IBM WebSphere MQ

WebSphere Les classes MQ pour Java sont contenues dans les fichiers d'archive Java (JAR), com.ibm.mq.jaret 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.

Installation et mise à niveau des fichiers JAR WebSphere MQ pour Java

La seule méthode prise en charge pour obtenir les classes WebSphere MQ pour les fichiers JAR Java sur un système consiste à installer le produit WebSphere MQ ou le client WebSphere MQ MQI SupportPac, ou à utiliser un outil de gestion de logiciels tel que Apache Maven, pour plus d'informations, voir «IBM WebSphere MQ classes for Java et outils de gestion de logiciels», à la page 688.

Ne déplacez pas ou ne copiez pas les classes WebSphere MQ pour les fichiers JAR Java à partir d'autres machines, sauf si vous utilisez un outil de gestion de logiciels.

- Les groupes de correctifs ne peuvent pas être appliqués à une "installation" dans laquelle les fichiers JAR ont été copiés à partir d'une autre machine, et il est beaucoup plus difficile de s'assurer que tous les fichiers JAR sont conservés en même temps que les autres et qu'ils sont à des niveaux compatibles.
- La copie des fichiers JAR WebSphere MQ classes for JMS entre les machines peut également générer plusieurs copies des fichiers résidant sur la même machine, ce qui peut entraîner des problèmes de maintenance du code et des problèmes de débogage.

N'incluez pas les classes WebSphere MQ pour les fichiers JAR Java dans les archives d'application.

- Les mises à jour des classes WebSphere MQ pour Java ne peuvent pas être appliquées à l'aide d'un groupe de correctifs WebSphere MQ .
- Le support IBM ne peut pas déterminer facilement la version des classes WebSphere MQ pour Java qui sont utilisées par l'application.
- Des problèmes peuvent se produire si plusieurs applications s'exécutant dans le même environnement d'exécution Java incluent des versions différentes des classes WebSphere MQ pour Java, car plusieurs versions des classes WebSphere MQ pour Java sont chargées simultanément dans l'environnement d'exécution Java.
- Si une application utilise le transport BINDINGS pour se connecter à un gestionnaire de files d'attente, toute mise à niveau majeure du gestionnaire de files d'attente nécessite également la mise à jour de l'application pour inclure le niveau correspondant des classes WebSphere MQ pour Java.

Par exemple, si un gestionnaire de files d'attente est mis à niveau vers le niveau WebSphere MQ version 7.1 , toutes les applications qui se connectent au gestionnaire de files d'attente à l'aide du transport BINDINGS doivent également être mises à jour pour inclure les classes WebSphere MQ version 7.1 pour Java.

La bibliothèque Java suivante est distribuée avec WebSphere MQ classes for Java:

- connector.jar (version 1.0)

Le modèle d'application appelé Postcard se trouve dans le fichier JAR com.ibm.mq.postcard.jar.

L'outil Javadoc a été utilisé pour générer les pages HTML contenant les spécifications des classes WebSphere MQ pour Java et des classes WebSphere MQ pour les API JMS. Les pages HTML se trouvent dans le sous-répertoire doc du répertoire d'installation WebSphere MQ classes for JMS. Sur les systèmes UNIX, Linuxet Windows , le sous-répertoire doc contient les pages HTML individuelles.

Une fois l'installation terminée, les fichiers et les exemples sont installés dans les emplacements indiqués dans «Répertoires d'installation pour WebSphere MQ classes for Java», à la page 681.

Après l'installation, sur toute plateforme autre que Windows, vous devez mettre à jour vos variables d'environnement comme décrit dans «Variables d'environnement pertinentes pour WebSphere MQ classes for Java», à la page 681.

Répertoires d'installation pour WebSphere MQ classes for Java

WebSphere Les classes MQ pour les fichiers Java sont installées dans des emplacements différents en fonction de la plateforme.

Tableau 82, à la page 681 montre où sont installés les fichiers WebSphere MQ classes for Java.

Tableau 82. WebSphere Classes MQ pour les répertoires d'installation Java

Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/java/lib
HP-UX, Linuxet Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.	

Certains modèles d'application, tels que les programmes de vérification d'installation (IVP), sont fournis avec WebSphere MQ. Tableau 83, à la page 681 indique où les modèles d'application sont installés. Les classes WebSphere MQ pour les exemples Java se trouvent dans un sous-répertoire appelé `wmqjava`. Les exemples PCF se trouvent dans un sous-répertoire appelé `pcf`.

Tableau 83. Répertoires d'exemples

Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
HP-UX, Linuxet Solaris	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Windows	*MQ_INSTALLATION_PATH*\tools\wmqjava\
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.	

Variables d'environnement pertinentes pour WebSphere MQ classes for Java

Si vous souhaitez exécuter des classes WebSphere MQ pour les applications Java, leurs chemins d'accès aux classes doivent inclure les classes WebSphere MQ pour Java et les répertoires d'exemples.

Pour que les applications WebSphere MQ classes for Java puissent s'exécuter, leur chemin d'accès aux classes doit inclure le répertoire WebSphere 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.

La Tableau 84, à la page 682 présente le paramètre CLASSPATH approprié à utiliser sur chaque plateforme pour exécuter les classes WebSphere MQ pour les applications Java, y compris les modèles d'application.

Tableau 84. Paramètre CLASSPATH pour exécuter les classes WebSphere MQ pour les applications Java, y compris les classes WebSphere MQ pour les exemples d'applications Java

Plateforme	**Paramètre CLASSPATH**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
HP-UX, Linuxet Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/samp/wmqjava/samples:
Windows	CLASSPATH=*MQ_INSTALLATION_PATH*\Java\lib\com.ibm.mq.jar; *MQ_INSTALLATION_PATH*\tools\wmqjava\samples;
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.	

Si vous effectuez la compilation à l'aide de l'option -Xlint, il se peut qu'un message vous avertissant 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 IBM WebSphere MQ Advanced Message Security.

Les scripts fournis avec WebSphere MQ classes for Java 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 WebSphere MQ classes for Java sont installées, comme indiqué dans WebSphere MQ classes for Java installation directory.

MQ_JAVA_LIB_PATH
Cette variable d'environnement indique le répertoire dans lequel sont stockées les classes WebSphere MQ pour les bibliothèques Java, comme indiqué dans Emplacement des classes WebSphere MQ pour les bibliothèques Java pour chaque plateforme. Certains scripts fournis avec WebSphere MQ classes for Java, tels qu'IVTRun, utilisent cette variable d'environnement.

Sous Windows, toutes les variables d'environnement sont définies automatiquement lors de l'installation. Sur toute autre plateforme, vous devez les définir vous-même.Sur un système UNIX , 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. Sous AIX, HP-UX, Linuxet Solaris, ces scripts se trouvent dans le répertoire *MQ_INSTALLATION_PATH*/java/bin .

Les classes IBM WebSphere MQ pour les bibliothèques Java

L'emplacement des bibliothèques IBM WebSphere 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 aux classes WebSphere MQ pour les bibliothèques Java, qui incluent les bibliothèques JNI. Tableau 85, à la page 683 affiche l'emplacement des classes WebSphere MQ pour les bibliothèques Java pour chaque plateforme.

Tableau 85. Emplacement des bibliothèques WebSphere MQ pour chaque plateforme

Plateforme	**Répertoire contenant les bibliothèques WebSphere MQ pour Java**
AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
HP-UX Linux (POWER, x86-64 et zSeries s390x (plateformes) Solaris (plateformesx86-64 et SPARC)	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Linux (plateformex86)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\Java\lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*\Java\lib64 (bibliothèques 64 bits)
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.	

Remarque :

1. Sous AIX, HP-UX, Linux (Power platform) ou Solaris, 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. Sous Windows, vous pouvez utiliser la variable d'environnement PATH pour spécifier l'emplacement des classes WebSphere MQ pour les bibliothèques Java au lieu de spécifier leur emplacement dans la commande **java** .
3. Pour utiliser WebSphere MQ classes for Java en mode liaisons sous IBM i, vérifiez que la bibliothèque QMQMJAVA figure dans votre liste de bibliothèques.

Tâches associées

Utilisation des classes WebSphere MQ pour Java

Prise en charge d'OSGi sur IBM WebSphere MQ classes for Java

OSGi fournit une infrastructure qui prend en charge le déploiement d'applications en tant que bundles. Un bundle OSGi est fourni avec IBM WebSphere MQ classes for Java .

OSGi fournit une infrastructureJava 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.

IBM WebSphere MQ classes for Java. inclut le bundle OSGi suivant.

com.ibm.mq.osgi.java_< numéro de version > .jar
Fichiers JAR permettant aux applications d'utiliser IBM WebSphere MQ classes for Java.

où < numéro de version > est le numéro de version de WebSphere MQ qui a été installé.

Le bundle est installé dans le sous-répertoire `java/lib/OSGi` de votre installation IBM WebSphere MQ ou dans le dossier `java\lib\OSGi` sous Windows.

Neuf autres bundles sont également installés dans le sous-répertoire `java/lib/OSGi` de votre installation IBM WebSphere MQ ou dans le dossier `java\lib\OSGi` sous Windows. Ces bundles font partie de IBM WebSphere MQ classes for JMS et ne doivent pas être chargés dans un environnement d'exécution OSGi dans lequel le bundle IBM WebSphere MQ classes for Java est chargé. Si le bundle OSGi IBM WebSphere MQ classes for Java est chargé dans un environnement d'exécution OSGi dans lequel les bundles IBM WebSphere MQ classes for JMS sont également chargés, des erreurs telles que:

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

lorsque des applications utilisant le bundle IBM WebSphere MQ classes for Java ou les bundles IBM WebSphere MQ classes for JMS sont exécutées.

Le bundle OSGi pour IBM WebSphere 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 WebSphere 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 prendre en compte le bundle IBM WebSphere MQ classes for Java , mais le bundle IBM WebSphere MQ classes for Java ne prend en compte aucun bundle d'utilisateurs. Par conséquent, le chargeur de classe utilisé dans un bundle IBM WebSphere 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 .

Le fichier de configuration IBM WebSphere MQ classes for Java

Un fichier de configuration IBM WebSphere MQ classes for Java spécifie les propriétés qui sont utilisées pour configurer le IBM WebSphere MQ classes for Java.

Le format d'un fichier de configuration IBM WebSphere MQ classes for Java est celui d'un fichier de propriétés Java standard.

V 7.5.0.9 Depuis IBM WebSphere MQ Version 7.5.0, groupe de correctifs 9, un exemple de fichier de configuration appelé `mqjava.config` est fourni dans le sous-répertoire `bin` du répertoire d'installation de IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java en tant que fichier `D:\mydir\mqjava.config` sur le système Windows local.

Un fichier de configuration IBM WebSphere 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 WebSphere MQ classes for Java

Un fichier de configuration IBM WebSphere MQ MQI client peut également spécifier les propriétés utilisées pour configurer IBM WebSphere MQ classes for Java. Toutefois, les propriétés qui sont spécifiées dans un fichier de configuration IBM WebSphere 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 WebSphere MQ MQI client en le spécifiant comme propriété dans un fichier de configuration IBM WebSphere MQ classes for Java . Pour remplacer un attribut dans un fichier de configuration IBM WebSphere MQ MQI client , utilisez une entrée au format suivant dans le fichier de configuration IBM WebSphere 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 du fichier de configuration IBM WebSphere MQ MQI client qui contient l'attribut.

propName
Nom de l'attribut tel qu'il est spécifié dans le fichier de configuration IBM WebSphere MQ MQI client .

propValue
Valeur de la propriété qui remplace la valeur de l'attribut spécifiée dans le fichier de configuration IBM WebSphere MQ MQI client .

Vous pouvez également remplacer un attribut dans un fichier de configuration IBM WebSphere 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 WebSphere MQ MQI client sont pertinents pour IBM WebSphere 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 WebSphere MQ classes for Java, voir «Utilisation d'une table de définition de canal du client avec IBM WebSphere MQ classes for Java», à la page 697 .

Tableau 86. Quelle section du fichier de configuration du client contient quel attribut

stanza	Attribut
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
sectionMessageBuffer du fichier de configuration du client	MaximumSize
sectionMessageBuffer du fichier de configuration du client	PurgeTime
sectionMessageBuffer du fichier de configuration du client	UpdatePercentage

Tableau 86. Quelle section du fichier de configuration du client contient quel attribut (suite)	
stanza	**Attribut**
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

Pour plus d'informations sur la configuration de IBM WebSphere MQ MQI client , voir Configuration d'un client à l'aide d'un fichier de configuration.

Tâches associées

Traçage des applications IBM WebSphere MQ classes for Java

Section Java Standard Environment Trace

Vous pouvez utiliser la section Java Standard Environment Trace Settings pour configurer la fonction de trace de IBM WebSphere MQ classes for Java .

com.ibm.msg.client.commonservices.trace.outputName = `traceOutputName`

`traceOutputName` est le répertoire et le nom de fichier dans lesquels la sortie de trace est envoyée.

Le nom par défaut du fichier de trace dépend de la version de IBM WebSphere MQ classes for Java utilisée par une application:

- Pour IBM WebSphere MQ classes for Java for Version 7.5.0, Fix Pack 8 ou version antérieure, *traceOutputName* utilise par défaut un fichier nommé `mqjms_%PID%.trc` dans le répertoire de travail en cours.
- V 7.5.0.9 Depuis IBM WebSphere MQ classes for Java à partir de Version 7.5.0, Fix Pack 9, *traceOutputName* utilise par défaut un fichier nommé `mqjava_%PID%.trc` dans le répertoire de travail en cours.

où *%PID%* est l'ID de processus en cours. 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 WebSphere 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 la valeur par défaut `NONE`, et par conséquent, n'exclut aucun package et aucune classe dans IBM WebSphere MQ classes for Java 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 = maxArrayBytes
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 des contenus des tableaux 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 dans un tableau d'octets envoyés au fichier de trace.

com.ibm.msg.client.commonservices.trace.limit = maxTraceBytes
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 qui est ajouté au nom de fichier. Le fichier de sortie de trace en cours ou le plus récent a le suffixe `.trc.0`, le prochain fichier de sortie de trace le plus récent se termine par `.trc.1`, etc. 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 pour 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.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 est spécifié dans **traceOutputName** , ce nom est utilisé à la place et 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 WebSphere 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 le IBM WebSphere 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 le IBM WebSphere MQ classes for Java se ferme de manière contrôlée. Utilisez la compression de trace uniquement si les problèmes examinés ne provoquent pas l'arrêt inattendu de la machine virtuelle Java elle-même. N'utilisez pas la compression de trace lorsque vous diagnostiquez des problèmes qui peuvent entraîner des arrêts `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:

Tableau 87. Éléments tracés pour chaque niveau de trace

Valeur	Ce qui est tracé
0	La trace est désactivée
1	Exceptions
3	Exceptions Avertissements
6	Exceptions Avertissements Points de trace d'information
8	Exceptions Avertissements Points de trace d'information Entrée et sortie de méthode
9	Exceptions Avertissements Points de trace d'information Entrée et sortie de méthode Données envoyées entre IBM WebSphere MQ classes for Java et un gestionnaire de files d'attente.

Remarque : Utilisez toujours la valeur 9 , sauf indication contraire du support IBM .

IBM WebSphere 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 WebSphere 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 WebSphere 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 de IBM WebSphere MQ Server, IBM WebSphere MQ Client ou IBM WebSphere 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 WebSphere 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 WebSphere MQ .
- Vous êtes responsable de la mise à jour du référentiel avec toute maintenance fournie par le support IBM .

Pour IBM WebSphere MQ Version 7.5, les fichiers JAR suivants doivent être installés dans le référentiel:

- com.ibm.mq.commonservices.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- connector.jar

Configuration post-installation pour les applications IBM WebSphere MQ

Après avoir installé IBM WebSphere MQ, vous pouvez configurer votre installation afin d'exécuter vos propres applications.

N'oubliez pas de consulter le fichier README de IBM WebSphere MQ pour des informations ultérieures ou plus spécifiques à votre environnement.

Avant de tenter d'exécuter une application IBM WebSphere MQ classes for Java en mode liaisons, vérifiez que vous avez configuré IBM WebSphere MQ comme décrit dans Configuration .

Configuration de votre gestionnaire de files d'attente pour l'acceptation des connexions client à partir de WebSphere 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.

Voir «Préparation et exécution des exemples de programmes», à la page 115 pour plus de détails.

Exécution des classes WebSphere MQ pour les applications Java sous le gestionnaire de sécurité Java

WebSphere Les classes MQ pour Java peuvent s'exécuter avec le gestionnaire de sécurité Java activé. Pour exécuter des applications avec le gestionnaire de sécurité activé, vous devez configurer votre machine virtuelle Java (JVM) avec un fichier de définition de règles approprié.

La méthode la plus simple consiste à modifier le fichier de règles fourni avec votre environnement d'exécution Java. Sur la plupart des systèmes, ce fichier est stocké dans le chemin `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 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 les classes WebSphere MQ pour Java lors de l'exécution sous le gestionnaire de sécurité Java.

Voici un exemple d'entrée de fichier de règles qui permet à WebSphere MQ classes for Java de s'exécuter avec succès sous le gestionnaire de sécurité par défaut:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
```

```
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase
"file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.commonservices.jar" {
  permission java.util.PropertyPermission "user.dir","read";
  permission java.util.PropertyPermission "line.separator","read";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.mq.commonservices", "read";
  permission java.util.logging.LoggingPermission "control";
  //For access to the trace properties file.
  permission java.io.FilePermission "/tmp/trace.properties", "read";
  //For access to the trace output files.
  permission java.io.FilePermission "/tmp/*", "read,write";
};
```

Remarques :

- *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.
- Cet exemple de fichier de règles permet aux classes WebSphere MQ for Java de fonctionner correctement sous le gestionnaire de sécurité, mais vous devrez peut-être activer votre propre code pour qu'il s'exécute correctement avant que vos applications ne fonctionnent.
- Pour permettre à WebSphere MQ classes for Java d'accéder aux fichiers d'archive Java (JAR) d'une application, ajoutez les droits suivants à la première instruction `grant` :

  ```
  permission java.io.FilePermission "/path_to_your_app/-", "read";
  ```

- Pour utiliser ces instructions `grant` 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 les classes WebSphere MQ pour Java et où vous stockez vos applications.
- L'exemple de code fourni avec WebSphere MQ classes for Java n'a pas été spécifiquement activé pour être utilisé avec le gestionnaire de sécurité ; cependant, les tests IVT s'exécutent avec ce fichier de règles et le gestionnaire de sécurité par défaut en place.

Vérification des classes IBM WebSphere MQ pour l'installation de Java

Un programme de vérification de l'installation, MQIVP, est fourni avec les classes IBM WebSphere MQ pour Java. Vous pouvez utiliser ce programme pour tester tous les modes de connexion des classes IBM WebSphere MQ pour 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 «Préparation et exécution des exemples de programmes», à la page 115. 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 des classes WebSphere MQ pour Java», à la page 677.

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 pertinentes pour WebSphere MQ classes for Java», à la page 681.
4. Accédez au répertoire `MQ_INSTALLATION_PATH/mqm/VRM/java/samples/wmqjava`, où `MQ_INSTALLATION_PATH` est le chemin d'accès à votre installation IBM WebSphere MQ et `VRM` est la version, l'édition et le numéro de modification du produit. 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 WebSphere MQ classes for Java (voir The WebSphere MQ classes for Java libraries).

   A l'invite marquée (1):

   - Pour utiliser une connexion TCP/IP, entrez un nom d'hôte de serveur IBM WebSphere 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 WebSphere MQ .

```
Please enter the IP address of the MQ server              : ipaddress(1)
Please enter the port to connect to                       : (1414)(2)
Please enter the server connection channel name            : channelname(2)
Please enter the queue manager name                       : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Remarque :

1. Si vous choisissez la connexion serveur, les invites marquées (2)ne s'affichent pas.

Résolution des problèmes liés à IBM WebSphere MQ

Exécutez initialement le programme de vérification de l'installation. Vous devrez peut-être également utiliser la fonction de trace.

Si un programme ne se termine pas correctement, exécutez le programme de vérification de l'installation et suivez les conseils donnés dans les messages de diagnostic. Ce programme est décrit dans «Vérification des classes IBM WebSphere MQ pour l'installation de Java», à la page 690.

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 classes WebSphere MQ pour les bibliothèques Java (voir Classes WebSphere MQ pour les bibliothèques Java).

Pour plus d'informations sur l'utilisation de la fonction de trace, voir Traçage des applications IBM WebSphere MQ classes for Java.

Introduction pour les programmeurs

Cette collection de rubriques contient des informations générales pour les programmeurs.

Pour plus d'informations sur l'écriture de programmes, voir «Ecriture des classes WebSphere MQ pour les applications Java», à la page 692.

L'interface WebSphere MQ classes for Java

L'interface de programmation d'application WebSphere MQ utilise des instructions qui agissent sur les objets. L'interface de programmation Java utilise des objets sur lesquels vous agissez en appelant des méthodes.

L'interface de programmation d'application WebSphere 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 WebSphere MQ sur lequel elles doivent fonctionner. Etant donné que Java est orienté objet, l'interface de programmation Java effectue cette opération. Votre programme se compose d'un ensemble d'objets WebSphere 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();
```

Ecriture des classes WebSphere MQ pour les applications Java

Cette collection de rubriques fournit des informations pour vous aider à écrire des applications Java afin d'interagir avec les systèmes WebSphere MQ .

Pour utiliser les classes WebSphere MQ pour Java afin d'accéder aux files d'attente WebSphere MQ , vous devez écrire des applications Java contenant des appels qui placent des messages dans des files d'attente WebSphere MQ et en extraire des messages. Pour plus de détails sur les classes individuelles, voir WebSphere MQ classes for Java.

Remarque : La reconnexion automatique du client n'est pas prise en charge par WebSphere MQ classes for Java.

WebSphere Classes MQ pour les modes de connexion Java

La façon dont vous programmez pour WebSphere 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 WebSphere 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.

WebSphere Classes MQ pour les connexions client Java

Lorsque WebSphere MQ classes for Java est utilisé en tant que client, il est similaire à IBM WebSphere MQ MQI client, mais présente un certain nombre de différences.

Si vous programmez pour *WebSphere MQ classes for Java* à utiliser en tant que client, tenez compte des différences suivantes:

- Il ne prend en charge que TCP/IP.
- Il ne lit aucune variable d'environnement WebSphere 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 WebSphere MQ sont pertinents pour les classes WebSphere MQ pour Java. Si vous spécifiez d'autres attributs, ils sont inefficaces.

stanza	**Attribut**
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
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

- Si vous vous connectez à un gestionnaire de files d'attente qui requiert la conversion de données de type caractères, le client Java V7 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'estpas prise en charge par WebSphere MQ classes for Java.

Lorsqu'elles sont utilisées en mode client, *WebSphere MQ classes for Java* ne prend pas en charge l'appel MQBEGIN.

Pour plus d'informations sur les environnements pris en charge, voir «Options de connexion pour WebSphere MQ classes for Java», à la page 678 .

WebSphere MQ classes for Java bindings mode

Le mode de liaison des classes WebSphere MQ pour Java diffère du mode client de trois manières principales.

Lorsqu'elles sont utilisées en mode liaisons, les classes WebSphere MQ pour Java utilisent l'interface JNI (Java Native Interface) pour appeler directement dans l'API de gestionnaire de files d'attente existante, plutôt que de communiquer via un réseau.

Par défaut, les applications qui utilisent les classes WebSphere MQ pour Java en mode liaisons se connectent à un gestionnaire de files d'attente à l'aide de l'option *ConnectOption*, MQCNO_STANDARD_BINDINGS.

Les classes WebSphere MQ pour 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 217.

Le mode liaisons prend en charge l'appel MQBEGIN pour initier des unités de travail globales coordonnées par le gestionnaire de files d'attente, sur toutes les plateformes, à l'exception de WebSphere MQ for IBM i et WebSphere MQ for z/OS.

La plupart des paramètres fournis par la classe `MQEnvironment` ne sont pas pertinents pour le mode liaisons et sont ignorés.

Pour plus d'informations sur les environnements pris en charge, voir «Options de connexion pour WebSphere MQ classes for Java», à la page 678 .

Définition des classes WebSphere MQ à utiliser pour la connexion Java

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 WebSphere 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 explique comment se connecter et se déconnecter d'un gestionnaire de files d'attente à l'aide des classes WebSphere MQ pour Java.

Configuration de l'environnement WebSphere MQ pour WebSphere 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 WebSphere MQ classes for JMS», à la page 800

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 WebSphere 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 WebSphere MQ Explorer **Application Connections** (où la zone est appelée **App name**).

Les noms d'application sont limités à 28 caractères et les noms plus longs sont tronqués pour tenir. Si aucun nom d'application n'est spécifié, une valeur par défaut est fournie. Le nom par défaut est basé sur la classe d'appel (principale), mais si ces informations ne sont pas disponibles, le texte `WebSphere 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`.

Remarque : Les gestionnaires de files d'attente exécutés sur les plateformes z/OS ne prennent pas en charge la définition des noms d'application.

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 WebSphere MQ

Un fichier de configuration client WebSphere MQ peut également spécifier les propriétés utilisées pour configurer les classes WebSphere MQ pour Java. Toutefois, les propriétés spécifiées dans un fichier de configuration client WebSphere MQ MQI 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 WebSphere 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 WebSphere MQ sont pertinents pour WebSphere MQ classes for Java. Si vous spécifiez ou remplacez d'autres attributs, cela n'a aucun effet.

stanza	**Attribut**
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
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 WebSphere 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 WebSphere MQ classes for Java

Une application client IBM WebSphere 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 WebSphere 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 commandesIBM WebSphere MQ Script (MQSC) ou de commandes PCF (IBM WebSphere MQ Programmable Command Format), ou à l'aide de IBM WebSphere MQ Explorer.

Lorsque l'application crée un objet `MQQueueManager` , le client IBM WebSphere 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 aux classes IBM WebSphere 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 WebSphere MQ classes for Java émet un `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 203.

  Si votre application utilise le regroupement de connexions, voir «Contrôle du pool de connexions par défaut dans WebSphere MQ classes for Java», à la page 719.
- Lorsque les classes IBM WebSphere MQ classes for Java trouvent une définition de canal de connexion client appropriée dans la table de définition de canal du client, elles utilisent 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 SSL (Secure Sockets Layer):
  - Un canal MQI utilise SSL uniquement si la définition de canal extraite de la table de définition de canal du client spécifie le nom d'un CipherSpec pris en charge par les classes IBM WebSphere 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 WebSphere 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 (Online Certificate Status Protocol). Les classes IBM WebSphere MQ classes for Java ne peuvent pas utiliser les informations OCSP dans un fichier de table de définition de canal du client. Toutefois, vous pouvez configurer OCSP comme indiqué à la section Using Online Certificate Protocol.

  Pour plus d'informations sur l'utilisation de SSL avec une table de définition de canal du client, voir Spécification de l'utilisation de SSL par un canal MQI.

  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 WebSphere MQ classes for Java», à la page 712. Pour plus d'informations sur l'écriture d'un exit de canal dans d'autres langues, voir «Utilisation d'exits de canal non écrits en Java avec WebSphere MQ classes for Java», à la page 715.

Spécification d'une plage de ports pour les connexions client IBM WebSphere 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 WebSphere MQ classes for Java tente de se connecter à un gestionnaire de files d'attente IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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, où 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», à la page 1102.

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 WebSphere 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 obtenir des informations générales sur la fonction de lecture anticipée, voir la rubrique connexe.

Dans WebSphere MQ classes for Java, vous utilisez 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 des classes WebSphere MQ pour 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 les classes WebSphere MQ pour Java

Dans les classes WebSphere MQ pour Java, la rubrique est représentée par la classe MQTopic et vous la publiez à l'aide des méthodes MQTopic.put().

Pour obtenir des informations générales sur la messagerie de publication / abonnement WebSphere MQ , voir Introduction à la messagerie de publication / abonnement WebSphere MQ .

Gestion des en-têtes de message WebSphere MQ avec WebSphere MQ classes for Java

Des classes Java sont fournies pour représenter différents types d'en-tête de message. Deux classes auxiliaires sont également fournies.

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.

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. A l'instar de MQHeaderIterator, il analyse le contenu d'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 d'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 WebSphere 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 WebSphere 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 Websphere MQ couramment utilisés suivants

- MQRFH-Règles et en-tête de formatage
- MQRFH2 -Tout comme MQRFH, utilisé pour transmettre des messages à destination et en provenance d'un courtier de messages appartenant à WebSphere Message Broker. 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 WebSphere 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 WebSphere 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 de rebut à l'aide des classes WebSphere MQ 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 MQDLH d'un message de rebut à l'aide de WebSphere 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 des classes WebSphere MQ 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 WebSphere 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 WebSphere 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- en l'occurrence, la zone Rto dans le dossier jms, qui représente le nom de la file d'attente de réponses dans un message JMS MQ .

```
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 classes WebSphere MQ for Java

Ces exemples utilisent les classes d'en-tête pour analyser et manipuler le contenu de l'en-tête WebSphere 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 WebSphere MQ même si 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 de l'en-tête WebSphere 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 WebSphere MQ classes for Java

Vous pouvez créer des classes Java pour les types d'en-tête non fournis avec WebSphere 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 WebSphere MQ classes for Java, vous 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.
}
```

Gestion des messages PCF avec WebSphere MQ classes for Java

Des 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 WebSphere MQ classes for Java

Les appels de fonction pour traiter les descripteurs de message n'ont pas d'équivalent dans WebSphere MQ classes for Java. Pour définir, renvoyer ou supprimer des propriétés de descripteur de message, utilisez les méthodes de la classe MQMessage.

Pour obtenir des informations générales sur les propriétés de message, voir «Noms de propriétés», à la page 20.

Dans WebSphere MQ classes for Java, l'accès aux messages s'effectue 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 WebSphere MQ MQCRTMH, MQDLTMH, MQMHBUF et MQBUFMH

Pour définir les propriétés de descripteur de message dans l'interface de procédure, utilisez l'appel MQSETMP. Dans WebSphere 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 WebSphere 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 WebSphere MQ classes for Java, utilisez la méthode deleteProperty de la classe MQMessage.

Traitement des erreurs dans WebSphere MQ classes for Java

Traiter les erreurs provenant des classes WebSphere MQ pour 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 WebSphere MQ sont différents de zéro. Cela simplifie la logique du programme de sorte que vous n'ayez pas à vérifier les codes retour après chaque appel à WebSphere 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 WebSphere MQ renvoyés dans les exceptions Java pour z/OS sont documentés dans Codes raison pour z/OS et Codes raison pour toutes les autres plateformes.

Les exceptions émises lors de l'exécution d'une application WebSphere 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.

Par défaut, la classe MQException envoie automatiquement les exceptions à System.err, qui est généralement dirigé vers la console. Si vous souhaitez arrêter l'affichage des exceptions sur la console, incluez une ligne dans votre application pour définir MQException.log= null.

Obtention et définition de valeurs d'attribut dans WebSphere 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 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 de file d'attente, cet objet n'est *pas* automatiquement ouvert 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 qui se trouvent 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. WebSphere 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 client WebSphere MQ MQI basé sur C, cela entraînerait un problème, car le partage des descripteurs par plusieurs unités d'exécution est limité. WebSphere MQ classes for Java assouplit cette contrainte, ce qui permet 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 WebSphere MQ classes for Java garantit que, pour une connexion particulière (instance d'objetMQQueueManager), tous les accès au gestionnaire de files d'attente WebSphere 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 WebSphere MQ classes for Java

Présentation de l'utilisation des exits de canal dans une application à l'aide des classes WebSphere MQ pour 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 WebSphere 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 WebSphere MQ classes for Java, par exemple si vous utilisez une application client écrite en C.

Tout chiffrement SSL 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 vous souhaitez que WebSphere MQ interprète.

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 WebSphere 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 WebSphere MQ classes for Java

Vous pouvez affecter un exit de canal à l'aide de IBM WebSphere MQ classes for Java.

Il n'existe pas d'équivalent direct du canal IBM WebSphere MQ dans IBM WebSphere 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 nouveaux et anciens équivalents sont spécifiés (par exemple, la zone sendExit , qui ne peut être utilisée que pour le type d'exit utilisé dans les versions de IBM WebSphere MQ antérieures à la version 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 WebSphere 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 sur un serveur d'applications, vous devez stocker les fichiers dans le répertoire indiqué dans Tableau 88, à la page 715 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 88, à la page 715 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 88. Répertoire des programmes d'exit de canal	
Plateforme	**Répertoire**
AIX, HP-UX, Linuxet Solaris	`/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 WebSphere MQ lors de l'installation. Le répertoire par défaut est `C:\Program Files\IBM\WebSphere MQ`.	

Transmission de données aux exits de canal dans WebSphere 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 vous souhaitez que WebSphere MQ interprète.

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 WebSphere MQ classes for Java», à la page 697.

Utilisation d'exits de canal non écrits en Java avec WebSphere MQ classes for Java

Comment utiliser des programmes d'exit de canal écrits en C à partir d'une application Java.

Dans WebSphere MQ Version 7.0, 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 un autre langage.

Indiquez le nom du programme d'exit au format `library(function)` et assurez-vous que l'emplacement du programme d'exit est inclus dans la variable d'environnement de chemin.

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 413.

Utilisation de classes d'exit externes

Dans les versions de WebSphere MQ antérieures à la version 7.0, trois classes ont été fournies pour vous permettre d'utiliser des exits de canal écrits dans des langages autres que Java:

- L'exit MQExternalSecurity, qui implémente l'interface MQSecurityExit
- Exit MQExternalSend, qui implémente l'interface MQSendExit
- L'exit MQExternalReceive, qui implémente l'interface MQReceiveExit

L'utilisation de ces classes reste valide, mais la nouvelle méthode est préférée.

Pour utiliser un exit de sécurité qui n'est pas écrit en Java, une application a d'abord dû créer un objet d'exit MQExternalSecurity. L'application a spécifié, en tant que paramètres sur le constructeur d'exit MQExternalSecurity, le nom de la bibliothèque contenant l'exit de sécurité, le nom du point d'entrée de l'exit de sécurité et les données utilisateur à transmettre à l'exit de sécurité lorsqu'il a été appelé. Les programmes d'exit de canal qui ne sont pas écrits en Java ont été stockés dans le répertoire indiqué dans Tableau 88, à la page 715.

Utilisation d'une séquence d'exits d'émission ou de réception de canal dans WebSphere MQ classes for Java

Une application WebSphere 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 en 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 de WebSphere MQ antérieures à la version 7.0, deux classes ont été fournies pour permettre des 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 WebSphere 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 en 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 WebSphere 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 WebSphere MQ classes for Java utiliser la fonction de compression intégrée à IBM WebSphere MQ.

A l'aide de la fonction fournie avec IBM WebSphere 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 WebSphere MQ, voir Data compression (COMPMSG) et Header compression (COMPHDR).

Une application IBM WebSphere 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));
:
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 WebSphere MQ classes for Java

Plusieurs instances d'un canal MQI peuvent être créées pour partager une connexion TCP/IP unique.

Dans IBM WebSphere 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 WebSphere 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 WebSphere MQ classes for Java

WebSphere Les classes MQ pour Java permettent de mettre en pool les connexions de secours en vue de leur réutilisation.

WebSphere MQ classes for Java fournit une prise en charge supplémentaire pour les applications qui gèrent plusieurs connexions aux gestionnaires de files d'attente WebSphere 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.

WebSphere 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 WebSphere 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 WebSphere 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.

WebSphere Les classes MQ pour Java fournissent 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 la modifier (pour plus de détails, voir «Fourniture d'un pool de connexions différent dans WebSphere MQ classes for Java», à la page 721).

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 WebSphere 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 pool de connexions différent dans WebSphere 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(). Elle exécute un certain nombre d'unités d'exécution, où chaque unité d'exécution utilise WebSphere MQ indépendamment. Ces unités d'exécution utilisent myConnMan lorsqu'elles forgent des connexions.

Fourniture de vos propres classes ConnectionManager for WebSphere MQ for Java

WebSphere MQ classes for Java fournit une implémentation partielle de l'architecture de connecteur Java EE , permettant l'utilisation des implémentations de javax.resource.spi.ConnectionManager .

Les applications et les fournisseurs de middleware peuvent fournir des implémentations alternatives des pools de connexions. WebSphere MQ classes for Java fournit une implémentation partielle de l'architecture de connecteur Java EE . Les implémentations de **javax.resource.spi.ConnectionManager** peuvent être utilisées comme gestionnaire de connexions par défaut ou être spécifiées sur le constructeur MQQueueManager .

WebSphere Les classes MQ pour Java sont conformes au contrat de gestion des connexions de l'architecture Java EE Connector. Lisez cette section en conjonction avec le contrat de gestion des connexions de l'architecture de connecteur Java EE (voir le site Web Java de Sun à l'adresse https://java.sun.com).

L'interface ConnectionManager ne définit qu'une seule méthode:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

Le constructeur MQQueueManager appelle allocateConnection sur le ConnectionManagerapproprié. Il transmet les implémentations appropriées de la fabrique ManagedConnectionet des informations ConnectionRequesten tant que paramètres pour décrire la connexion requise.

ConnectionManager recherche dans son pool un objet javax.resource.spi.ManagedConnection qui a été créé avec des objets ManagedConnectionFactory et ConnectionRequestInfo identiques. Si ConnectionManager trouve des objets ManagedConnection appropriés, il crée un objet java.util.Set contenant le candidat ManagedConnections. Ensuite, ConnectionManager appelle ce qui suit:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

L'implémentation WebSphere MQ de la fabrique ManagedConnectionignore le paramètre subject. Cette méthode sélectionne et renvoie une ManagedConnection appropriée à partir de l'ensemble, ou renvoie la valeur null si elle ne trouve pas de ManagedConnectionappropriée. S'il n'existe pas de ManagedConnection appropriée dans le pool, ConnectionManager peut en créer une en utilisant:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

A nouveau, le paramètre subject est ignoré. Cette méthode se connecte à un gestionnaire de files d'attente WebSphere MQ et renvoie une implémentation de javax.resource.spi.ManagedConnection qui représente la nouvelle connexion. Une fois que le ConnectionManager a obtenu une ManagedConnection (provenant du pool ou nouvellement créée), il crée un descripteur de connexion à l'aide de:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

Ce descripteur de connexion peut être renvoyé à partir de allocateConnection().

Un ConnectionManager doit enregistrer un intérêt dans ManagedConnection via:

```
mc.addConnectionEventListener()
```

Le programme d'écoute ConnectionEventest averti si une erreur grave se produit sur la connexion ou lorsque MQQueueManager.disconnect () est appelée. Lorsque MQQueueManager.disconnect () est appelée, le programme d'écoute ConnectionEventpeut effectuer l'une des opérations suivantes:

- Réinitialisez ManagedConnection à l'aide de l'appel mc.cleanup(), puis renvoyez ManagedConnection au pool
- Destruction de ManagedConnection à l'aide de l'appel mc.destroy()

Si ConnectionManager est le ConnectionManagerpar défaut, il peut également enregistrer un intérêt dans l'état de l'ensemble géré par MQEnvironment de MQPoolTokens. Pour ce faire, commencez par construire un objet MQPoolServices , puis enregistrez un objet MQPoolServicesEventListener avec l'objet MQPoolServices :

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

Le programme d'écoute est averti lorsqu'un jeton MQPoolToken est ajouté ou supprimé de l'ensemble ou lorsque le ConnectionManager par défaut est modifié. L'objet MQPoolServices permet également d'interroger la taille actuelle de l'ensemble de MQPoolTokens.

Coordination JTA/JDBC à l'aide de WebSphere MQ classes for Java

WebSphere Les classes MQ pour Java prennent en charge la méthode MQQueueManager.begin (), qui permet à WebSphere 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 https://www.ibm.com/software/integration/wmq/requirements/.

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`.

V 7.5.0.7 Le `jdbcora12.dll` fourni fournit la compatibilité avec Oracle 12C, pour une installation de serveur IBM WebSphere MQ 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 vos classes IBM WebSphere MQ pour l'installation de Java . Vous devez déclarer la bibliothèque XA, également décrite comme fichier de commutation de chargement, au gestionnaire de files d'attente. Utilisez IBM WebSphere 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`

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, WebSphere 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 WebSphere 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 du chemin d'accès à la 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 WebSphere 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 admises sont db2 pour DB2 et `oracle` pour Oracle.

Voici un exemple de commande **make** :

```
make db2
```

Notez les points suivants :

- 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 différents répertoires WebSphere MQ . 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.

Après avoir créé les bibliothèques de commutation pour DB2, Oracleou les deux, 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 strophes XAResourceManager pour les bases de données DB2 ou Oracle , vous devez remplacer l'entrée SwitchFile dans chaque strophe par l'une des 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 déjà de strophes XAResourceManager pour DB2 ou Oracle , ou si vous souhaitez ajouter des strophes XAResourceManager supplémentaires, voir Administration 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

Il existe certains problèmes et limitations liés à la prise en charge de JTA/JDBC , certains dépendant du système de gestion de base de données utilisé.

Etant donné que cette prise en charge appelle les pilotes JDBC , l'implémentation de ces pilotes JDBC peut avoir un impact 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. **Toujours** : évitez d'arrêter brutalement une base de données alors que des applications y détiennent des connexions ouvertes.

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, voir la documentation DB2 .

La section XAResourceManager doit être configurée pour utiliser ThreadOfControl=PROCESS. Pour DB2 version 8.1 et les versions ultérieures, cela ne correspond pas au paramètre d'unité d'exécution de contrôle par défaut 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().

Prise en charge de SSL (Secure Sockets Layer) dans WebSphere MQ classes for Java

WebSphere Les classes MQ pour les applications client Java prennent en charge le chiffrement SSL (Secure Sockets Layer). Vous avez besoin d'un fournisseur JSSE pour utiliser le chiffrement SSL.

Les classes WebSphere MQ pour les applications client Java utilisant TRANSPORT (CLIENT) prennent en charge le chiffrement SSL (Secure Sockets Layer). SSL 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.

WebSphere MQ classes for Java utilise JSSE (Java Secure Socket Extension) pour gérer le chiffrement SSL 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 WebSphere 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 WebSphere MQ classes for Java», à la page 697.

Activation de SSL dans IBM WebSphere MQ classes for Java

Pour activer SSL, spécifiez une CipherSuite. Il existe deux façons de spécifier une CipherSuite.

SSL est pris en charge uniquement pour les connexions client. Pour activer SSL, 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 de chiffrement 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 et CipherSuites SSL dans WebSphere MQ classes for Java», à la page 731 contient une table mappant les CipherSpecs pris en charge par IBM WebSphere MQ à leurs CipherSuites équivalentes connues de JSSE.

Pour activer SSL, 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 SSL avec un CipherSpec de RC4_MD5_EXPORT:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Bien que le CipherSpec du canal soit RC4_MD5_EXPORT, l'application Java doit spécifier une CipherSuite de SSL_RSA_EXPORT_WITH_RC4_40_MD5. Voir «CipherSpecs et CipherSuites SSL dans WebSphere MQ classes for Java», à la page 731 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 WebSphere MQ classes for Java», à la page 697

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 WebSphere 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 SSL 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 UNIX, Linux et Windows

Utilisation du nom distinctif du gestionnaire de files d'attente dans IBM WebSphere MQ classes for Java

Le gestionnaire de files d'attente s'identifie à l'aide d'un certificat SSL, qui contient un nom distinctif (DN). Une application client IBM WebSphere 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 WebSphere MQ classes for Java

Indiquez les listes de révocation de certificat à utiliser via la classe java.security.cert.CertStore . IBM WebSphere MQ classes for Java puis vérifie 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 sur IBM WebSphere 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 WebSphere MQ classes for Java et WebSphere 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 WebSphere MQ classes for Java

Une application client WebSphere MQ classes for Java peut contrôler le moment où 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for Java Utilisez cette fabrique SSLSocketFactory pour vous connecter au gestionnaire de files d'attente IBM WebSphere 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 ou du magasin de clés de confiance JSSE dans WebSphere 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 l'emplacement du magasin de clés ou du magasin de clés de confiance, les classes WebSphere MQ pour les applications 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 SSL avec WebSphere MQ classes for Java

Un certain nombre de codes anomalie peuvent être émis par les classes WebSphere MQ pour Java lors de la connexion à un gestionnaire de files d'attente à l'aide de SSL.

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 SSL.

MQRC_JSSE_ERREUR
Le fournisseur JSSE a signalé une erreur qui n'a pas pu être traitée par WebSphere 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 SSL spécifiées, mais une erreur s'est produite lors de l'initialisation de l'environnement SSL.

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 WebSphere MQ est disponible dans «CipherSpecs et CipherSuites SSL dans WebSphere MQ classes for Java», à la page 731.

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 et CipherSuites SSL dans WebSphere MQ classes for Java

La possibilité pour une application IBM WebSphere MQ classes for Java d'établir une connexion à un gestionnaire de files d'attente dépend du CipherSpec spécifié à l'extrémité serveur du canal MQI et du CipherSuite spécifié à l'extrémité client.

Pour chaque combinaison de CipherSpec et CipherSuite, la capacité d'une application IBM WebSphere MQ classes for Java à se connecter à un gestionnaire de files d'attente dépend de la valeur de la

zone sslFipsRequired dans la classe MQEnvironment ou de la valeur de la propriété d'environnement CMQC.SSL_FIPS_REQUIRED_PROPERTY.

A l'extrémité serveur d'un canal MQI, le nom d'un CipherSpec peut être spécifié comme valeur du paramètre SSLCIPH dans une commande DEFINE CHANNEL CHLTYPE (SVRCONN). A l'extrémité client d'un canal MQI, une application IBM WebSphere MQ classes for Java peut définir la zone sslCipherSuite dans la classe MQEnvironment ou définir la propriété d'environnement CMQC.SSL_CIPHER_SUITE_PROPERTY.

Configuration de votre application pour l'utilisation des mappages IBM Java ou Oracle Java CipherSuite

A partir de IBM WebSphere MQ Version 7.5.0, groupe de correctifs 5, vous pouvez configurer si votre application utilise les mappages IBM Java CipherSuite à WebSphere MQ CipherSpec par défaut ou les mappages Oracle CipherSuite à WebSphere MQ CipherSpec . Par conséquent, vous pouvez utiliser les CipherSuites TLS que votre application utilise un environnement d'exécution Java IBM ou un environnement d'exécution Java Oracle . La propriété système Java `com.ibm.mq.cfg.useIBMCipherMappings` contrôle les mappages qui sont utilisés. La propriété peut avoir l'une des valeurs suivantes:

conforme
Utilisez les mappages IBM Java CipherSuite to WebSphere MQ CipherSpec .

Cette valeur est la valeur par défaut.

false
Utilisez les mappages Oracle CipherSuite à WebSphere MQ CipherSpec .

Le tableau suivant répertorie les CipherSpecs pris en charge par IBM WebSphere MQ et leurs CipherSuiteséquivalentes. La table indique également si une application IBM WebSphere MQ classes for Java peut établir une connexion à un gestionnaire de files d'attente si un CipherSpec est spécifié à l'extrémité serveur du canal MQI et que la suite de chiffrement CipherSuite équivalente est spécifiée à l'extrémité client.

Tableau 89. CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuites équivalentes

CipherSpec	Equivalent CipherSuite	Connexion possible si SFIPS [1] est défini sur YES?
NULL_MD5	SSL_RSA_WITH_NULL_MD5	Non
NULL_SHA	SSL_RSA_WITH_NULL_SHA	Non
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5 (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	Non
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (IBM JRE) SSL_RSA_EXPORT_WITH_RC4_40_MD5 (Oracle JRE)	Non

Tableau 89. CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuites équivalentes (suite)

CipherSpec	**Equivalent CipherSuite**	**Connexion possible si SFIPS [1] est défini sur YES?**
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256 (IBM JRE) TLS_RSA_WITH_NULL_SHA256 (Oracle JRE)	Non [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA (Oracle JRE)	Oui [5] [7]
TLS_RSA_WITH_AES_128_CBC_SHA256	SSL_RSA_WITH_AES_128_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_128_CBC_SHA256 (Oracle JRE)	Oui [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA (Oracle JRE)	Oui [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256 (IBM JRE) TLS_RSA_WITH_AES_256_CBC_SHA256 (Oracle JRE)	Oui [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8]	SSL_RSA_WITH_DES_CBC_SHA	Non[3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8] [9]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Oui
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non[4]

Tableau 89. CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuites équivalentes (suite)		
CipherSpec	**Equivalent CipherSuite**	**Connexion possible si SFIPS [1] est défini sur YES?**
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (IBM JRE) Aucun équivalent pour l'environnement d'exécution Java Oracle .	Non[6]

Remarques :

1. Dans une application IBM WebSphere MQ classes for Java , indiquez que seuls les algorithmes certifiés FIPS doivent être utilisés en définissant la zone sslFipsRequired dans la classe MQEnvironment sur `true` et indiquez que des algorithmes non certifiés FIPS peuvent également être utilisés en définissant la zone sslFipsRequired sur `false`. Vous pouvez également définir la propriété d'environnement CMQC.SSL_FIPS_REQUIRED_PROPERTY.
2. Ce CipherSpec n'a pas de CipherSuiteéquivalente.
3. Ce CipherSpec a été certifié FIPS 140-2 avant le 19th mai 2007.
4. Ce CipherSpec a été certifié FIPS 140-2 avant le 19th mai 2007. Le nom FIPS_WITH_DES_CBC_SHA est historique et reflète le fait que ce CipherSpec était auparavant conforme à FIPS (mais ne l'est plus). Ce CipherSpec est déprécié et son utilisation est déconseillée.
5. Ces CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) ne peuvent pas être utilisés pour sécuriser une connexion de WebSphere MQ Explorer à un gestionnaire de files d'attente, sauf si les fichiers de règles sans restriction appropriés sont appliqués à l'environnement d'exécution Java utilisé par l'explorateur.

   Pour plus d'informations sur les fichiers de règles, voir Informations sur la sécurité .
6. Le nom FIPS_WITH_3DES_EDE_CBC_SHA est historique et reflète le fait que ce CipherSpec était auparavant conforme à FIPS (mais ne l'est plus). Ce CipherSpec est déprécié et son utilisation est déconseillée.
7. Ces CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) requièrent des environnements d'exécution Java IBM 6.0 SR13 FP2 , 7.0 SR4 FP2 ou version ultérieure.
8. Ces CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) peuvent utiliser SSLv3 ou TLS. Par défaut, lorsque FIPS n'est pas activé, SSLv3 est utilisé. Pour utiliser TLS, définissez la propriété système Java **com.ibm.mq.cfg.preferTLS** sur `true`.
9. 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.

Information associée

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client

FIPS (Federal Information Processing Standards) pour UNIX, Linux et Windows

Blogue MQdev: MQ Java, TLS Ciphers, Non-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837

Blogue MQdev: Relation entre MQ CipherSpecs et Java Cipher Suites

Exécution des classes WebSphere MQ pour les applications 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 classes WebSphere MQ pour les bibliothèques Java (voir WebSphere MQ classes pour les bibliothèques Java).

WebSphere MQ classes for Java-comportement dépendant de l'environnement

Les classes WebSphere MQ pour Java vous permettent de créer des applications pouvant s'exécuter sur différentes versions de WebSphere MQ. Cette collection de rubriques décrit le comportement des classes Java qui dépendent de ces différentes versions.

WebSphere Les classes MQ pour Java fournissent 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 le document Application Programming Reference correspondant au gestionnaire de files d'attente.

Classes principales dans WebSphere MQ classes for Java

WebSphere Les classes MQ pour Java contiennent 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 variations pour les classes principales de WebSphere MQ classes for Java», à la page 736.

- 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
  – position
  – 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 variations pour les classes principales de WebSphere MQ classes for Java», à la page 736 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. (Voir «Options de connexion pour WebSphere MQ classes for Java», à la page 678.)

Restrictions et variations pour les classes principales de WebSphere 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 le même que si un gestionnaire de files d'attente Windows, UNIX ou Linux WebSphere MQ est utilisé, à l'exception des restrictions et des variations mineures suivantes.

*Restrictions relatives aux valeurs MQGMO_ * dans WebSphere 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 WebSphere 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 WebSphere 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 variantes des valeurs MQCNO_ * dans WebSphere 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 les classes WebSphere MQ pour 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 WebSphere 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.

Fonctions en dehors des classes principales de WebSphere MQ classes for Java

Les classes WebSphere MQ pour 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 pour un gestionnaire de files d'attente WebSphere MQ sur les systèmes UNIX, Linuxou Windows en mode liaison. Sinon, il échoue avec MQRC_ENVIRONMENT_ERROR.

Pour plus d'informations, voir «Coordination JTA/JDBC à l'aide de WebSphere MQ classes for Java», à la page 723.

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 WebSphere 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 WebSphere 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 des 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 WebSphere 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 WebSphere MQ .

Notez que les classes JMS WebSphere MQ ne sont pas prises en charge pour une utilisation dans une application Java CICS.

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.

Exécution des classes IBM WebSphere MQ pour les applications Java sur la plateforme Java Enterprise Edition
Certaines restrictions et considérations de conception doivent être prises en compte avant l'utilisation des classes IBM WebSphere MQ pour Java dans Java EE

Les classes IBM WebSphere MQ pour Java sont soumises à des restrictions lorsqu'elles sont utilisées dans un environnement Java EE . D'autres considérations doivent également être prises en compte lors de la conception, de l'implémentation et de la gestion d'une application IBM WebSphere MQ pour 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 des classes IBM WebSphere MQ pour Java est IBM WebSphere MQ lui-même. Bien qu'une application sous contrôle JTA

puisse utiliser des classes IBM WebSphere MQ pour 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 WebSphere MQ dans les transactions coordonnées par le serveur d'applications, vous devez utiliser à la place les classes IBM WebSphere MQ pour JMS.

Création d'unités d'exécution

IBM WebSphere 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 WebSphere 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 des classes IBM WebSphere MQ pour Java, il se peut que le serveur d'applications ne puisse pas faire la distinction entre le code d'application et le code IBM WebSphere MQ pour Java . Par conséquent, les unités d'exécution précédemment décrites entraînent la non-conformité de l'application avec la spécification de conteneur. IBM WebSphere MQ classes for JMS ne rompt 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 les classes IBM WebSphere MQ pour l'API 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, les classes IBM WebSphere MQ pour Java peuvent enfreindre les règles d'unités d'exécution de 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 des classes IBM WebSphere MQ pour 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 des classes IBM WebSphere MQ pour Java sont antérieures à l'environnement Java EE . IBM WebSphere MQ Les classes pour Java peuvent être utilisées 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 SSL 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 via la création d'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 à l'aide de classes IBM WebSphere MQ pour Java dans le même processus JVM, et il n'est pas possible de remplacer ce comportement pour des applications particulières. Quelques exemples :
  - configuration des propriétés SSL, 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 .
- SSL est configuré pour les applications utilisant des classes IBM WebSphere MQ pour 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 les classes IBM WebSphere MQ pour Java de manière appropriée pour fournir le niveau de sécurité requis et que vous n'utilisez pas la configuration du serveur d'applications.

Restrictions du mode de liaison

IBM WebSphere 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 WebSphere MQ (RA) fourni dans WebSphere Application Server soient différentes. Par exemple, WebSphere Application Server Version 7.0, qui fournit un niveau IBM WebSphere MQ RA de 7.0.1, peut être installé sur la même machine qu'un gestionnaire de files d'attente Version 6.0 .

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.

Utilisation des classes WebSphere MQ pour JMS

WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS) est le fournisseur JMS fourni avec WebSphere MQ. Outre l'implémentation des interfaces définies dans le package javax.jms , WebSphere MQ classes for JMS fournit deux ensembles d'extensions à l'API JMS.

La spécification JMS définit un ensemble d'interfaces que les applications peuvent utiliser pour effectuer des opérations de messagerie. Le package javax.jms définit les interfaces JMS et un fournisseur JMS implémente ces interfaces pour un produit de messagerie spécifique. WebSphere MQ version 7.5 utilise actuellement la spécification JMS 1.1 . WebSphere MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour WebSphere MQ.

La spécification JMS s'attend 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 extrait ces objets à l'aide de l'interface JNDI (Java Naming and Directory Interface). WebSphere MQ classes for JMS prend en charge l'utilisation d'objets gérés et un administrateur peut utiliser l'outil d'administration JMS WebSphere MQ ou WebSphere MQ Explorer pour créer et gérer des objets gérés.

WebSphere 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 JMS WebSphere MQ
Les éditions précédentes de WebSphere MQ classes for JMS contiennent des extensions qui sont implémentées dans des objets tels que MQConnectionFactory, MQQueue et MQTopic. Ces objets possèdent des propriétés et des méthodes spécifiques à WebSphere MQ. Les objets peuvent être des objets administrés ou une application peut créer les objets de façon dynamique lors de la phase d'exécution. Cette édition de WebSphere MQ classes for JMS gère ces extensions, qui sont désormais appelées WebSphere MQ extensions JMS. Vous pouvez continuer à utiliser, sans modification, toutes les applications qui utilisent ces extensions.

Les extensions JMS IBM
Cette édition de WebSphere MQ classes for JMS fournit un ensemble plus générique d'extensions à l'API JMS, qui ne sont pas spécifiques à WebSphere MQ en tant que système de messagerie. Ces extensions sont appelées extensions JMS IBM et ont les objectifs généraux suivants:

- Pour fournir un niveau de cohérence plus élevé entre les fournisseurs JMS IBM
- 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 JMS IBM à un autre

Les extensions fournissent une fonction similaire à celle fournie dans Message Service Client for C/C++ et Message Service Client for .NET.

Pourquoi utiliser WebSphere MQ classes for JMS?

L'utilisation de WebSphere MQ classes for JMS présente les avantages suivants:

- Vous pouvez réutiliser des compétences JMS.

  WebSphere MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour WebSphere MQ en tant que système de messagerie. Si votre organisation est nouvelle dans WebSphere 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 WebSphere MQ plutôt qu'à l'une des autres API fournies avec WebSphere MQ.

- JMS fait 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 JavaServer (JSP), 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.

- Un administrateur peut créer et gérer des objets gérés par JMS dans un référentiel central et les classes WebSphere MQ pour les applications JMS peuvent extraire ces objets à l'aide de l'interface JNDI (Java Naming and Directory Interface).

  Les fabriques de connexions et les destinations JMS encapsulent des informations spécifiques à WebSphere 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. Cet arrangement offre donc à l'application une certaine indépendance par rapport à la configuration WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 de pont à l'aide de 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.

Initiation à WebSphere MQ classes for JMS

Cette rubrique fournit une présentation de WebSphere MQ classes for JMS et vous indique ce que vous devez savoir avant d'utiliser WebSphere MQ classes for JMS.

Prérequis pour WebSphere MQ classes for JMS

Pour développer et exécuter des classes WebSphere MQ pour les applications JMS, vous avez besoin de certains composants logiciels comme prérequis.

Pour obtenir les informations les plus récentes sur les prérequis pour WebSphere MQ classes for JMS, voir le fichier Readme WebSphere MQ .

Pour développer des classes WebSphere MQ pour les applications JMS, vous avez besoin d'un kit de développement de logiciels (SDK) Java 2. Les détails des JDK pris en charge avec votre système d'exploitation sont disponibles sur la page WebSphere MQ System requirements. Voir WebSphere MQ Requirements.

Pour exécuter des classes WebSphere MQ pour les applications JMS, vous avez besoin des composants logiciels suivants:

- Un gestionnaire de files d'attente WebSphere MQ
- Un environnement d'exécution Java (JRE), pour chaque système sur lequel vous exécutez des applications

Si vous avez besoin de connexions SSL pour utiliser des modules cryptographiques certifiés FIPS 140-2, vous avez besoin du fournisseur IBM Java JSSE FIPS (IBMJSSEFIPS). Chaque kit de développement de logiciels IBM Java 2 SDK et JRE version 5 ou ultérieure contient IBMJSSEFIPS.

Vous pouvez utiliser des adresses Internet Protocol version 6 (IPv6) dans vos WebSphere MQ pour les applications JMS à condition que les adresses IPv6 soient prises en charge par votre machine virtuelle Java (JVM) et l'implémentation TCP/IP sur votre système d'exploitation. L'outil d'administration JMS WebSphere MQ (voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965) accepte également les adresses IPv6 .

L'outil d'administration JMS WebSphere MQ et WebSphere MQ Explorer utilisent l'interface JNDI (Java Naming and Directory Interface) pour accéder à un service d'annuaire qui stocke les objets gérés. Les applications WebSphere 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 à des appels au service d'annuaire. Les fournisseurs de services suivants sont fournis avec WebSphere MQ classes for JMS:

- Un fournisseur de services LDAP (Lightweight Directory Access Protocol) dans les fichiers ldap.jar et providerutil.jar. Le fournisseur de services LDAP permet d'accéder à un service d'annuaire basé sur un serveur LDAP.
- Un fournisseur de services de système de fichiers dans les fichiers fscontext.jar et providerutil.jar. 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.

Préparation de programmes JMS pour le client IBM WebSphere MQ for HP Integrity NonStop Server

Cette rubrique explique ce que vous devez savoir avant de développer et d'exécuter des programmes JMS pour le client IBM WebSphere MQ for HP Integrity NonStop Server.

Les classes IBM WebSphere MQ pour JMS sont installées dans le cadre de l'installation du client IBM WebSphere MQ pour HP Integrity NonStop Server . Pour plus de détails sur un récapitulatif du contenu de l'installation, voir Système de fichiers.

Certains aspects de la fonctionnalité client sont spécifiques au système d'exploitation hôte. Pour plus d'informations sur les fonctions prises en charge pour le client IBM WebSphere MQ pour HP Integrity NonStop Server, voir IBM WebSphere MQ client for HP Integrity NonStop Server supported environments and features.

Prérequis

Pour générer et exécuter des applications JMS, le composant *HP Integrity NonStop Server for Java* doit être installé et disponible.

Configuration

Pour plus d'informations sur la configuration de l'environnement pour l'exécution et la génération d'applications dans lesquelles vous pouvez utiliser les classes IBM WebSphere MQ pour JMS, voir «Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749.

Pour plus d'informations sur les étapes requises pour configurer un gestionnaire de files d'attente afin qu'il accepte les connexions à partir des applications client, voir «Configuration post-installation pour les classes WebSphere MQ pour les applications JMS», à la page 799.

Pour plus d'informations sur la validation de votre environnement IBM WebSphere MQ classes for JMS, voir «Test de vérification de l'installation point à point pour WebSphere MQ classes for JMS», à la page 802.

Applications d'écriture

Pour plus d'informations sur l'écriture d'applications JMS, voir «Ecriture des classes WebSphere MQ pour les applications JMS», à la page 833.

Pour plus d'informations sur l'utilisation de l'outil d'administration JMS IBM WebSphere MQ , voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965.

Echantillons

Des exemples d'application sont fournis dans le sous-répertoire suivant de l'installation: `opt/mqm/samp/jms`.

Pour plus d'informations sur les étapes de configuration requises pour exécuter les exemples, voir «Préparation et exécution des exemples de programmes», à la page 115.

Résolution des problèmes

Pour plus d'informations sur la résolution des problèmes, voir «Résolution des problèmes liés aux classes IBM WebSphere MQ pour JMS», à la page 823.

Installation et configuration des classes WebSphere MQ pour JMS

Cette section décrit les répertoires et les fichiers créés lors de l'installation de WebSphere MQ classes for JMS et explique comment configurer WebSphere MQ classes for JMS après l'installation.

Concepts associés

«Ce qui est installé pour IBM WebSphere MQ classes for JMS», à la page 746
Un certain nombre de fichiers et de répertoires sont créés lorsque vous installez les classes IBM WebSphere MQ pour JMS. Sous Windows , une partie de la configuration est effectuée lors de l'installation en définissant automatiquement des variables d'environnement. Sur les autres plateformes et dans certains environnements Windows, vous devez définir des variables d'environnement avant de pouvoir exécuter des classes IBM WebSphere MQ pour les applications JMS.

«Exécution des classes WebSphere MQ pour les applications JMS sous le gestionnaire de sécurité Java», à la page 756
WebSphere MQ classes for JMS peut s'exécuter avec le gestionnaire de sécurité Java activé. Pour exécuter des applications avec le gestionnaire de sécurité activé, vous devez configurer votre machine virtuelle Java (JVM) avec un fichier de configuration de règles approprié.

«L'adaptateur de ressources IBM WebSphere MQ», à la page 760
L'adaptateur de ressources permet aux applications exécutées dans un serveur d'applications d'accéder aux ressources IBM WebSphere MQ . Il prend en charge les communications entrantes et sortantes.

«Configuration post-installation pour les classes WebSphere MQ pour les applications JMS», à la page 799
Cette rubrique vous indique les droits dont ont besoin les applications WebSphere MQ 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.

«Test de vérification de l'installation point à point pour WebSphere MQ classes for JMS», à la page 802
Un programme de test de vérification d'installation point à point (IVT) est fourni avec WebSphere MQ classes for JMS. 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.

«Test de vérification de l'installation de publication / abonnement pour WebSphere MQ classes for JMS», à la page 806
Un programme de test de vérification de l'installation (IVT) de publication / abonnement est fourni avec WebSphere 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.

«Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810
Le programme IVT est fourni sous la forme d'un fichier EAR. Pour utiliser le programme, vous devez le déployer et définir des objets en tant que ressources JCA.

«Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 779
Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

«Prise en charge d'OSGi», à la page 822
OSGi fournit une infrastructure qui prend en charge le déploiement d'applications en tant que bundles. Neuf bundles OSGi sont fournis avec IBM WebSphere MQ classes for JMS .

«Résolution des problèmes liés aux classes IBM WebSphere MQ pour JMS», à la page 823

Vous pouvez examiner les problèmes en exécutant les programmes de vérification de l'installation et en utilisant les fonctions de trace et de journal.

Tâches associées

«Installation et test de l'adaptateur de ressources MQ dans WAS CE», à la page 812
Installation de l'adaptateur de ressources IBM WebSphere MQ et exécution de l'application de test de vérification de l'installation (IVT) dans WebSphere Application Server CE.

«Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé», à la page 814
Si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou un mode de liaison différent de celui du client, vous devez modifier l'application IVT et les scripts associés dans WebSphere Application Server CE avant de déployer l'adaptateur de ressources ou l'application IVT.

«Déploiement de l'application IVT dans JBoss avec un environnement IBM WebSphere MQ personnalisé», à la page 817
Lors de l'installation de l'adaptateur de ressources IBM WebSphere MQ dans JBoss, si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou utiliser un mode de liaison différent à la place du mode client, vous devez d'abord modifier l'application IVT et les scripts associés dans JBoss avant de déployer l'adaptateur de ressources ou l'application IVT.

Identification des incidents pour l'adaptateur de ressources IBM WebSphere MQ

Référence associée

«Scripts fournis avec WebSphere MQ classes for JMS», à la page 821
Un certain nombre de scripts sont fournis pour vous aider à exécuter les tâches courantes qui doivent être effectuées lors de l'utilisation des classes WebSphere MQ pour JMS.

Ce qui est installé pour IBM WebSphere MQ classes for JMS

Un certain nombre de fichiers et de répertoires sont créés lorsque vous installez les classes IBM WebSphere MQ pour JMS. Sous Windows , une partie de la configuration est effectuée lors de l'installation en définissant automatiquement des variables d'environnement. Sur les autres plateformes et dans certains environnements Windows, vous devez définir des variables d'environnement avant de pouvoir exécuter des classes IBM WebSphere MQ pour les applications JMS.

Pour la plupart des systèmes d'exploitation, les classes IBM WebSphere MQ pour JMS sont installées en tant que composant facultatif lorsque vous installez IBM WebSphere MQ. Pour le client IBM WebSphere MQ pour HP Integrity NonStop Server, les classes IBM WebSphere MQ pour JMS sont installées par défaut. Pour plus d'informations sur l'installation de IBM WebSphere MQ, voir:

Installation d'un serveur WebSphere MQ
Installation d'un client IBM WebSphere MQ

Tableau 90, à la page 746 indique où les fichiers IBM WebSphere MQ classes for JMS sont installés sur chaque plateforme.

Tableau 90. Répertoires d'installation de IBM WebSphere MQ classes for JMS

Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/java
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/java
HP-UX, Linuxet Solaris	*MQ_INSTALLATION_PATH*/java
Windows	*MQ_INSTALLATION_PATH*\java
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.	

Le répertoire d'installation comprend:

- Les classes IBM WebSphere MQ pour les fichiers JAR JMS, qui se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\lib.
- Les bibliothèques natives IBM WebSphere 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 WebSphere MQ , voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 751.
- Scripts supplémentaires décrits dans «Scripts fournis avec WebSphere MQ classes for JMS», à la page 821. Ces scripts se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\bin.
- Spécifications de l'API IBM WebSphere MQ classes for JMS. L'outil Javadoc a été utilisé pour générer les pages HTML contenant les spécifications de l'API.

  Les pages HTML se trouvent dans le répertoire *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses.

  Sur les systèmes UNIX, Linuxet Windows, ce sous-répertoire contient les pages HTML individuelles.
- Prise en charge de OGSi. Les bundles OSGi sont installés dans le répertoire java \lib\OSGi et décrits dans «Prise en charge d'OSGi», à la page 822.
- L'adaptateur de ressources IBM WebSphere MQ , qui peut être déployé sur n'importe quel serveur d'applications compatible JCA 1.5 (ou version ultérieure).

  L'adaptateur de ressources IBM WebSphere MQ se trouve dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\jca ; pour plus d'informations, voir «L'adaptateur de ressources IBM WebSphere MQ», à la page 760
- Sous Windows, les symboles pouvant être utilisés pour le débogage sont installés dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\symboles.

Le répertoire d'installation inclut également certains fichiers appartenant à d'autres composants IBM WebSphere MQ . Ces répertoires sont les suivants:

- Le transport IBM WebSphere MQ pour SOAP, qui fournit un transport JMS pour SOAP, est installé dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\soap. Pour plus d'informations sur le transport IBM WebSphere MQ pour SOAP, voir la section du centre de documentation décrivant «Transport WebSphere MQ pour SOAP», à la page 978.
- Sur les plateformes réparties, IBM WebSphere MQ Bridge for HTTP est installé dans le répertoire *MQ_INSTALLATION_PATH*\java\lib\http. Pour plus d'informations sur le pont IBM WebSphere MQ pour HTTP, voir la section du centre de documentation décrivant «WebSphere MQ Bridge for HTTP», à la page 1055

Certains modèles d'application sont fournis avec IBM WebSphere MQ classes for JMS. Le Tableau 91, à la page 747 indique où les modèles d'application sont installés sur chaque plateforme.

Tableau 91. Répertoires d'exemples

Plateforme	**Répertoire**
AIX	*MQ_INSTALLATION_PATH*/samp/jms
HP Integrity NonStop Server	*MQ_INSTALLATION_PATH*/opt/mqm/samp/jms
HP-UX, Linuxet Solaris	*MQ_INSTALLATION_PATH*/samp/jms
Windows	*MQ_INSTALLATION_PATH*\tools\jms
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.	

Après l'installation, vous devrez peut-être effectuer certaines tâches de configuration pour compiler et exécuter des applications.

«Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749 décrit le chemin d'accès aux classes requis pour exécuter des classes IBM WebSphere MQ simples pour les applications JMS. Cette rubrique décrit également les fichiers JAR supplémentaires qui doivent être référencés dans des circonstances spéciales et les variables d'environnement que vous devez définir pour exécuter les scripts fournis avec IBM WebSphere MQ classes for JMS.

Si vous avez besoin que votre application IBM WebSphere MQ classes for JMS soit liée à du code écrit dans des langages autres que Java (par exemple, pour utiliser le transport de liaisons lors de la connexion à un gestionnaire de files d'attente), «Configuration des bibliothèques JNI (Java Native Interface)», à la page 751 explique où trouver l'emplacement des bibliothèques JNI (Java Native Interface) à spécifier en tant que paramètre de la commande Java.

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 WebSphere MQ classes for JMS est décrit dans «Le fichier de configuration IBM WebSphere MQ classes for JMS», à la page 753.

Installation et mise à niveau des fichiers JAR WebSphere MQ classes for JMS

Le seul moyen pris en charge pour obtenir les fichiers JAR IBM WebSphere MQ classes for JMS sur un système consiste à installer le produit IBM WebSphere MQ ou le WebSphere MQ V7.5 Clients SupportPac- MQC75, ou à utiliser un outil de gestion de logiciels tel que Apache Maven, pour plus d'informations, voir «IBM WebSphere MQ classes for JMS et outils de gestion de logiciels», à la page 755.

Ne déplacez pas ou ne copiez pas les fichiers JAR IBM WebSphere MQ classes for JMS ou les bibliothèques natives vers d'autres machines ou vers un autre emplacement sur une machine où les classes IBM WebSphere MQ for JMS ont été installées, sauf si vous utilisez un outil de gestion de logiciels.

- Les groupes de correctifs ne peuvent pas être appliqués à une "installation" dans laquelle des fichiers JAR ont été copiés à partir d'une autre machine, car cela rend beaucoup plus difficile de s'assurer que tous les fichiers JAR sont conservés en même temps que les autres et qu'ils sont à des niveaux compatibles.
- La copie des fichiers JAR IBM WebSphere MQ classes for JMS entre les machines peut également générer plusieurs copies des fichiers résidant sur la même machine, ce qui peut entraîner des problèmes de maintenance du code et des problèmes de débogage.
- La commande dspmqver , utilisée pour afficher les informations de version d'une installation IBM WebSphere MQ , affiche uniquement les informations de version des classes IBM WebSphere MQ pour JMS installées dans le répertoire `\java\lib` .

  Si plusieurs copies des fichiers se trouvent sur la même machine, l'exécution de **dspmqver** risque de ne pas fournir des informations précises sur la version des classes IBM WebSphere MQ for JMS utilisées par une application.

N'incluez pas les fichiers JAR IBM WebSphere MQ classes for JMS dans les archives d'application (telles que les archives d'application d'entreprise ou les fichiers EAR).

- Les mises à jour des classes IBM WebSphere MQ pour JMS ne peuvent pas être appliquées à l'aide d'un groupe de correctifs IBM WebSphere MQ .
- Le support IBM ne peut pas déterminer facilement la version des classes IBM WebSphere MQ for JMS utilisées par l'application.
- Des problèmes peuvent se produire si plusieurs applications s'exécutant dans le même environnement d'exécution Java incluent des versions différentes des classes IBM WebSphere MQ pour JMS, car plusieurs versions des classes IBM WebSphere MQ pour JMS sont chargées dans l'environnement d'exécution Java en même temps.

  Les exceptions suivantes sont des exemples de ces problèmes:

```
java.lang.ClassCastException :
com.ibm.mq.jmqi.system.JmqiSystemEnvironment incompatible with
com.ibm.mq.jmqi.system.JmqiSystemEnvironment

java.lang.ClassCastException :
com.ibm.mq.jms.MQQueue incompatible with com.ibm.mq.jms.MQQueue
```

- Si une application utilise le transport BINDINGS pour se connecter à un gestionnaire de files d'attente, toute mise à niveau majeure du gestionnaire de files d'attente nécessite également la mise à jour de l'application pour inclure le niveau correspondant des classes IBM WebSphere MQ pour JMS.

  Par exemple, si un gestionnaire de files d'attente est mis à niveau vers le niveau IBM WebSphere MQ version 7.5 , toutes les applications qui se connectent au gestionnaire de files d'attente à l'aide du transport BINDINGS doivent également être mises à jour pour inclure les classes IBM WebSphere MQ version 7.5 pour JMS.

Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS

Avant de pouvoir compiler et exécuter des applications IBM WebSphere MQ classes for JMS, le paramètre de votre variable d'environnement CLASSPATH doit inclure le fichier JAR (Java archive) IBM WebSphere MQ classes for JMS. 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 WebSphere MQ classes for JMS, d'autres variables d'environnement doivent être définies.

Pour compiler et exécuter des classes IBM WebSphere MQ pour des applications JMS, utilisez le paramètre CLASSPATH pour votre plateforme, comme illustré dans la Tableau 92, à la page 749. Le paramètre inclut le répertoire d'exemples, afin que vous puissiez compiler et exécuter les classes IBM WebSphere MQ pour les modèles d'application JMS. Vous pouvez également spécifier le chemin d'accès aux classes dans la commande **java** au lieu d'utiliser la variable d'environnement.

Tableau 92. Paramètre CLASSPATH pour compiler et exécuter les classes IBM WebSphere MQ pour les applications JMS, y compris les modèles d'application

Plateforme	**Paramètre CLASSPATH**
AIX	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP Integrity NonStop Server	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
HP-UX, Linuxet Solaris	CLASSPATH=*MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/samp/jms:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:
Fenêtres	CLASSPATH=*MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms;
z/OS	CLASSPATH=*MQ_INSTALLATION_PATH*/mqm/V7R0M0/java/lib/ com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V6R0M0/java/samples/jms:
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.	

Le manifeste du fichier JAR com.ibm.mqjms.jar contient des références à la plupart des autres fichiers JAR requis par les classes IBM WebSphere MQ pour les applications 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 and Directory Interface) pour extraire des 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 des classes IBM WebSphere MQ pour Java, com.ibm.mq.jar, à votre chemin d'accès aux classes.
- Si vous compilez votre code Java à l'aide d'un kit SDK (Software Development Kit) Java 2 version 1.4.2, vous devez ajouter les fichiers JAR suivants à votre chemin d'accès aux classes:
  - jms.jar
  - com.ibm.mq.jmqi.jar

  De plus, 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
  - jndi.jar
  - ldap.jar
  - providerutil.jar

  Et si votre application utilise JTA, vous devez également ajouter jta.jar à votre chemin d'accès aux classes.

  Notez que ces fichiers JAR supplémentaires sont requis uniquement pour la compilation de vos applications et non pour leur exécution.

Si vous effectuez la compilation à l'aide de l'option -Xlint, il se peut qu'un message vous avertissant 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 Extended Security Edition.

Les scripts fournis avec IBM WebSphere 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 WebSphere MQ classes for JMS est installé.

MQ_JAVA_LIB_PATH
Cette variable d'environnement indique le répertoire dans lequel sont stockées les bibliothèques WebSphere MQ pour JMS, comme illustré dans la Tableau 93, à la page 751.

Sous Windows, toutes les variables d'environnement sont définies automatiquement lors de l'installation. Sur toute autre plateforme, vous devez les définir vous-même.

Pour définir les variables d'environnement si vous utilisez une machine virtuelle Java 32 bits sur des systèmes UNIX, HP Integrity NonStop Server, ou Linux , vous pouvez utiliser le script setjmsenv. Pour définir les variables d'environnement si vous utilisez une machine virtuelle Java 64 bits sur un système UNIX ou Linux , vous pouvez utiliser le script setjmsenv64. 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 WebSphere MQ est installé.

Vous pouvez utiliser le script setjmsenv ou setjmsenv64 de différentes manières: vous pouvez l'utiliser 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 en entrant `. ./setjmsenv` ou `. ./setjmsenv64`

Configuration des bibliothèques JNI (Java Native Interface)

Les applications IBM WebSphere 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 les programmes d'exit de canal écrits dans des langages autres que Java doivent être exécutées dans un environnement qui accède aux bibliothèques JNI (Java Native Interface).

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 la machine virtuelle Java (JVM) puisse charger la bibliothèque mqjbnd avant de démarrer l'application IBM WebSphere MQ classes for JMS.

IBM WebSphere 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 les classes IBM WebSphere MQ pour JMS et le gestionnaire de files d'attente. La bibliothèque mqjbnd installée avec IBM WebSphere MQ version 7.5 peut être utilisée pour se connecter à n'importe quel gestionnaire de files d'attente IBM WebSphere MQ version 7.5 (ou antérieure).

mqjexitstub02
La bibliothèque mqjexitstub02 est chargée par les classes IBM WebSphere MQ pour 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 WebSphere 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 93. Emplacement des bibliothèques IBM WebSphere MQ classes for JMS pour chaque plateforme

Plateforme	**Répertoire contenant les bibliothèques IBM WebSphere MQ classes for JMS**
AIX	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
HP-UX	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Linux (PUISSANCE , x86-64 et zSeries s390x (plateformes)	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)

Tableau 93. Emplacement des bibliothèques IBM WebSphere MQ classes for JMS pour chaque plateforme (suite)

Plateforme	**Répertoire contenant les bibliothèques IBM WebSphere MQ classes for JMS**
Linux (plateformex86) Linux (Plateforme zSeries)	*MQ_INSTALLATION_PATH* /java/lib
Solaris (plateformesx86-64 et SPARC)	*MQ_INSTALLATION_PATH*/java/lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*/java/lib64 (bibliothèques 64 bits)
Windows	*MQ_INSTALLATION_PATH*\java\lib (bibliothèques 32 bits) *MQ_INSTALLATION_PATH*\java\lib64 (bibliothèques 64 bits)
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel IBM WebSphere MQ est installé.	

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>
     ```

     Linux 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 WebSphere MQ. Par exemple, pour une machine virtuelle Java 64 bits et un emplacement d'installation IBM WebSphere MQ par défaut, vous pouvez utiliser les paramètres suivants:

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

     Solaris HP-UX Linux
     ```
     export LD_LIBRARY_PATH=/opt/mqm/java/
     lib64:$LD_LIBRARY_PATH
     ```

     Windows
     ```
     set PATH=C:\Program Files\IBM\MQ\java\lib64;%PATH%
     ```

     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 les classes IBM WebSphere MQ pour l'application JMS à l'aide de la commande suivante:

   ```
   java application-name
   ```

   où *application-name* est le nom de l'application IBM WebSphere MQ classes for JMS à exécuter.

   Une exception contenant le code anomalie IBM WebSphere MQ 2495 (MQRC_MODULE_NOT_FOUND) est émise par les classes IBM WebSphere MQ pour JMS si:

   - L'application IBM WebSphere MQ classes for JMS est exécutée dans un environnement d'exécution Java 32 bits et un environnement 64 bits a été configuré pour les classes IBM WebSphere MQ pour JMS, car l'environnement d'exécution Java 32 bits ne peut pas charger la bibliothèque native Java 64 bits.
   - L'application IBM WebSphere MQ classes for JMS est exécutée dans un environnement d'exécution Java 64 bits et un environnement 32 bits a été configuré pour les classes IBM WebSphere MQ pour JMS, car l'environnement d'exécution Java 64 bits ne peut pas charger la bibliothèque native Java 32 bits.

Le fichier de configuration IBM WebSphere MQ classes for JMS

Un fichier de configuration WebSphere MQ classes for JMS spécifie les propriétés utilisées pour configurer les classes WebSphere MQ pour JMS.

Le format d'un fichier de configuration WebSphere MQ classes for JMS 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 WebSphere 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 WebSphere MQ classes for JMS. 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 WebSphere MQ classes for JMS. 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 WebSphere MQ classes for JMS en tant que fichier D:\mydir\mjms.config sur le système Windows local.

Lorsqu'une application démarre, WebSphere MQ classes for JMS 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, WebSphere MQ classes for JMS utilise les valeurs par défaut pour toutes les propriétés. Si nécessaire, vous pouvez remplacer n'importe quelle propriété du fichier de configuration en la spécifiant comme propriété système dans la commande **java** .

Un fichier de configuration WebSphere MQ classes for JMS peut être utilisé avec n'importe quel transport pris en charge entre une application et un gestionnaire de files d'attente ou un courtier.

Notez que vous ne pouvez pas spécifier de trace de démarrage en définissant une propriété dans le fichier de configuration WebSphere MQ classes for JMS. Vous pouvez spécifier la trace de démarrage uniquement en définissant une propriété système dans la commande **java** , comme illustré dans l'exemple suivant:

```
java -Dcom.ibm.msg.client.commonservices.trace.startup=true
     -Dcom.ibm.msg.client.config.location=file:/D:/mydir/myjms.config
      MyAppClass
```

Remplacement des propriétés spécifiées dans un fichier de configuration de client WebSphere MQ MQI

Un fichier de configuration du client WebSphere MQ MQI peut également spécifier les propriétés utilisées pour configurer les classes WebSphere MQ pour JMS. Toutefois, les propriétés spécifiées dans un fichier de configuration client WebSphere MQ MQI 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 de client WebSphere MQ MQI en le spécifiant comme propriété dans un fichier de configuration WebSphere MQ classes for JMS. Pour remplacer un attribut dans un fichier de configuration client WebSphere MQ MQI, utilisez une entrée au format suivant dans le fichier de configuration WebSphere MQ classes for JMS:

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Les variables de l'entrée ont les significations suivantes:

section
Nom de la strophe dans le fichier de configuration du client WebSphere MQ MQI qui contient l'attribut

propName
Nom de l'attribut tel qu'il est spécifié dans le fichier de configuration du client WebSphere MQ MQI

propValue
Valeur de la propriété qui remplace la valeur de l'attribut spécifié dans le fichier de configuration du client WebSphere MQ MQI

Vous pouvez également remplacer un attribut dans un fichier de configuration du client WebSphere MQ MQI 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 du client MQI WebSphere MQ sont pertinents pour WebSphere MQ classes for JMS. Si vous spécifiez ou remplacez d'autres attributs, cela n'a aucun effet.

stanza	**Attribut**
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
sectionMessageBuffer du fichier de configuration du client	MaximumSize

stanza	Attribut
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

IBM WebSphere 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 WebSphere MQ classes for JMS.

De nombreuses grandes organisations de développement utilisent ces outils pour gérer de manière centralisée les référentiels de bibliothèques tierces.

Les IBM WebSphere MQ classes for JMS 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 de IBM WebSphere MQ Server, Client ou 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 WebSphere 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 WebSphere MQ .
- Vous êtes responsable de la mise à jour du référentiel avec toute maintenance fournie par le support IBM .

Pour IBM WebSphere MQ Version 7.5, les fichiers JAR suivants doivent être installés dans le référentiel:

- com.ibm.mqjms.jar.
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.headers.jar
- CL3Export.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- CL3Nonexport.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- jndi.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- ldap.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- rmm.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- dhbcore.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- jms.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS.
- fscontext.jar est requis si vous utilisez IBM WebSphere MQ classes for JMS et que vous accédez aux objets gérés par JMS qui sont stockés dans un contexte JNDI de système de fichiers.
- providerutil.jar si vous utilisez IBM WebSphere MQ classes for JMS et que vous accédez aux objets gérés par JMS qui sont stockés dans un contexte JNDI de système de fichiers.

Exécution des classes WebSphere MQ pour les applications JMS sous le gestionnaire de sécurité Java

WebSphere MQ classes for JMS peut s'exécuter avec le gestionnaire de sécurité Java activé. Pour exécuter des applications avec le gestionnaire de sécurité activé, vous devez configurer votre machine virtuelle Java (JVM) avec un fichier de configuration de règles approprié.

La méthode la plus simple consiste à modifier le fichier de configuration des règles fourni avec votre environnement d'exécution Java. Sur la plupart des systèmes, ce fichier est stocké dans le chemin `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 du programme `policytool` fourni avec votre environnement d'exécution Java.

Important : V 7.5.0.8 Partout où c'était possible, le terme *allowlist* a remplacé le terme *whitelist*. Une exception est les noms de propriété système Java suivants.

Si vous utilisez le mécanisme du gestionnaire de sécurité Java 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.
- PropertyPermission (lecture) sur les propriétés com.ibm.mq.jms.whitelist, com.ibm.mq.jms.whitelist.discoveret com.ibm.mq.jms.whitelist.mode .

La liste autorisée ClassName est prise en charge avec l' APAR IT14385 et IBM WebSphere MQ Version 7.5.0, groupe de correctifs 8. Pour plus d'informations, voir «ClassName liste autorisée dans JMS ObjectMessage», à la page 757.

Voici un exemple de deux entrées dans un fichier de configuration de règles qui permettent à WebSphere MQ classes for JMS de s'exécuter correctement sous le gestionnaire de sécurité par défaut:

```
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mq.jmqi.jar" {
  //Required
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  //Required if mqclient.ini/mqs.ini configuration files are used
  permission java.io.FilePermission "/var/mqm/mqclient.ini","read";
  permission java.io.FilePermission "/var/mqm/mqs.ini","read";
  //For the client transport type.
  permission java.net.SocketPermission "*","connect";
  //For the bindings transport type.
  permission java.lang.RuntimePermission "loadLibrary.*";
  //For applications that use CCDT tables (access to the CCDT
AMQCLCHL.TAB)
  permission java.io.FilePermission
"/var/mqm/qmgrs/QMGR/@ipcc/AMQCLCHL.TAB","read";
  //For applications that use User Exits
  permission java.io.FilePermission "/var/mqm/exits/*","read";
  permission java.lang.RuntimePermission "createClassLoader";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
grant codeBase "file:MQ_INSTALLATION_PATH/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "os.name","read";
  permission java.util.PropertyPermission "console.encoding","read";
  permission java.lang.RuntimePermission "setContextClassLoader";
  //tracing permissions
  permission java.util.PropertyPermission "com.ibm.msg.client.commonservices.*","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
  permission java.util.logging.LoggingPermission "control";
  //Wherever trace output is expected
  permission java.io.FilePermission "/tmp/*","read,write";
  //Required for the z/OS platform
  permission java.util.PropertyPermission
"com.ibm.vm.bitmode","read";
};
```

Remarques :

- *MQ_INSTALLATION_PATH* représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.
- La première instruction `grant` contient les droits requis par WebSphere MQ classes for JMS, et la deuxième instruction `grant` contient les droits requis par une application WebSphere MQ classes for JMS.
- Pour permettre à WebSphere MQ classes for JMS d'accéder aux fichiers d'archive Java (JAR) d'une application, ajoutez les droits suivants à la première instruction `grant` :

```
permission java.io.FilePermission "/path_to_your_app/-", "read";
```

- Pour utiliser ces instructions `grant` 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 WebSphere MQ classes for JMS et où vous stockez vos applications.
- Les exemples d'application fournis avec WebSphere MQ classes for JMS et les scripts permettant de les exécuter n'activent pas le gestionnaire de sécurité.

V 7.5.0.8 ClassName liste autorisée dans JMS ObjectMessage

V 7.5.0.8 Dans WebSphere MQ classes for JMS, la prise en charge de la liste autorisée des classes dans l'implémentation de l'interface JMS ObjectMessage offre une atténuation potentielle des risques de sécurité liés au mécanisme de sérialisation et de désérialisation des objets Java.

Remarque : Partout où c'était possible, le terme *allowlist* a remplacé le terme *whitelist*. Toutefois, il existe une exception : les noms de propriété système Java mentionnés dans cette rubrique.

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 les ObjectMessages JMS (Java Message Service) 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.

Liste autorisée dans WebSphere MQ classes for JMS

Avec APAR IT14385 et IBM WebSphere MQ Version 7.5.0, groupe de correctifs 8, WebSphere MQ classes for JMS prend en charge la liste autorisée des classes dans l'implémentation de l'interface JMS ObjectMessage . Une liste autorisée définit les classes Java qui peuvent être sérialisées avec ObjectMessage.setObject() et désérialisées avec ObjectMessage.getObject().

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.

Génération de la liste autorisée

Important : WebSphere MQ classes for JMS ne peut pas être distribué avec une liste autorisée. Le choix des classes à transférer à l'aide de ObjectMessages est un choix de conception d'application et IBM WebSphere MQ ne peut pas l'anticiper.

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.whitelist
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 gestionnaire de sécurité Java , vous devez vous assurer que les fichiers JAR WebSphere MQ classes for JMS disposent d'un accès en lecture et en écriture à ce fichier.

com.ibm.mq.jms.whitelist.discover

- 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.whitelist, 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.whitelist.mode
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 : Le fichier peut nécessiter une conversion si vous le déplacez entre des systèmes hétérogènes.

- 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 UNIX ou 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 JMS IBM WebSphere MQ 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 IBM WebSphere MQ 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 du gestionnaire de sécurité Java 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.

- PropertyPermission (lecture) sur les propriétés com.ibm.mq.jms.whitelist, com.ibm.mq.jms.whitelist.discoveret com.ibm.mq.jms.whitelist.mode .

Concepts associés

«Exécution des classes WebSphere MQ pour les applications JMS sous le gestionnaire de sécurité Java», à la page 756
WebSphere MQ classes for JMS peut s'exécuter avec le gestionnaire de sécurité Java activé. Pour exécuter des applications avec le gestionnaire de sécurité activé, vous devez configurer votre machine virtuelle Java (JVM) avec un fichier de configuration de règles approprié.

L'adaptateur de ressources IBM WebSphere MQ

L'adaptateur de ressources permet aux applications exécutées dans un serveur d'applications d'accéder aux ressources IBM WebSphere MQ . Il prend en charge les communications entrantes et sortantes.

Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) fournit un moyen standard de connecter des applications s'exécutant dans un environnement Java EE à un système d'information d'entreprise (EIS) tel que IBM WebSphere MQ ou Db2. L'adaptateur de ressources IBM WebSphere MQ implémente les interfaces JCA 1.5 et contient les IBM WebSphere 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 WebSphere 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 WebSphere 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 arrivant à une destination JMS est distribué à un bean géré par message, qui traite le message de manière asynchrone.

Pour plus d'informations sur le IBM WebSphere MQ classes for JMS, voir «Utilisation des classes WebSphere MQ pour JMS», à la page 741.

L'adaptateur de ressources contient également le IBM WebSphere MQ classes for Java. Les classes sont automatiquement disponibles pour les applications exécutées sur un serveur d'applications dans lequel l'adaptateur de ressources a été déployé et permettent aux applications exécutées sur ce serveur d'applications d'utiliser l'API IBM WebSphere MQ classes for Java lors de l'accès aux ressources d'un gestionnaire de files d'attente IBM WebSphere MQ . Pour plus d'informations sur le IBM WebSphere MQ classes for Java, voir «Utilisation des classes WebSphere MQ pour Java», à la page 677.

L'utilisation de IBM WebSphere 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 des classes IBM WebSphere MQ pour les applications Java sur la plateforme Java Enterprise Edition», à la page 739.

Autre documentation requise pour la prise en charge d'un adaptateur de ressources JCA

Pour plus d'informations sur la configuration d'un adaptateur de ressources JCA, reportez-vous à la documentation de votre serveur d'applications.

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 WebSphere MQ pour chaque serveur d'applications. Cette documentation se concentre 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.

Pour comprendre cette documentation, vous devez connaître les classes JMS et WebSphere MQ for JMS. De nombreuses propriétés utilisées pour configurer l'adaptateur de ressources WebSphere MQ sont équivalentes aux propriétés des objets WebSphere MQ classes for JMS et ont la même fonction.

Installation de l'adaptateur de ressources WebSphere MQ

L'adaptateur de ressources WebSphere 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.

L'adaptateur de ressources WebSphere MQ est fourni en tant que fichier d'archive de ressource (RAR) appelé wmq.jmsra.rar. Ce fichier est installé avec WebSphere MQ classes for JMS dans le répertoire indiqué dans Tableau 94, à la page 761.

Tableau 94. Répertoire contenant wmq.jmsra.rar pour chaque plateforme

Plateforme	**Répertoire**
AIX, HP-UX, Linux et Solaris	*MQ_INSTALLATION_PATH* /java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH* \java\lib\jca
MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.	

Le fichier RAR contient les classes WebSphere MQ pour JMS et l'implémentation WebSphere MQ des interfaces JCA.

Vous devez installer le fichier RAR de l'adaptateur de ressources WebSphere MQ sur votre serveur d'applications, mais cela dépend du serveur d'applications. Consultez la documentation de votre serveur d'applications pour plus d'informations sur l'installation d'un fichier RAR d'adaptateur de ressources.

Pour les connexions de liaisons sur les systèmes UNIX and Linux , vous devez vous assurer 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 classes WebSphere MQ pour les bibliothèques JMS, voir Tableau 93, à la page 751. Sous Windows, ce répertoire est automatiquement ajouté au chemin système lors de l'installation des classes WebSphere MQ pour JMS.

Les transactions sont prises en charge à la fois en mode client et en mode liaisons.

L'adaptateur de ressources WebSphere MQ et la version de WebSphere MQ classes for JMS utilisées par l'adaptateur de ressources doivent être au même niveau d'édition.

WebSphere Application Server et l'adaptateur de ressources WebSphere MQ

N'utilisez pas l'adaptateur de ressources WebSphere MQ avec WebSphere Application Server version 6. WebSphere Application Server, V7, inclut une version de l'adaptateur de ressources WebSphere MQ V7 .

N'utilisez pas l'adaptateur de ressources WebSphere MQ dans WebSphere Application Server, V6. Pour accéder aux ressources d'un gestionnaire de files d'attente WebSphere MQ depuis WebSphere Application Server à partir d'une application JMS, utilisez le fournisseur de messagerie WebSphere MQ . Le fournisseur de messagerie WebSphere MQ contient une version des classes WebSphere MQ for JMS.

WebSphere Application Server, V7, inclut une version de l'adaptateur de ressources WebSphere MQ V7 .

Pour plus d'informations, voir la note technique Quelle version de WebSphere MQ Resource Adapter (RA) est fournie avec WebSphere Application Server ?.

WebSphere Application Server Liberty et l'adaptateur de ressources IBM WebSphere MQ

L'adaptateur de ressources IBM WebSphere MQ Version 7.5 peut être installé dans WebSphere Application Server Liberty version 8.5.5, groupe de correctifs 2 ou ultérieur, à l'aide de la fonction wmqJmsClient-1.1 . Vous pouvez également, sous réserve de certaines restrictions, installer l'adaptateur de ressources à l'aide de la prise en charge générique de Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

Restrictions générales lors de l'installation de l'adaptateur de ressources dans Liberty

Les restrictions suivantes s'appliquent à l'adaptateur de ressources Version 7.5 lors de l'utilisation de la fonction wmqJmsClient-1.1 et de la prise en charge JCA générique:

- Les IBM WebSphere MQ classes for Java ne sont pas prises en charge dans Liberty. Ils ne doivent pas être utilisés avec la fonction de messagerie IBM WebSphere MQ Liberty ni avec la prise en charge JCA générique. Pour plus d'informations, voir Utilisation des interfaces Java WebSphere MQ dans les environnements J2EE/JEE.
- Le type de transport de l'adaptateur de ressources IBM WebSphere MQ est BINDINGS_THEN_CLIENT. Ce type de transport n'est pas pris en charge dans la fonction de messagerie IBM WebSphere MQ Liberty.
- La fonction IBM WebSphere MQ Advanced Message Security (IBM WebSphere MQ AMS) n'est pas incluse dans la fonction de messagerie IBM WebSphere MQ Liberty.

L'adaptateur de ressources IBM WebSphere MQ Version 7.5 ne peut pas être utilisé avec la fonction wmqJmsClient-2.0 .

Configuration de l'adaptateur de ressources WebSphere MQ

Pour configurer l'adaptateur de ressources WebSphere MQ , vous devez définir diverses ressources JCA et propriétés système.

Définissez les ressources JCA dans les catégories suivantes:

- Les propriétés de l'objet ResourceAdapter, qui représentent les propriétés globales de l'adaptateur de ressources, telles que le niveau de suivi des diagnostics. Ces propriétés sont décrites dans «Configuration de l'objet ResourceAdapter», à la page 763.
- Les propriétés d'un objet ActivationSpec, qui déterminent comment un bean géré par message est activé pour les communications entrantes. Ces propriétés sont décrites dans «Configuration de l'adaptateur de ressources pour les communications entrantes», à la page 765.
- Propriétés d'un objet ConnectionFactory , que le serveur d'applications utilise pour créer un objet JMS ConnectionFactory pour les communications sortantes. Ces propriétés sont décrites dans «Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 779.
- 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 pour la communication sortante. Ces propriétés sont également décrites dans «Configuration de l'adaptateur de ressources pour les communications sortantes», à la page 779.

Le fichier RAR de l'adaptateur de ressources WebSphere 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://java.sun.com/xml/ns/j2ee/connector_1_5.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. Par exemple, WebSphere Application Server Community Edition requiert un plan de déploiement appelé geronimo-ra.xml.

Si vous utilisez SSL (Secure Sockets Layer), 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 SSL. Le serveur d'applications peut également réinitialiser ces propriétés à des valeurs différentes. Pour plus

d'informations sur l'utilisation de SSL avec WebSphere MQ classes for JMS, voir «Utilisation de SSL (Secure Sockets Layer) avec WebSphere MQ classes for JMS», à la page 936.

Un programme de test de vérification de l'installation (IVT) est fourni avec l'adaptateur de ressources WebSphere MQ , mais vous devez le configurer avant de pouvoir exécuter le programme. Pour plus d'informations sur les éléments à configurer pour exécuter le programme IVT, voir «Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810.

Les journaux de l'adaptateur de ressources, les messages d'avertissement et d'erreur utilisent le même mécanisme que les classes IBM WebSphere MQ pour JMS. Pour plus de détails, voir «Consignation etIBM WebSphere MQ classes for JMS», à la page 824. Pour WebSphere Application Server, ces messages sont automatiquement redirigés vers le journal de sortie du serveur d'applications. Pour les autres serveurs d'applications, tels que WAS CE et JBoss, ces derniers vont, par défaut, accéder à un fichier appelé mqjms.log. Pour configurer l'adaptateur de ressources de sorte qu'il consigne en plus les messages d'avertissement dans le journal de sortie standard de vos serveurs d'applications, définissez la propriété système JVM suivante pour votre serveur d'applications:

```
-Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
```

Pour plus de détails sur la définition d'une propriété système JVM, consultez la documentation de votre serveur d'applications.

Configuration de l'objet ResourceAdapter
L'objet ResourceAdapter encapsule les propriétés globales de l'adaptateur de ressources WebSphere MQ . Définissez ces propriétés à l'aide des fonctions de votre adaptateur de ressources.

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.

Pour plus d'informations sur la définition des propriétés associées à la trace de diagnostic, voir Traçage de l'adaptateur de ressources IBM WebSphere MQ.

L'adaptateur de ressources gère un pool de connexions internes de connexions JMS qui sont utilisées pour distribuer des messages aux beans gérés par message. Tableau 95, à la page 763 répertorie les propriétés de l'objet ResourceAdapter qui sont associées au pool de connexions.

Tableau 95. 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 WebSphere 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 WebSphere MQ en cas d'échec d'une connexion.

Tableau 95. Propriétés de l'objet ResourceAdapter associées au pool de connexions (suite)			
Nom de la propriété	**Tapez**	**Valeur par défaut**	**Description**
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 WebSphere 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).

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 WebSphere 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 UNIX 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 WebSphere 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 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 les messages JMS d'une file d'attente WebSphere 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 WebSphere MQ
- Propriétés utilisées pour créer un consommateur de connexion JMS qui distribue des 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.

Le Tableau 96, à la page 765 répertorie les propriétés d'un objet ActivationSpec qui sont utilisées pour créer une connexion JMS à un gestionnaire de files d'attente WebSphere MQ .

Tableau 96. 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'affichage **Connexions d'application** de l'explorateur IBM WebSphere MQ (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
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

Tableau 96. 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
brokerSubQueue[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 de publication / abonnement WebSphere MQ . Ou pour utiliser un courtier WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker ou WebSphere Business Integration Message Broker en mode compatibilité. Cette valeur est la valeur par défaut si TRANSPORT est défini sur BIND ou CLIENT. • **V2** -Pour utiliser un courtier WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker ou WebSphere Business Integration Message Broker 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[1]	int	• **3 600 000** • Entier positif	Intervalle, en millisecondes, entre les exécutions en arrière-plan de l'utilitaire de nettoyage de publication / abonnement

Tableau 96. 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
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
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.
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

Tableau 96. 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
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.
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	Liste des techniques pouvant être utilisées pour compresser les données de message sur une connexion

Tableau 96. 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
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
messageSelection[1]	String	• **client** • COURTIER	Détermine si la sélection des messages est effectuée par WebSphere 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 96. 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 WebSphere 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 96. 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, WebSphere MQ classes for JMS recherche dans la file d'attente WebSphere MQ les messages appropriés dans la séquence déterminée par l'attribut *MsgDeliverySequence* de la file d'attente. Lorsque WebSphere MQ classes for JMS trouve un message approprié et le distribue au destinataire, WebSphere MQ classes for JMS reprend la recherche du message approprié suivant à partir de sa position actuelle dans la file d'attente. WebSphere 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, arrive à expiration. Dans chaque cas, WebSphere 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 WebSphere 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 96. 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 WebSphere 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 SSL
SSLCIPHERSUITE	String	• **null** • Nom d'une CipherSuite	CipherSuite à utiliser pour une connexion SSL
sslFipsObligatoire [2]	Boolean	• **faux** • Oui	Indique si une connexion SSL doit utiliser une CipherSuite prise en charge par le fournisseur Java JSSE FIPS IBM (IBMJSSEFIPS)

Tableau 96. 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 SSL, 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 SSL avant que les clés secrètes utilisées par SSL 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 .
statusRefreshInterval[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 WebSphere MQ classes for JMS stocke les données persistantes sur les abonnements actifs
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.

Tableau 96. 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**
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 7.0 de WebSphere MQ classes for JMS. Elle n'affecte pas une application connectée à un gestionnaire de files d'attente version 7.0 sauf si la propriété providerVersion est définie sur un numéro de version inférieur à 7.
2. Pour des informations importantes sur l'utilisation de la propriété sslFipsRequired, voir «Limitations de l'adaptateur de ressources IBM WebSphere MQ», à la page 798.
3. Pour plus d'informations sur la configuration de l'adaptateur de ressources afin qu'il puisse localiser un exit, voir «Configuration de IBM WebSphere MQ classes for JMS pour l'utilisation des exits de canal», à la page 943.

Tableau 97, à la page 774 répertorie les propriétés d'un objet ActivationSpec qui sont utilisées pour créer un consommateur de connexion JMS.

Tableau 97. 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.
destinationType	String	• javax.jms.Queue • javax.jms.Topic	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.
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

Tableau 97. 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
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 uniquement lorsque l'application s'exécute sur WebSphere Application Server version 7 ou ultérieure.
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 WebSphere 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
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 le bean géré par message est autorisé à utiliser la lecture anticipée pour extraire des messages non persistants de la destination dans une mémoire tampon interne avant de les recevoir

Tableau 97. 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
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.
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 97. 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 WebSphere MQ ou d'une rubrique. • true-La propriété appelée destination est interprétée comme le nom d'un objet javax.jms.Queue ou javax.jms.Topic dans l'espace de nom JNDI du serveur d'applications.	Détermine comment la valeur de la propriété appelée destination est interprétée

Les propriétés ActivationSpec appelées destination et destinationType doivent être définies explicitement. Toutes les autres propriétés sont facultatives.

Un objet ActivationSpec peut avoir des propriétés en conflit. Par exemple, vous pouvez spécifier des propriétés SSL 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, ce qui indique que vous souhaitez utiliser des abonnements durables, vous devez également définir la propriété subscriptionName .

Si les propriétés ccdtURL et le canal sont définis, une exception InvalidPropertyest émise. Toutefois, si vous définissez uniquement la propriété ccdtURL , la propriété appelée channel reste avec la 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.

La plupart des propriétés d'un objet ActivationSpec sont équivalentes aux propriétés des objets WebSphere MQ classes for JMS ou des paramètres des méthodes WebSphere MQ classes for JMS. Toutefois, trois propriétés d'optimisation et une propriété de mode d'utilisation n'ont pas d'équivalent dans WebSphere MQ classes for JMS:

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 97, à la page 774.

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:

```
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 WebSphere MQ», à la page 761.

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.

Traitement des messages incohérents entrants dans l'adaptateur de ressources

Dans certains cas, un message distribué à un bean géré par message peut être annulé dans une file d'attente WebSphere 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 WebSphere MQ de sorte que les classes WebSphere MQ pour JMS transfèrent

automatiquement un message incohérent vers une autre file d'attente pour un examen plus approfondi ou le supprime.

Pour plus de détails sur la gestion des messages incohérents, voir «Traitement des messages incohérents dans IBM WebSphere MQ classes for JMS», à la page 919.

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 UNIX, Linux et Windows

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.

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:

```
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 WebSphere MQ . Le servlet reçoit ensuite le message qu'il a envoyé.

Pour configurer la communication sortante, définissez les ressources 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 administré, 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.

Tableau 98, à la page 780 répertorie les propriétés d'un objet ConnectionFactory .

Tableau 98. Propriétés d'un objet ConnectionFactory

Nom de la propriété	Type	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 WebSphere MQ (où la zone est appelée **App name**).
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.
brokerPubQueue[1]	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).
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.
brokerSubQueue[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. Pour plus d'informations, voir la propriété BROKERSUBQ .

Tableau 98. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
brokerVersion[1]	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 WebSphere MQ . Ou pour utiliser un courtier de IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker ou WebSphere Business Integration Message Broker en mode compatibilité. Cette valeur est la valeur par défaut si TRANSPORT est défini sur BIND ou CLIENT. • **V2** -Pour utiliser un courtier de IBM WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker ou WebSphere Business Integration Message Broker 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[1]	int	• **3 600 000** • Entier positif	Intervalle, en millisecondes, entre les exécutions en arrière-plan de l'utilitaire de nettoyage de publication / abonnement.

Tableau 98. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
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 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.

Tableau 98. Propriétés d'un objet ConnectionFactory (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	Liste des techniques pouvant être utilisées pour compresser les données de message sur une connexion.
messageSelection[1]	String	• **client** • COURTIER	Détermine si la sélection des messages est effectuée par IBM WebSphere 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 98. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Type	Valeurs valides (valeur par défaut en gras)	Description
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.
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.
pubAckInterval[1]	int	• **25** • Entier positif	Nombre de messages publiés par un diffuseur de publications avant que IBM WebSphere 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 98. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Type	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 WebSphere 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[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, WebSphere MQ classes for JMS recherche dans la file d'attente IBM WebSphere MQ les messages appropriés dans l'ordre déterminé par l'attribut *MsgDeliverySequence* de la file d'attente. Lorsque WebSphere MQ classes for JMS trouve un message approprié et le distribue au destinataire, WebSphere MQ classes for JMS reprend la recherche du message approprié suivant à partir de sa position actuelle dans la file d'attente. WebSphere 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, arrive à expiration. Dans chaque cas, WebSphere 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 WebSphere MQ classes for Java, MQSecurityExit	Identifie un programme d'exit de sécurité de canal.

Tableau 98. 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 WebSphere 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 SSL.

Tableau 98. 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 SSL.
sslFipsObligatoire[2]	Boolean	• **faux** • Oui	Indique si une connexion SSL 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 SSL, 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 SSL avant que les clés secrètes utilisées par SSL 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 .
statusRefreshInterval[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 la valeur de la substance est QUEUE.
subscriptionStore[1]	String	• **Courtier** • MIGRATE • QUEUE	Détermine où WebSphere MQ classes for JMS stocke les données persistantes sur les abonnements actifs.
Correspondance targetClient	Boolean	• **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 .

Tableau 98. 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 conditions suivantes sont remplies: • 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 YES et 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 98. Propriétés d'un objet ConnectionFactory (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
tempQPrefix	String	• **"" (chaîne vide)** • Préfixe qui peut être utilisé pour former le nom d'une file d'attente dynamique IBM WebSphere 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 WebSphere MQ , structure MQOD, mais le dernier caractère non blanc doit être un astérisque (*). Si la valeur de la propriété est une chaîne vide, WebSphere 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 WebSphere MQ .
TEMPTOPICPREFIX	String	Toute chaîne non nulle constituée uniquement de caractères valides pour une chaîne de rubrique IBM WebSphere MQ	Lors de la création de rubriques temporaires, JMS génère une chaîne de rubrique de la forme "TEMP/*TEMPTOPICPREFIX*/*unique_id*" ou, si cette propriété est laissée avec la valeur par défaut, uniquement "TEMP/*unique_id*". 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.
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.
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.

Tableau 98. Propriétés d'un objet ConnectionFactory (suite)			
Nom de la propriété	**Type**	**Valeurs valides (valeur par défaut en gras)**	**Description**
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. Cette propriété peut être utilisée avec la version 7.0 de IBM WebSphere MQ classes for JMS mais n'affecte pas une application connectée à un gestionnaire de files d'attente version 7.0 sauf si la propriété providerVersion est définie sur un numéro de version inférieur à 7.
2. Pour des informations importantes sur l'utilisation de la propriété sslFipsRequired, voir «Limitations de l'adaptateur de ressources IBM WebSphere MQ», à la page 798.
3. Pour plus d'informations sur la configuration de l'adaptateur de ressources afin qu'il puisse localiser un exit, voir «Configuration de IBM WebSphere MQ classes for JMS pour l'utilisation des exits de canal», à la page 943.

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
```

Le Tableau 99, à la page 790 répertorie les propriétés communes à un objet Queue et à un objet Topic.

Tableau 99. 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.

Tableau 99. 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
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.
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.
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 WebSphere MQ . • PERS-Un message est persistant. • NON-Un message est non persistant. • ELEVE-La persistance d'un message est déterminée par l'attribut *NonPersistentMessageClass* de la file d'attente WebSphere MQ en fonction de l'explication fournie dans «Messages persistants JMS», à la page 935.	Persistance d'un message envoyé à la destination.

Tableau 99. 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
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 WebSphere 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.
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`

Tableau 99. 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**
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 IBM WebSphere MQ non JMS.	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 100, à la page 793 répertorie les propriétés spécifiques à un objet Queue.

Tableau 100. 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 WebSphere MQ sous-jacente.
Nom baseQueue	String	• **"" (chaîne vide)** • Un nom de file d'attente	Nom de la file d'attente IBM WebSphere MQ sous-jacente.

Le Tableau 101, à la page 793 répertorie les propriétés spécifiques à un objet de rubrique.

Tableau 101. 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[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.
brokerDurSubQueue[1]	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 WebSphere MQ Explorer.

Tableau 101. Propriétés spécifiques à un objet de rubrique (suite)

Nom de la propriété	Tapez	Valeurs valides (valeur par défaut en gras)	Description
brokerPubQueue[1]	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 de file d'attente brokerPubde l'objet ConnectionFactory . Toutefois, si vous ne définissez pas la valeur de cette propriété, la valeur de la propriété de file d'attente brokerPubde l'objet ConnectionFactory est utilisée à la place.
brokerPubQueueManager[1]	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[1]	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.

Remarque :

1. Cette propriété peut être utilisée avec la version 7.0 de IBM WebSphere MQ classes for JMS mais n'affecte pas une application connectée à un gestionnaire de files d'attente version 7.0 sauf si la propriété providerVersion de l'objet ConnectionFactory est définie sur un numéro de version inférieur à 7.

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

Référence associée

FIPS (Federal Information Processing Standards) pour UNIX, Linux et Windows

V 7.5.0.9 *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 WebSphere 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 WebSphere Application Server 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 WebSphere 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 WebSphere 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 Application Server 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 907
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 WebSphere MQ ou une rubrique et, en option, 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 779
Pour configurer les communications sortantes, définissez les propriétés d'un objet ConnectionFactory et d'un objet de destination administré.

Mode ASF et non ASF
Le mode ASF (Application Server Facilities) est la méthode par défaut utilisée par le service d'écoute des messages dans WebSphere Application Server pour traiter les messages.

Le service d'écoute de messages dispose de deux modes de fonctionnement, Application Server Facilities (ASF) et non-Application Server Facilities (non-ASF):

- Le mode ASF fournit une prise en charge transactionnelle et des accès concurrents pour les applications. Pour les beans d'unité de message de publication / abonnement, le mode ASF offre un meilleur débit et un meilleur accès concurrent, car en mode non ASF, le programme d'écoute est à unité d'exécution unique.
- Le mode non ASF est principalement destiné à être utilisé avec des fournisseurs de messagerie tiers qui ne prennent pas en charge l'ASF JMS, qui est une extension facultative de la spécification JMS. Le mode non ASF est également transactionnel, mais, comme la longueur du chemin est plus courte que pour le mode ASF, il fournit généralement des performances améliorées.

Pour activer le mode de fonctionnement non ASF pour tous les programmes d'écoute de bean géré par message sur le serveur d'applications, définissez cette propriété sur une valeur différente de zéro.

Remarque :

Le mode non ASF ne pouvant pas être sélectionné sur les systèmes z/OS , vous ne devez pas définir de valeur différente de zéro pour cette propriété dans ce cas.

Traitement des messages en mode ASF

En mode ASF, les sessions de serveur et les unités d'exécution sont uniquement allouées pour le travail lorsqu'un message adapté au bean géré par message (MDB) est détecté. Le nombre d'unités d'exécution qu'un bean géré par message peut traiter simultanément est déterminé par la valeur de la propriété **Maximum Sessions** pour le port d'écoute ou la spécification d'activation.

Traitement des messages en mode non ASF

En mode non ASF, les unités d'exécution sont actives dès le démarrage du port d'écoute ou de la spécification d'activation. Le nombre d'unités d'exécution actives est déterminé par la valeur spécifiée pour la propriété **Maximum Sessions** . Le nombre d'unités d'exécution spécifié dans la propriété **Maximum Sessions** est actif, quel que soit le nombre de messages pouvant être traités. Chaque unité d'exécution active est une connexion réseau physique individuelle.

IBM WebSphere MQ Version 7.0 ou ultérieure vous permet d'avoir jusqu'à dix unités d'exécution partageant une seule connexion réseau physique.

Concepts associés

ClassesIBM WebSphere MQ pour JMS Application Server Facilities
Cette rubrique décrit comment WebSphere 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.

Tâches associées

Configuration des spécifications d'activation pour le mode non ASF
Les spécifications d'activation constituent la méthode normalisée de gestion et de configuration de la relation entre un bean géré par message (MDB) exécuté dans WebSphere Application Server et une destination dans IBM WebSphere MQ. Cette tâche explique comment configurer WebSphere Application Server pour qu'il utilise le mode non ASF pour traiter les messages.

Information associée

Traitement des messages en mode ASF et en mode non ASF

Configuration des spécifications d'activation pour le mode non ASF

Les spécifications d'activation constituent la méthode normalisée de gestion et de configuration de la relation entre un bean géré par message (MDB) exécuté dans WebSphere Application Server et une destination dans IBM WebSphere MQ. Cette tâche explique comment configurer WebSphere Application Server pour qu'il utilise le mode non ASF pour traiter les messages.

Avant de commencer

La manière dont vous définissez les propriétés d'une spécification d'activation dépend des interfaces d'administration fournies par votre serveur d'applications. Cette tâche suppose que vous utilisez WebSphere Application Server version 7 ou ultérieure comme serveur d'applications et IBM WebSphere MQ comme fournisseur de messagerie.

Remarque :

Le mode non ASF ne peut pas être sélectionné sur les systèmes z/OS .

Pourquoi et quand exécuter cette tâche

Les propriétés d'une spécification d'activation déterminent comment un bean géré par message (MDB) reçoit les messages JMS d'une file d'attente IBM WebSphere MQ . Pour configurer le mode non ASF, définissez les propriétés d'une ou de plusieurs spécifications d'activation.

Il existe plusieurs configurations IBM WebSphere MQ que vous pouvez utiliser en mode non ASF. Avec les configurations suivantes, chaque unité d'exécution utilise une connexion réseau physique distincte:

- Un gestionnaire de files d'attente IBM WebSphere MQ version 7.x , utilisant une fabrique de connexions dont la propriété Version du fournisseur est définie sur 6.
- Un gestionnaire de files d'attente IBM WebSphere MQ version 7.x , utilisant une fabrique de connexions dont la propriété Version du fournisseur est définie sur 7 ou non spécifiée, se connectant sur un canal IBM WebSphere MQ dont le paramètre **SHARECNV** (partage de conversations) est défini sur 0.

Pour configurer une non-ASF, définissez la propriété ActivationSpec **NON.ASF.RECEIVE.TIMEOUT** sur un entier positif, qui 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. Pour plus de détails, voir **Propriétés personnalisées du service d'écoute de messages**.

Ce paramètre est valide uniquement lorsque l'application s'exécute sur WebSphere Application Server version 7 ou ultérieure.

Procédure

1. Démarrez la console d'administration de WebSphere Application Server.
2. Affichez la page des paramètres du service d'écoute:
   a) Dans le panneau de navigation, sélectionnez **Serveurs > Types de serveurs > WebSphere**.
   b) Dans la sous-fenêtre de contenu, cliquez sur le nom du serveur d'applications.
   c) Sous **Communications**, cliquez sur **Messagerie > Service d'écoute de message**.
3. Définissez la propriété personnalisée **NON.ASF.RECEIVE.TIMEOUT** en tant que propriétés personnalisées du service d'écoute des messages.
   a) Cliquez sur **Propriétés personnalisées**.
   b) Cliquez sur **Nouveau**.
   c) Entrez le nom de la propriété, **NON.ASF.RECEIVE.TIMEOUT**, dans la zone **Nom** .
   d) Entrez la valeur requise dans la zone **Valeur** .

e) Cliquez sur **OK**.

4. Sauvegardez vos modifications dans la configuration principale.
5. Pour activer la configuration modifiée, arrêtez puis redémarrez le serveur d'applications.

Résultats

Vous avez configuré les propriétés du service d'écoute de messages pour que WebSphere Application Server utilise le mode non ASF.

Remarque : Lorsque vous utilisez le mode non ASF, vous devez vous assurer que vous accordez suffisamment de temps pour que le traitement soit terminé avant que le délai d'expiration total de la durée de vie de la transaction soit atteint, afin d'éviter les dépassements de délai de transaction non souhaités. Pour plus de détails, voir **NON.ASF.RECEIVE.TIMEOUT** dans la documentation du produit WebSphere Application Server .

Concepts associés

«Mode ASF et non ASF», à la page 796
Le mode ASF (Application Server Facilities) est la méthode par défaut utilisée par le service d'écoute des messages dans WebSphere Application Server pour traiter les messages.

Configuration de l'adaptateur de ressources pour les communications entrantes
Pour configurer les communications entrantes, définissez les propriétés d'un ou plusieurs objets ActivationSpec.

Information associée

Beans gérés par message

Service d'écoute de messages

Traitement des messages en mode ASF et en mode non ASF

Mode de traitement des messages en mode non ASF

Limitations de l'adaptateur de ressources IBM WebSphere MQ

Lorsque vous utilisez l'adaptateur de ressources IBM WebSphere MQ , certaines fonctions de IBM WebSphere MQ ne sont pas disponibles ou sont limitées.

L'adaptateur de ressources IBM WebSphere MQ présente les limitations suivantes:

- L'adaptateur de ressources IBM WebSphere MQ est pris en charge sur toutes les plateformes IBM WebSphere MQ , à l'exception de z/OS.
- L'adaptateur de ressources IBM WebSphere MQ ne prend pas en charge les connexions en temps réel à un courtier. Il prend en charge uniquement les connexions à un gestionnaire de files d'attente IBM WebSphere MQ en mode client ou liaisons.
- L'adaptateur de ressources IBM WebSphere 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 WebSphere MQ émet le code anomalie MQRC_UNSUPPORTED_CIPHER_SUITE, même si une connexion SSL 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 introduite dans IBM WebSphere MQ Version 7.0.1 n'est pas prise en charge pour les connexions sortantes lors de l'exécution dans un conteneur JEE (EJB/Servlet). La nouvelle tentative de connexion n'est pas du tout prise en charge pour JMS sortant 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.

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 UNIX, Linux et Windows

Configuration post-installation pour les classes WebSphere MQ pour les applications JMS

Cette rubrique vous indique les droits dont ont besoin les applications WebSphere MQ 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 WebSphere 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 doit être autorisée à utiliser le gestionnaire de files d'attente avec lequel elle fonctionne.

Pour plus de détails sur le contrôle d'accès dans IBM WebSphere MQ, voir Configuration de la sécurité sous Windows, systèmes UNIX and Linux .

WebSphere Les classes MQ pour les applications 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 les classes WebSphere MQ for JMS se connectent à un gestionnaire de files d'attente IBM WebSphere MQ en mode de migration du fournisseur de messagerie IBM WebSphere MQ :

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Pour plus d'informations sur le mode de migration du fournisseur de messagerie IBM WebSphere MQ , voir Quand utiliser **PROVIDERVERSION**

De plus, si les classes WebSphere MQ for JMS se connectent à un gestionnaire de files d'attente dans ce mode, toute application qui publie des messages doit avoir accès 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 WebSphere MQ Resource Adapter ou le fournisseur de messagerie WebSphere Application Server IBM WebSphere 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 WebSphere MQ pour 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 WebSphere MQ classes for JMS

Une application WebSphere MQ classes for JMS peut se connecter à un gestionnaire de files d'attente en mode client ou liaisons. En mode client, WebSphere MQ classes for JMS se connecte au gestionnaire de files d'attente via TCP/IP. En mode liaisons, WebSphere MQ classes for JMS se connecte directement au gestionnaire de files d'attente à l'aide de l'interface JNI (Java Native Interface).

Une application s'exécutant dans WebSphere Application Server sous z/OS peut se connecter à un gestionnaire de files d'attente en mode liaisons ou client, mais une application s'exécutant dans un autre environnement sous z/OS peut se connecter à un gestionnaire de files d'attente uniquement en mode liaisons. 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 de WebSphere 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 de WebSphere 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 WebSphere MQ classes for JMS peut s'exécuter sur le même système que le gestionnaire de files d'attente ou sur un autre système. Dans chaque cas, WebSphere 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 WebSphere MQ classes for JMS doit s'exécuter sur le même système que le gestionnaire de files d'attente.

Les classes WebSphere MQ pour JMS se connectent directement au gestionnaire de files d'attente à l'aide de l'interface JNI (Java Native Interface). Pour utiliser le transport de liaisons, les classes WebSphere MQ pour JMS doivent être exécutées dans un environnement qui a accès aux bibliothèques WebSphere MQ

Java Native Interface ; voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 751 pour plus d'informations.

Les classes WebSphere MQ pour JMS prennent en charge les valeurs suivantes pour *ConnectOption* :

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_LIEN_PARTAGE
- MQCNO_LIEN_ISOLÉ_LIAISON
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_Q_MGR

Pour modifier les options de connexion utilisées par les classes WebSphere MQ pour 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 217

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 se connectent les classes WebSphere MQ pour JMS .

Pour plus de détails sur la façon de déterminer les CCSID pris en charge par un environnement d'exécution Java, voir WebSphere MQ FDC with Probe id 21 generated when using WebSphere MQ V7 classes for Java ou WebSphere MQ V7 classes for JMS .

Liaisons, puis mode client

Il s'agit de l'option par défaut. Lors de la connexion à un gestionnaire de files d'attente dans ce mode, une application WebSphere MQ classes for JMS tente de se connecter en mode liaisons, ce qui nécessite que le gestionnaire de files d'attente réside sur la même machine que l'application. Si la connexion échoue, l'application tente de se connecter en mode client, ce qui permet au gestionnaire de files d'attente de résider localement sur la même machine que l'application ou à distance.

Configuration de votre gestionnaire de files d'attente de sorte que les classes WebSphere MQ pour les applications JMS puissent se connecter en mode client

Pour configurer votre gestionnaire de files d'attente de sorte que les classes WebSphere MQ pour les applications JMS puissent se connecter en mode client, vous devez créer une définition de canal de connexion serveur et démarrer un programme d'écoute.

Sous z/OS, la fonction de connexion client doit être installée.

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)
```

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.

Vous pouvez également créer une définition de canal de connexion serveur à l'aide de IBM WebSphere MQ Explorer, qui s'exécute sous Linux et Windows, ou des panneaux d'opérations et de contrôle sous z/OS.

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.

Sur toutes les plateformes, vous pouvez utiliser la commande MQSC START LISTENER pour démarrer un programme d'écoute mais, sauf sous z/OS, vous devez d'abord créer un objet programme d'écoute à l'aide de la commande MQSC DEFINE LISTENER. Prenons cet exemple :

```
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) PORT(1414)
START LISTENER(LISTENER.TCP)
```

Sur les systèmes UNIX, Linuxet 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.

Vous pouvez également démarrer un programme d'écoute à l'aide de WebSphere MQ Explorer, qui s'exécute sous Linux et Windows, ou des panneaux d'opérations et de contrôle sous z/OS.

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.

Test de vérification de l'installation point à point pour WebSphere MQ classes for JMS

Un programme de test de vérification d'installation point à point (IVT) est fourni avec WebSphere MQ classes for JMS. 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 «Types d'objet JMS», à la page 970.

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.

Un script est fourni pour exécuter le programme IVT. Le script est appelé IVTRun sur les systèmes UNIX and Linux et IVTRun.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation de WebSphere MQ classes for JMS.

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 «Préparation et exécution des exemples de programmes», à la page 115. 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]
```

Aucun script équivalent n'est fourni sur les systèmes z/OS , mais vous pouvez exécuter l'IVT en mode liaisons en appelant directement la classe Java à l'aide de la commande suivante:

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

Le chemin d'accès aux classes doit contenir com.ibm.mqjms.jar.

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 la description de la propriété PROVIDERVERSION dans Propriétés des objets IBM WebSphere 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: WebSphere MQ Client for Java
     JMSXDeliveryCount: 1
     JMS_IBM_PutDate: 20070815
     JMS_IBM_PutTime: 09310400
     JMS_IBM_Format: MQSTR
     JMS_IBM_MsgType: 8
A simple text message from the MQJMSIVT
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

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 JMS WebSphere MQ afin qu'il puisse utiliser le service d'annuaire pour stocker des objets gérés. Pour plus d'informations sur ces prérequis, voir «Prérequis pour WebSphere MQ classes for JMS», à la page 743. Pour plus d'informations sur la configuration de l'outil d'administration JMS WebSphere MQ , voir «Configuration de l'outil d'administration JMS», à la page 966.

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 UNIX and Linux et IVTSetup.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation WebSphere MQ classes for JMS. Pour exécuter le script, entrez la commande suivante:

```
IVTSetup
```

Le script appelle l'outil d'administration JMS WebSphere MQ 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 JMS WebSphere MQ ou WebSphere 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 JMS WebSphere MQ , voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965. Pour plus d'informations sur l'utilisation de WebSphere MQ Explorer, voir l'aide fournie avec WebSphere 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 UNIX and Linux et IVTTidy.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation WebSphere 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 «Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749.
- 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. Pour plus d'informations, voir «Préparation et exécution des exemples de programmes», à la page 115.

Test de vérification de l'installation de publication / abonnement pour WebSphere MQ classes for JMS

Un programme de test de vérification de l'installation (IVT) de publication / abonnement est fourni avec WebSphere 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 «Types d'objet JMS», à la page 970.

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 UNIX and Linux et PSIVTRun.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation WebSphere 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 «Préparation et exécution des exemples de programmes», à la page 115 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 est pertinent uniquement si le paramètre -v spécifie un numéro de version de gestionnaire de files d'attente inférieur à 7 et que vous utilisez WebSphere Event Broker ou WebSphere Message Broker comme courtier de publication / abonnement.

-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 WebSphere 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
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f6d627720202020202001edb14620006706
  JMSTimestamp:     1187182520203
  JMSCorrelationID: ID:414d5120514d5f6d627720202020202001edb14620006704
  JMSDestination:   topic://MQJMS/PSIVT/Information
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXUserID: mwhite
    JMS_IBM_Encoding: 273
    JMS_IBM_PutApplType: 26
    JMSXAppID: QM_mbw
    JMSXDeliveryCount: 1
    JMS_IBM_PutDate: 20070815
```

```
    JMS_IBM_ConnectionID: 414D5143514D5F6D627720202020202001EDB14620006601
    JMS_IBM_PutTime: 12552020
    JMS_IBM_Format: MQSTR
    JMS_IBM_MsgType: 8
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

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 JMS WebSphere MQ afin qu'il puisse utiliser le service d'annuaire pour stocker des objets gérés. Pour plus d'informations sur ces prérequis, voir «Prérequis pour WebSphere MQ classes for JMS», à la page 743. Pour plus d'informations sur la configuration de l'outil d'administration JMS WebSphere MQ , voir «Configuration de l'outil d'administration JMS», à la page 966.

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 UNIX and Linux et IVTSetup.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation WebSphere MQ classes for JMS. Pour exécuter le script, entrez la commande suivante:

```
IVTSetup
```

Le script appelle l'outil d'administration JMS WebSphere MQ 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, connectez-vous à un gestionnaire de files d'attente autre que le gestionnaire de files d'attente par défaut ou utilisez WebSphere Event Broker ou WebSphere Message Broker à la place de la fonction de publication / abonnement intégrée, vous devez utiliser l'outil d'administration JMS WebSphere MQ ou WebSphere MQ Explorer pour modifier les propriétés appropriées de l'objet de fabrique MQTopicConnection. Pour plus d'informations sur l'utilisation de l'outil d'administration JMS WebSphere MQ , voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965. Pour plus d'informations sur l'utilisation de WebSphere MQ Explorer, voir l'aide fournie avec WebSphere 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 UNIX and Linux et IVTTidy.bat sur Windows, et se trouve dans le sous-répertoire `bin` du répertoire d'installation WebSphere 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 «Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749.
- 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 «Préparation et exécution des exemples de programmes», à la page 115.

Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ

Le programme IVT est fourni sous la forme d'un fichier EAR. Pour utiliser le programme, vous devez le déployer et définir des objets en tant que ressources JCA.

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.jmsra.ivt.ear. Ce fichier est installé avec WebSphere MQ classes for JMS dans le même répertoire que le fichier RAR de l'adaptateur de ressources WebSphere MQ , wmq.jmsra.rar. Pour plus d'informations sur l'emplacement d'installation de ces fichiers, voir «Installation de l'adaptateur de ressources WebSphere MQ», à la page 761.

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 WebSphere MQ . Vous pouvez éventuellement utiliser le programme IVT pour vérifier que l'adaptateur de ressources WebSphere MQ a été correctement configuré pour prendre en charge les transactions réparties.

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, mais l'ensemble de propriétés suivant est un exemple simple:

objet ConnectionFactory

```
channel:               SYSTEM.DEF.SVRCONN
hostName:              localhost
port:                  1414
queueManager:          ExampleQM
transportType:         CLIENT
```

Objet de file d'attente

```
baseQueueManagerName:  ExampleQM
baseQueueName:         TEST.QUEUE
```

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 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/
```

Figure 122, à la page 811 affiche la page initiale du programme IVT.

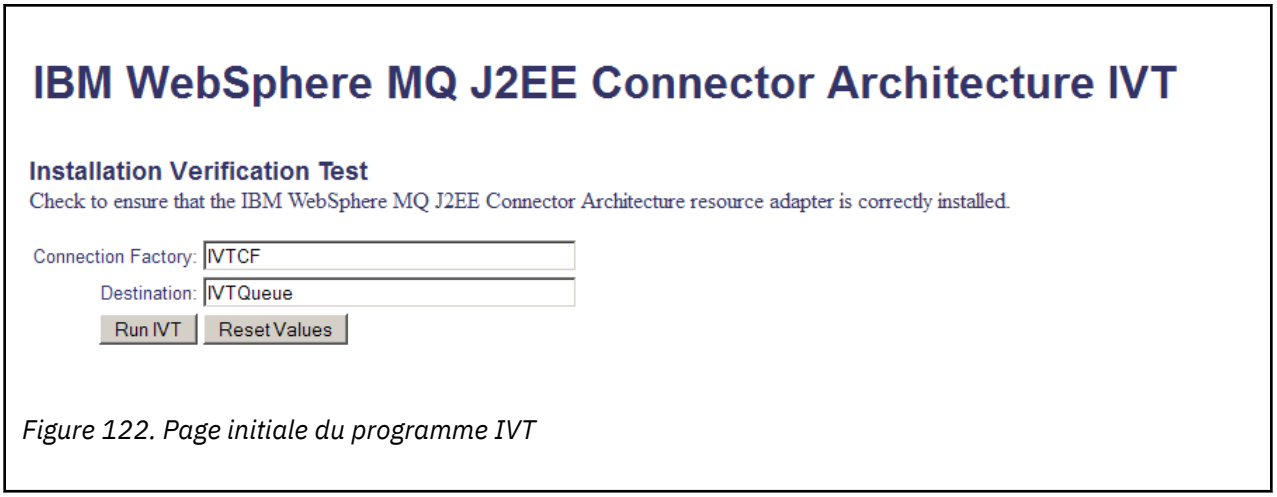


Figure 122. Page initiale du programme IVT

Pour exécuter le test, cliquez sur **Exécuter IVT**. Figure 123, à la page 811 affiche la page qui s'affiche si l'IVT aboutit.

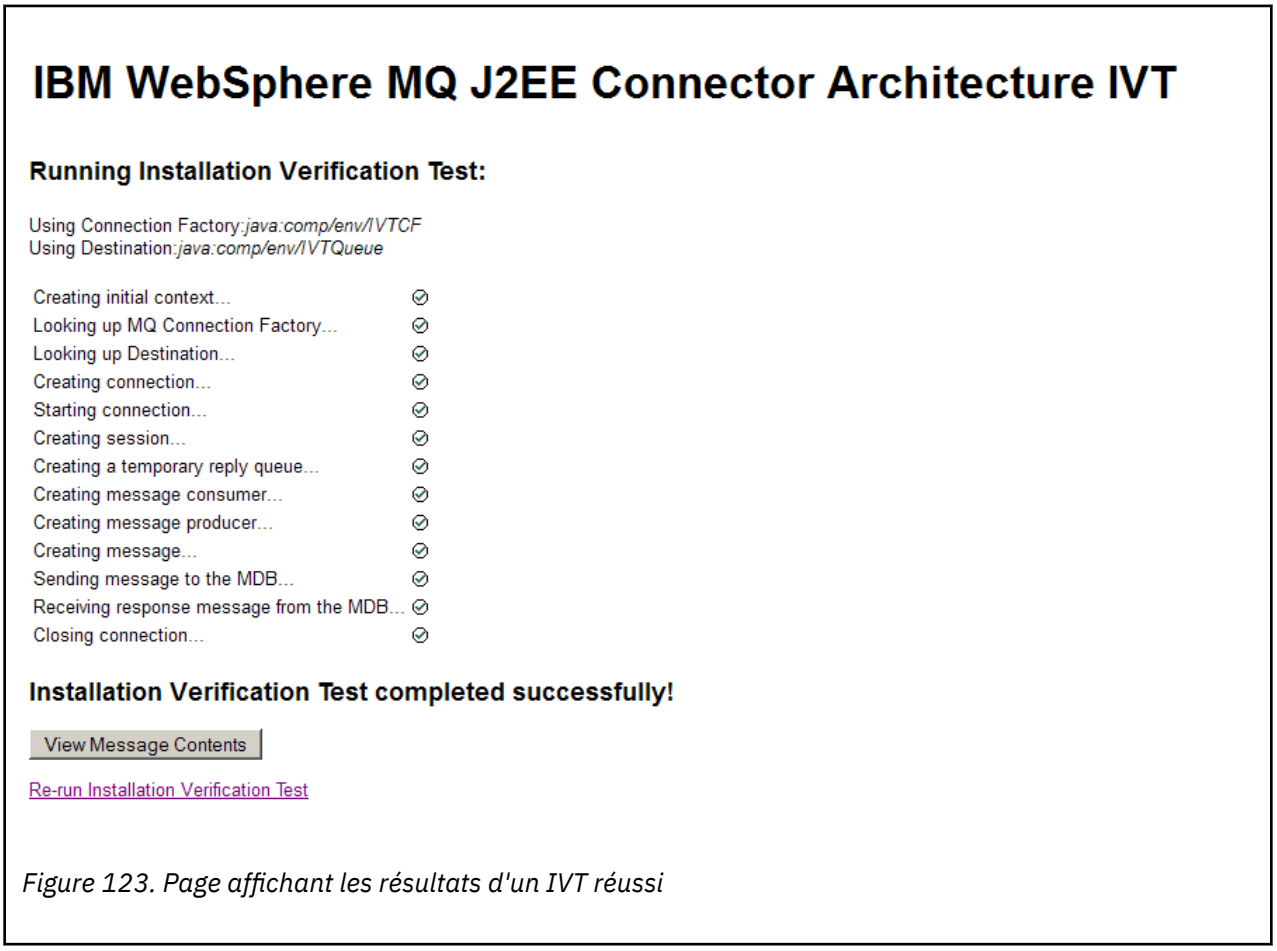


Figure 123. Page affichant les résultats d'un IVT réussi

En cas d'échec de l'IVT, une page similaire à celle de la Figure 124, à la page 812 s'affiche. Pour obtenir plus d'informations sur la cause de l'échec, cliquez sur **Afficher la trace de pile**.

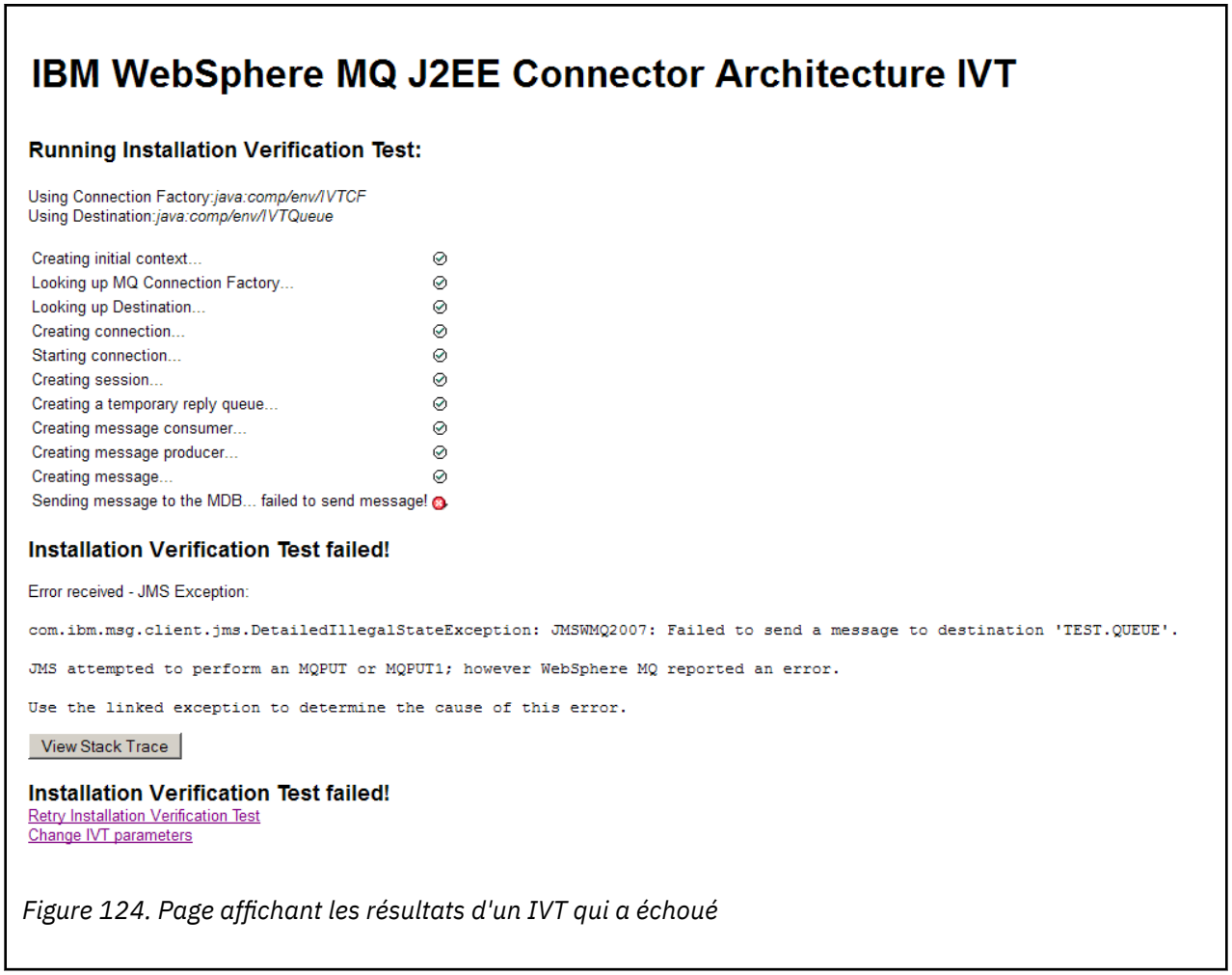


Figure 124. Page affichant les résultats d'un IVT qui a échoué

Pour des instructions détaillées et des informations sur les scripts d'utilitaire fournis pour déployer l'application IVT sur les serveurs d'applications JBoss et WAS CE, voir:

Tâches associées

«Installation et test de l'adaptateur de ressources MQ dans WAS CE», à la page 812
Installation de l'adaptateur de ressources IBM WebSphere MQ et exécution de l'application de test de vérification de l'installation (IVT) dans WebSphere Application Server CE.

«Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816
Après avoir installé l'adaptateur de ressources IBM WebSphere MQ sur JBoss AS 5.1 ou 6, vous pouvez tester l'installation de l'adaptateur de ressources en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Installation et test de l'adaptateur de ressources MQ dans WAS CE

Installation de l'adaptateur de ressources IBM WebSphere MQ et exécution de l'application de test de vérification de l'installation (IVT) dans WebSphere Application Server CE.

Avant de commencer

Cette tâche suppose que vous disposez d'un serveur WebSphere Application Server CE 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 WebSphere MQ sur votre système local et que vous maîtrisez les tâches d'administration standard.

Si vous utilisez l'adaptateur de ressources pour vous connecter à un client IBM WebSphere MQ et que vous devez effectuer des transactions XA distribuées, vous devez suivre les étapes supplémentaires marquées **Client XA uniquement**.

1. Créez un gestionnaire de files d'attente appelé ExampleQMet définissez-le comme décrit dans «Préparation et exécution des exemples de programmes», à la page 115 en notant que le programme d'écoute doit être démarré sur le port 1414, que le canal à utiliser est appelé SYSTEM.DEF.SVRCONN et que la file d'attente utilisée par l'application IVT est nommée TEST.QUEUE. La 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. Si vous souhaitez utiliser un gestionnaire de files d'attente différent, des détails de connexion différents ou une file d'attente différente, voir «Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé», à la page 814.
2. Procurez-vous le fichier d'adaptateur de ressources (wmq.jmsra.rar), l'application IVT (wmq.jmsra.ivt.ear) et les fichiers WAS_CE_jmsra_deployment_plan.xml et WAS_CE_jmsra_ivt_deployment_plan.xml deployment plan files. Pour plus de détails sur l'emplacement de ces fichiers, voir «Installation de l'adaptateur de ressources WebSphere MQ», à la page 761.

Pour une description des liaisons et des connexions en mode client, voir «Modes de connexion pour WebSphere MQ classes for JMS», à la page 800.

Si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou utiliser un autre mode de liaison au lieu du mode client, voir «Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé», à la page 814.

Procédure

1. **Client XA uniquement**: éditez votre copie du fichier WAS_CE_jmsra_deployment_plan.xml .
   a) Recherchez la définition de connexion jms / ivt/IVTCF et modifiez-la de sorte que la fabrique de connexions soit activée pour la transaction XA.
      i) Mettez en commentaire la section NonXA :

      ```
      <conn:xa-transaction>
      ```

      ii) Supprimez la mise en commentaire de la section de configuration XA:

      ```
      <conn:xa-transaction>
          <conn:transaction-caching/>
      </conn:xa-transaction>
      ```

   b) Enregistrez vos modifications.
2. Facultatif : **Client XA uniquement**: modifiez le descripteur d'assemblage du bean géré par message pour qu'il nécessite des transactions. Cela force le bean géré par message de l'IVT à participer à une transaction XA, bien que l'application IVT fonctionne toujours sans cette modification.
   a) Ouvrez le fichier wmq.jmsra.ivt.ear.
   b) Ouvrez le fichier WMQ_IVT_MDB.jar qu'il contient.
   c) Modifier META-INF/ejb-jar.xml.
      i) Mettez en commentaire ou supprimez la ligne dans le descripteur d'assemblage:

      ```
      <trans-attribute>NotSupported</trans-attribute>
      ```

      ii) Supprimez la mise en commentaire de la ligne dans le descripteur d'assemblage:

      ```
      <trans-attribute>Required</trans-attribute>
      ```

      iii) Sauvegardez vos modifications et mettez à jour le fichier dans le fichier WMQ_IVT_MDB.jar .
      iv) Mettez à jour le fichier wmq.jmsra.ivt.ear avec le fichier WMQ_IVT_MDB.jar modifié.

3. Déployez l'adaptateur de ressources sur votre serveur à l'aide du fichier de plan de déploiement modifié.
   a) Pour ce faire, sur la ligne de commande, entrez la commande WAS CE suivante:

```
deploy -u system -p manager deploy wmq.jmsra.rar WAS_CE_jmsra_deployment_plan.xml
```

   b) A l'aide de l'interface d'administration Web, accédez à **Applications > Déployeur:**
      i) Définissez l'archive pour qu'elle soit le fichier wmq.jmsra.rar .
      ii) Définissez le plan comme étant le fichier WAS_CE_jmsra_deployment_plan.xml .
      iii) Vérifiez que l'option'Démarrer l'application après l'installation'est sélectionnée.
      iv) Cliquez sur **Install**.
4. Déployez l'application IVT sur votre serveur à l'aide du plan de déploiement fourni.
   a) Sur la ligne de commande, cette opération peut être effectuée à l'aide de la commande WAS CE suivante:

```
deploy -u system -p manager deploy wmq.jmsra.ivt.ear WAS_CE_jmsra_ivt_deployment_plan.xml
```

   b) A l'aide de l'interface d'administration Web, accédez à **Applications > Déployeur:**
      i) Définissez l' **archive** sur le fichier `wmq.jmsra.ivt.ear` .
      ii) Définissez le **plan** sur le fichier `WAS_CE_jmsra_ivt_deployment_plan.xml` .
      iii) Vérifiez que l'option **Démarrer l'application après l'installation** est sélectionnée.
      iv) Cliquez sur **Install**.
5. Exécutez l'application IVT. Pour plus de détails, voir «Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810. Pour WAS CE, l'URL par défaut est `http://localhost:8080/WMQ_IVT/`.

Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé

Si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou un mode de liaison différent de celui du client, vous devez modifier l'application IVT et les scripts associés dans WebSphere Application Server CE avant de déployer l'adaptateur de ressources ou l'application IVT.

Pourquoi et quand exécuter cette tâche

Si vous souhaitez effectuer un déploiement dans une configuration différente de celle spécifiée dans «Installation et test de l'adaptateur de ressources MQ dans WAS CE», à la page 812, c'est-à-dire si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou un mode de liaison différent à la place du mode client, procédez comme suit avant de déployer l'adaptateur de ressources ou l'application IVT.

Procédure

1. Si vous souhaitez spécifier un autre gestionnaire de files d'attente et une autre file d'attente à utiliser pour l'application IVT, définissez des valeurs pour le gestionnaire de files d'attente et la file d'attente dans `WAS_CE_jmsra_deployment_plan.xml`. Pour plus de détails, voir «Définition de valeurs pour le gestionnaire de files d'attente et la file d'attente», à la page 815.
2. Si vous souhaitez spécifier un gestionnaire de files d'attente et une file d'attente différents dans la configuration du bean géré par message (MDB), définissez des valeurs pour le gestionnaire de files d'attente et la file d'attente que vous utilisez dans `WAS_CE_jmsra_ivt_deployment_plan.xml`. Pour plus de détails, voir «Définition de valeurs pour la configuration MDB», à la page 815.
3. Si vous configurez l'adaptateur de ressources pour qu'il se connecte à IBM WebSphere MQ en mode liaisons, vérifiez que les bibliothèques JNI se trouvent sur le chemin système ou sur le chemin d'accès

à WAS CE. Pour plus d'informations, voir «Installation et test de l'adaptateur de ressources MQ dans WAS CE», à la page 812.

4. Si vous avez déjà déployé l'adaptateur de ressources, vous pouvez le redéployer avec le plan de déploiement modifié pour modifier les paramètres à l'aide de la commande suivante:

```
deploy --user system --password manager redeploy wmq.jmsra.rar
WAS_CE_jmsra_deployment_plan.xml
```

Que faire ensuite

Continuez à déployer l'adaptateur de ressources comme décrit dans «Installation et test de l'adaptateur de ressources MQ dans WAS CE», à la page 812.

Définition de valeurs pour le gestionnaire de files d'attente et la file d'attente

Explique comment définir des valeurs pour le gestionnaire de files d'attente et la file d'attente que vous utilisez dans `WAS_CE_jmsra_deployment_plan.xml`.

Procédure

Dans `WAS_CE_jmsra_deployment_plan.xml`, définissez les valeurs du gestionnaire de files d'attente et de la file d'attente que vous utilisez pour l'application IVT.

Pour la définition de connexion jms / ivt/IVTCF:

1. Définissez la valeur de l'élément queueManager sur le nom de votre gestionnaire de files d'attente.
2. Si vous utilisez une connexion client, définissez la valeur des différents éléments de connexion client de sorte qu'elle soit appropriée pour une connexion à votre gestionnaire de files d'attente.
3. Si vous utilisez une connexion de liaisons:
   a. Définissez la valeur de l'élément transportType sur BINDINGS.
   b. Mettez en commentaire ou supprimez les différents éléments de connexion client.
4. Pour la destination de message jms / ivt/IVTQueue, définissez la valeur de l'élément baseQueueName sur le nom de la file d'attente que vous avez créée pour l'application IVT
5. Enregistrez vos modifications.

Définition de valeurs pour la configuration MDB

Explique comment définir des valeurs pour la configuration MDB dans `WAS_CE_jmsra_deployment_plan.xml`.

Procédure

Dans `WAS_CE_jmsra_ivt_deployment_plan.xml` , définissez les valeurs du gestionnaire de files d'attente et de la file d'attente que vous utilisez dans la configuration du bean géré par message.

Pour le bean géré par message WMQ_IVT_MDB:

1. Définissez la valeur de l'élément queueManager sur le nom de votre gestionnaire de files d'attente.
2. Si vous utilisez une connexion client, définissez la valeur des différents éléments de connexion client de sorte qu'elle soit appropriée pour une connexion à votre gestionnaire de files d'attente.
3. Si vous utilisez une connexion de liaisons:
   a. Définissez la valeur de l'élément transportType sur BINDINGS.
   b. Mettez en commentaire ou supprimez les différents éléments de connexion client.
4. Enregistrez vos modifications.

Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6

Après avoir installé l'adaptateur de ressources IBM WebSphere MQ sur JBoss AS 5.1 ou 6, vous pouvez tester l'installation de l'adaptateur de ressources en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Avant de commencer

Important : Ces instructions concernent JBoss AS 5.1 et 6, elles ne sont pas valides pour JBoss AS 7.

Pour plus d'informations sur l'installation de l'adaptateur de ressources dans JBoss EAP 6.3, voir «Installation et test de l'adaptateur de ressources dans JBoss EAP 6.3», à la page 818.

Cette tâche suppose que vous disposez d'un serveur JBoss en cours d'exécution et que vous êtes familiarisé avec les tâches d'administration standard associées. Cette tâche suppose également que vous disposez d'une installation IBM WebSphere MQ sur votre système local et que vous maîtrisez les tâches d'administration standard.

Si vous utilisez l'adaptateur de ressources pour vous connecter à un client IBM WebSphere MQ et que vous devez effectuer des transactions XA réparties, vous devez suivre les étapes supplémentaires marquées **Client XA uniquement**. Pour une description des liaisons et des connexions en mode client, voir «Modes de connexion pour WebSphere MQ classes for JMS», à la page 800.

Procédure

1. Créez un gestionnaire de files d'attente appelé ExampleQMet définissez-le comme décrit dans «Préparation et exécution des exemples de programmes», à la page 115.

   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.

   Si vous souhaitez utiliser un gestionnaire de files d'attente différent, des détails de connexion différents ou une file d'attente différente, voir «Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé», à la page 814.
2. Procurez-vous le fichier d'adaptateur de ressources (`wmq.jmsra.rar`), l'application IVT (`wmq.jmsra.ivt.ear`) et le fichier `jboss-jmsra-ds.xml` .

   Pour connaître l'emplacement de ces fichiers, voir «Installation de l'adaptateur de ressources WebSphere MQ», à la page 761.
3. **Client XA uniquement**: éditez le fichier `jboss-jmsra-ds.xml` pour activer les transactions XA sur la fabrique de connexions.
   a) Mettez en commentaire ou supprimez la ligne dans la définition de fabrique de connexions `<local-transaction/>`.
   b) Supprimez la mise en commentaire de la ligne dans la définition de fabrique de connexions `<xa-transaction/>`.
   c) Enregistrez vos modifications.
4. **Client XA uniquement**: (facultatif) Modifiez le descripteur d'assemblage du bean géré par message pour exiger des transactions. Cela force le bean géré par message de l'IVT à participer à une transaction XA, bien que l'application IVT fonctionne toujours sans cette modification.
   a) Ouvrez le fichier `wmq.jmsra.ivt.ear`.
   b) Ouvrez le fichier `WMQ_IVT_MDB.jar` qu'il contient.
   c) Editez `META-INF/ejb-jar.xml` :

i) Mettez en commentaire ou supprimez la ligne dans le descripteur d'assemblage:

```
<trans-attribute>NotSupported</trans-attribute>
```

ii) Supprimez la mise en commentaire de la ligne dans le descripteur d'assemblage:

```
<trans-attribute>Required</trans-attribute>
```

iii) Sauvegardez vos modifications et mettez à jour le fichier dans le fichier `WMQ_IVT_MDB.jar` .

iv) Mettez à jour le fichier `wmq.jmsra.ivt.ear` avec le fichier `WMQ_IVT_MDB.jar`modifié.

5. Déployez l'adaptateur de ressources sur votre serveur en copiant le fichier `wmq.jmsra.rar` dans le répertoire `jboss/server/default/deploy`.
6. Créez les ressources JMS requises pour l'application IVT en copiant le fichier `jboss-jmsra-ds.xml` dans le répertoire `jboss/server/default/deploy`.
7. Déployez l'application IVT en copiant le fichier `wmq.jmsra.ivt.ear` dans le répertoire `jboss/server/default/deploy`.
8. Exécutez l'application IVT. Pour plus de détails, voir «Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810. Pour JBoss, l'URL par défaut est `http://localhost:8080/WMQ_IVT/`.

Déploiement de l'application IVT dans JBoss avec un environnement IBM WebSphere MQ personnalisé

Lors de l'installation de l'adaptateur de ressources IBM WebSphere MQ dans JBoss, si vous souhaitez utiliser une file d'attente, un gestionnaire de files d'attente, un port, un hôte, un canal ou utiliser un mode de liaison différent à la place du mode client, vous devez d'abord modifier l'application IVT et les scripts associés dans JBoss avant de déployer l'adaptateur de ressources ou l'application IVT.

Pourquoi et quand exécuter cette tâche

Important : Ces instructions s'appliquent uniquement à Java EE versions 6 et 5, et non à Java EE version 7. L'utilisation de ces instructions pour JBoss version 8 (WildFly) n'est donc pas prise en charge.

Si vous souhaitez effectuer un déploiement dans une configuration différente de celle spécifiée dans «Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816, c'est-à-dire si vous souhaitez utiliser un gestionnaire de files d'attente, une file d'attente, un port, un hôte, un canal ou un mode de liaison différent à la place du mode client, procédez comme suit avant de déployer l'adaptateur de ressources ou l'application IVT.

Procédure

1. Si vous souhaitez spécifier un autre gestionnaire de files d'attente et une autre file d'attente à utiliser pour l'application IVT, définissez des valeurs pour le gestionnaire de files d'attente et la file d'attente.
   a) Pour la définition de connexion jms / ivt/IVTCF:
      i) Définissez la valeur de la propriété config-property queueManager sur le nom de votre gestionnaire de files d'attente.
      ii) Si vous utilisez une connexion client, définissez la valeur des différents éléments de connexion client de sorte qu'elle soit appropriée pour une connexion à votre gestionnaire de files d'attente.
      iii) Si vous utilisez une connexion de liaisons, définissez la valeur de l'élément transportType sur BINDINGS, puis mettez en commentaire ou supprimez les différents éléments de connexion client.
   b) Pour le mbean jms / ivt/IVTQueue, définissez la valeur de l'élément baseQueueName sur le nom de la file d'attente que vous avez créée pour l'application IVT.
   c) Enregistrez vos modifications.

2. Si vous souhaitez spécifier un gestionnaire de files d'attente et une file d'attente différents dans la configuration du bean géré par message, modifiez la configuration du bean géré par message pour qu'il se connecte au gestionnaire de files d'attente et à la file d'attente.
   a) Ouvrez le fichier `wmq.jmsra.ivt.ear`.
   b) Ouvrez le fichier `WMQ_IVT_MDB.jar` qui s'y trouve.
   c) Editez `META-INF/ejb-jar.xml` :
      i) Définissez la valeur de queueManager activation-config-property sur le nom de votre gestionnaire de files d'attente.
      ii) Si vous utilisez une connexion client, définissez la valeur des différentes propriétés d'activation de la connexion client-config-afin qu'elle soit appropriée pour une connexion à votre gestionnaire de files d'attente.
      iii) Si vous utilisez une connexion de liaisons, définissez la valeur de la propriété transportType activation-config-property sur BINDINGS, puis mettez en commentaire ou supprimez les différents éléments de connexion client.
   d) Sauvegardez les modifications et mettez à jour le fichier dans le fichier `WMQ_IVT_MDB.jar` .
   e) Mettez à jour le fichier `wmq.jmsra.ivt.ear` avec le fichier `WMQ_IVT_MDB.jar`modifié.
3. Si vous configurez l'adaptateur de ressources pour qu'il se connecte à IBM WebSphere MQ en mode liaisons, vérifiez que les bibliothèques JNI se trouvent dans le chemin système ou dans le chemin d'accès à JBoss. Pour des détails, voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 751.

Que faire ensuite

Continuez à déployer l'adaptateur de ressources comme décrit dans «Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816.

.

Installation et test de l'adaptateur de ressources dans JBoss EAP 6.3

Après avoir installé l'adaptateur de ressources IBM WebSphere MQ dans JBoss Enterprise Application Platform (EAP) 6.3, sur un serveur autonome ou sur un serveur s'exécutant dans un domaine géré, vous pouvez tester l'installation de l'adaptateur de ressources en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Pourquoi et quand exécuter cette tâche

Important : Ces instructions s'appliquent uniquement à JBoss EAP 6.3 . Pour plus d'informations sur l'installation de l'adaptateur de ressources dans JBoss AS 5.1 et 6, voir «Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816.

Cette tâche suppose que vous disposez d'un serveur JBoss en cours d'exécution et que vous êtes familiarisé avec les tâches d'administration standard associées. Cette tâche suppose également que vous disposez d'une installation IBM WebSphere MQ sur votre système local et que vous maîtrisez l'administration standard.

Procédure

1. Créez un gestionnaire de files d'attente appelé ExampleQMet définissez-le comme décrit dans «Préparation et exécution des exemples de programmes», à la page 115.

   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.

Si vous souhaitez utiliser un gestionnaire de files d'attente différent, des détails de connexion différents ou une file d'attente différente, voir «Déploiement de l'application IVT sur WAS CE avec un environnement MQ personnalisé», à la page 814.

2. Procurez-vous le fichier d'adaptateur de ressources (`wmq.jmsra.rar`) et l'application IVT (`wmq.jmsra.ivt.ear`).

   Pour connaître l'emplacement de ces fichiers, voir «Installation de l'adaptateur de ressources WebSphere MQ», à la page 761.
3. Installez l'adaptateur de ressources, puis testez l'installation en exécutant l'application de test de vérification de l'installation (IVT):
   - Si vous installez l'adaptateur de ressources sur un serveur autonome, voir «Installation et test sur un serveur autonome», à la page 819.
   - Si vous installez l'adaptateur de ressources sur un serveur s'exécutant dans un domaine géré, voir «Installation et test sur un serveur s'exécutant dans un domaine géré», à la page 820.

Installation et test sur un serveur autonome

Après avoir installé l'adaptateur de ressources IBM WebSphere MQ sur JBoss EAP 6.3 sur un serveur autonome, vous pouvez tester l'installation de l'adaptateur de ressources en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Pourquoi et quand exécuter cette tâche

Les informations de cette tâche concernent l'installation et le test de l'adaptateur de ressources sur un serveur autonome. Si vous installez l'adaptateur de ressources sur un serveur s'exécutant dans un domaine géré, voir «Installation et test sur un serveur s'exécutant dans un domaine géré», à la page 820.

Important : Ces instructions s'appliquent uniquement à JBoss EAP 6.3 . Pour plus d'informations sur l'installation de l'adaptateur de ressources dans JBoss AS 5.1 et 6, voir «Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816.

Procédure

1. Déployez l'adaptateur de ressources sur votre serveur en copiant le fichier `wmq.jmsra.rar` dans le répertoire <*EAP_HOME*>`/standalone/deployments`.
2. Créez les ressources JMS requises pour l'application IVT en ajoutant les entrées suivantes à la section `<resource-adapters>` du fichier <*EAP_HOME*>`/standalone/configuration/standalone-full.xml` :

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
     <connection-definition
          class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
          jndi-name="java:jboss/jms/ivt/IVTCF"
          enabled="true"
          use-java-context="true"
          pool-name="IVTCF">
       <config-property name="port">
           1414
       </config-property>
       <config-property name="hostName">
           localhost
       </config-property>
       <config-property name="channel">
           SYSTEM.DEF.SVRCONN
       </config-property>
       <config-property name="transportType">
           CLIENT
       </config-property>
```

```
       <config-property name="queueManager">
           ExampleQM
       </config-property>
     </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
          TEST.QUEUE
      </config-property>
    </admin-object>
  </admin-objects>
</resource-adapter>
```

3. Ajoutez les informations suivantes aux paramètres de démarrage de votre serveur d'applications:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Déployez l'application IVT en copiant le fichier `wmq.jmsra.ivt.ear` dans le répertoire <*EAP_HOME*>`/standalone/deployments`.
5. Démarrez le serveur d'applications.
6. Exécutez l'application IVT.

   Pour plus d'informations, voir «Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810. Pour JBoss, l'URL par défaut est `http://localhost:8080/WMQ_IVT/`.

   Remarque : Les noms JNDI utilisés pour les ressources JMS requises pour l'exécution de l'application IVT sont les suivants:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   Lorsque vous lancez l'application IVT à l'aide de l'URL spécifiée ci-dessus, entrez les noms JNDI de ces ressources dans leurs zones respectives, puis exécutez l'application en cliquant sur **Exécuter IVT**.

Installation et test sur un serveur s'exécutant dans un domaine géré

Après avoir installé l'adaptateur de ressources IBM WebSphere MQ sur JBoss EAP 6.3 sur un serveur s'exécutant dans un domaine géré, vous pouvez tester l'installation de l'adaptateur de ressources en installant et en exécutant l'application de test de vérification de l'installation (IVT).

Pourquoi et quand exécuter cette tâche

Les informations de cette tâche concernent l'installation et le test de l'adaptateur de ressources sur un serveur s'exécutant dans un domaine géré. Si vous installez l'adaptateur de ressources sur un serveur autonome, voir «Installation et test sur un serveur autonome», à la page 819.

Important : Ces instructions s'appliquent uniquement à JBoss EAP 6.3 . Pour plus d'informations sur l'installation de l'adaptateur de ressources dans JBoss AS 5.1 et 6, voir «Installation et test de l'adaptateur de ressources dans JBoss AS 5.1 et 6», à la page 816.

Procédure

1. Déployez l'adaptateur de ressources sur votre serveur à l'aide de la console de gestion JBoss ou de l'interface de ligne de commande de gestion.
2. Créez les ressources JMS requises pour l'application IVT en ajoutant les entrées suivantes à la section `<resource-adapters>` du fichier `<EAP_HOME>/domain/configuration/domain.xml file`:

```
<resource-adapter id="wmq.jmsra">
  <archive>
      wmq.jmsra.rar
  </archive>
```

```
  <transaction-support>NoTransaction</transaction-support>
  <connection-definitions>
    <connection-definition
         class-name="com.ibm.mq.connector.outbound.ManagedConnectionFactoryImpl"
         jndi-name="java:jboss/jms/ivt/IVTCF"
         enabled="true"
         use-java-context="true"
         pool-name="IVTCF">
      <config-property name="port">
          1414
      </config-property>
      <config-property name="hostName">
          localhost
      </config-property>
      <config-property name="channel">
          SYSTEM.DEF.SVRCONN
      </config-property>
      <config-property name="transportType">
          CLIENT
      </config-property>
      <config-property name="queueManager">
          ExampleQM
      </config-property>
    </connection-definition>
  </connection-definitions>
  <admin-objects>
    <admin-object class-name="com.ibm.mq.connector.outbound.MQQueueProxy"
                  jndi-name="java:jboss/jms/ivt/IVTQueue"
                  pool-name="IVTQueue">
      <config-property name="baseQueueName">
          TEST.QUEUE
      </config-property>
    </admin-object>
  </admin-objects>
</resource-adapter>
```

3. Ajoutez les informations suivantes aux paramètres de démarrage de votre serveur d'applications:

```
-Dcom.ibm.mq.connector.IVTMDBCFJNDIName=java:jboss/jms/ivt/IVTCF
```

4. Arrêtez et redémarrez le serveur d'applications.
5. Déployez l'application IVT à l'aide de la console de gestion JBoss ou de l'interface de ligne de commande de gestion.
6. Exécutez l'application IVT.

   Pour plus d'informations, voir «Programme de test de vérification de l'installation de l'adaptateur de ressources WebSphere MQ», à la page 810. Pour JBoss, l'URL par défaut est `http://localhost:8080/WMQ_IVT/`.

   Remarque : Les noms JNDI utilisés pour les ressources JMS requises pour l'exécution de l'application IVT sont les suivants:

```
Connection Factory : IVTCF
JNDI name          : java:jboss/jms/ivt/IVTCF

Destination        : IVTQueue
JNDI name          : java:jboss/jms/ivt/IVTQueue
```

   Entrez les noms JNDI de ces ressources dans leurs zones respectives lorsque vous lancez l'application IVT à l'aide de l'URL spécifiée ci-dessus, puis exécutez l'application en cliquant sur **Exécuter IVT**.

Scripts fournis avec WebSphere MQ classes for JMS

Un certain nombre de scripts sont fournis pour vous aider à exécuter les tâches courantes qui doivent être effectuées lors de l'utilisation des classes WebSphere MQ pour JMS.

Tableau 102, à la page 822 répertorie tous les scripts et leurs utilisations. Les scripts se trouvent dans le sous-répertoire `bin` du répertoire d'installation WebSphere MQ classes for JMS.

Tableau 102. Scripts fournis avec WebSphere MQ classes for JMS	
Fournisseur	**Utiliser**
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 «Test de vérification de l'installation point à point pour WebSphere MQ classes for JMS», à la page 802.
JMSAdmin[1]	Exécute l'outil d'administration JMS WebSphere MQ , comme décrit dans «Appel de l'outil d'administration IBM WebSphere MQ classes for JMS», à la page 965.
JMSAdmin.config	Fichier de configuration de l'outil d'administration JMS WebSphere MQ , comme décrit dans «Configuration de l'outil d'administration JMS», à la page 966.
PSIVTRun[1]	Exécute le programme de test de vérification de l'installation de publication / abonnement, comme décrit dans «Test de vérification de l'installation de publication / abonnement pour WebSphere MQ classes for JMS», à la page 806.
PSReportDump.class	Cette classe est gérée à des fins de compatibilité avec les éditions précédentes, mais n'effectue aucune fonction.
setjmsenv	Définit les variables d'environnement pour l'exécution d'une application WebSphere MQ classes for JMS dans une machine virtuelle Java (JVM) 32 bits sur les systèmes UNIX and Linux , comme décrit dans «Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749.
setjmsenv64	Définit les variables d'environnement pour l'exécution d'une application WebSphere MQ classes for JMS dans une machine virtuelle Java 64 bits sur des systèmes UNIX and Linux , comme décrit dans «Variables d'environnement utilisées par IBM WebSphere MQ classes for JMS», à la page 749.
Remarque : 1. Sous Windows, le nom de fichier porte l'extension .bat.	

Prise en charge d'OSGi

OSGi fournit une infrastructure qui prend en charge le déploiement d'applications en tant que bundles. Neuf bundles OSGi sont fournis avec IBM WebSphere MQ classes for JMS .

OSGi fournit une infrastructureJava 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.

IBM WebSphere MQ classes for JMS. inclut les bundles OSGi suivants.

com.ibm.msg.client.osgi.jms< numéro de version > .jar
Couche de code commune dans le IBM WebSphere MQ classes for JMS. Pour plus d'informations sur l'architecture en couches des classes WebSphere MQ pour JMS, voir «Une architecture en couches», à la page 826.

com.ibm.msg.client.osgi.jms.prereq_< numéro de version > .jar
Fichiers d'archive Java (JAR) prérequis pour la couche commune.

com.ibm.msg.client.osgi.commonservices.j2se_<version > .jar
Services communs pour les applications Java Platform, Standard Edition (Java SE).

com.ibm.msg.client.osgi.nls_< numéro de version > .jar
Messages pour la couche commune.

com.ibm.msg.client.osgi.wmq_< numéro de version > .jar
Le fournisseur de messagerie IBM WebSphere MQ dans IBM WebSphere MQ classes for JMS. Pour plus d'informations sur l'architecture en couches de IBM WebSphere MQ classes for JMS , voir «Une architecture en couches», à la page 826.

com.ibm.msg.client.osgi.wmq.prereq_< numéro de version > .jar
Fichiers JAR prérequis pour le fournisseur de messagerie IBM WebSphere MQ .

com.ibm.msg.client.osgi.wmq.nls_< numéro de version > .jar
Messages du fournisseur de messagerie IBM WebSphere MQ .

com.ibm.mq.osgi.directip_< numéro de version > .jar
Fichiers JAR permettant au fournisseur de messagerie IBM WebSphere MQ de créer une connexion en temps réel à un courtier.

où < numéro de version > est le numéro de version de WebSphere MQ qui a été installé.

Les bundles sont installés dans le sous-répertoire `java/lib/OSGi` de votre installation WebSphere MQ ou dans le dossier `java\lib\OSGi` sous Windows.

Le bundle com.ibm.mq.osgi.java < numéro de version > .jar, qui est également installé dans le sous-répertoire `java/lib/OSGi` de votre installation WebSphere MQ ou dans le dossier `java\lib\OSGi` sous Windows, fait partie des classes WebSphere MQ pour Java. Ce bundle ne doit pas être chargé dans un environnement d'exécution OSGi dans lequel les classes WebSphere MQ pour JMS sont chargées.

Les bundles OSGi des classes WebSphere MQ pour 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 WebSphere MQ classes for JMS, les rubriques temporaires ne fonctionnent pas. En outre, les classes d'exit de canal écrites dans Java ne sont pas prises en charge en raison d'un problème inhérent au chargement de classes dans un environnement de chargeur de classes multiples tel que OSGi. Un bundle d'utilisateurs peut connaître les classes IBM WebSphere MQ pour les bundles JMS, mais les bundles IBM WebSphere MQ classes for JMS ne connaissent aucun bundle d'utilisateurs. Par conséquent, le chargeur de classe utilisé dans un bundle IBM WebSphere 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 .

Résolution des problèmes liés aux classes IBM WebSphere MQ pour JMS

Vous pouvez examiner les problèmes en exécutant les programmes de vérification de l'installation et en utilisant les fonctions de trace et de journal.

Si un programme ne s'exécute pas correctement, exécutez l'un des programmes de vérification de l'installation, comme décrit dans «Test de vérification de l'installation point à point pour WebSphere MQ classes for JMS», à la page 802 et «Test de vérification de l'installation de publication / abonnement pour WebSphere MQ classes for JMS», à la page 806, et suivez les conseils donnés dans les messages de diagnostic.

Consignation etIBM WebSphere MQ classes for JMS

Par défaut, la sortie du journal est envoyée au fichier `mqjms.log` . Vous pouvez le rediriger vers un fichier ou un répertoire spécifique.

La fonction de consignation IBM WebSphere MQ classes for JMS est fournie pour signaler les problèmes graves, en particulier les problèmes qui peuvent indiquer des erreurs de configuration plutôt que des erreurs de programmation. Par défaut, la sortie de journal est envoyée au fichier `mqjms.log` dans le répertoire de travail de la machine virtuelle Java.

Vous pouvez rediriger la sortie du journal vers un autre fichier en définissant la propriété com.ibm.msg.client.commonservices.log.outputName. La valeur de cette propriété peut être:

- Nom de chemin unique.
- Liste de noms de chemin séparés par des virgules (toutes les données sont consignées dans tous les fichiers).

Chaque nom de chemin peut être:

- Absolu ou relatif.
- `stderr` ou `System.err` pour représenter le flux d'erreur standard.
- `sttdout` ou `System.out` pour représenter le flux de sortie standard.

Si la valeur de la propriété identifie un répertoire, la sortie du journal est écrite dans `mqjms.log` dans ce répertoire. Si la valeur de la propriété identifie un fichier spécifique, la sortie du journal est écrite dans ce fichier.

Vous pouvez définir cette propriété dans le fichier de configuration IBM WebSphere MQ classes for JMS ou en tant que propriété système dans la commande **java** . Dans l'exemple suivant, la propriété est définie en tant que propriété système et identifie un fichier spécifique:

```
java -Djava.library.path=library_path
     -Dcom.ibm.msg.client.commonservices.log.outputName=/mydir/mylog.txt
      MyAppClass
```

Dans la commande, *chemin_bibliothèque* est le chemin d'accès au répertoire contenant les bibliothèques IBM WebSphere MQ classes for JMS (voir «Configuration des bibliothèques JNI (Java Native Interface)», à la page 751).

Vous pouvez désactiver la sortie de journal en définissant la propriété com.ibm.msg.client.commonservices.log.status sur OFF. La valeur par défaut de cette propriété est ON.

Les valeurs `System.err` et `System.out` peuvent être définies pour envoyer une sortie de journal aux flux `System.err` et `System.out` .

Introduction à WebSphere MQ classes for JMS, pour les programmeurs

WebSphere MQ classes for JMS est le fournisseur JMS fourni avec WebSphere MQ. WebSphere MQ classes for JMS implémente les interfaces définies dans le package javax.jms et fournit également deux ensembles d'extensions à l'API JMS. Les applications Java Platform, Standard Edition (Java SE) et Java Platform, Enterprise Edition (Java EE) peuvent utiliser WebSphere MQ classes for JMS.

La spécification JMS définit un ensemble d'interfaces que les applications peuvent utiliser pour effectuer des opérations de messagerie. La version la plus récente de la spécification est la version 1.1. Le package javax.jms spécifie les détails des interfaces JMS et un fournisseur JMS implémente ces interfaces pour un produit de messagerie spécifique. WebSphere MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour WebSphere MQ.

Le flux logique au sein d'une application JMS commence par les objets ConnectionFactory et Destination. L'application utilise un objet ConnectionFactory pour créer un objet Connection, qui représente la connexion active depuis l'application vers un serveur de messagerie. L'application utilise l'objet Connection pour créer un objet Session, qui est un contexte avec une seule unité d'exécution permettant de générer et de consommer des messages. L'application peut ensuite utiliser l'objet Session et l'objet Destination pour créer un objet MessageProducer, que l'application utilise pour envoyer des messages

à la destination indiquée. La destination est soit une file d'attente, soit une rubrique du système de messagerie et est encapsulée par l'objet Destination. L'application peut aussi utiliser l'objet Session et un objet Destination pour créer un objet MessageConsumer, qu'elle utilise pour recevoir des messages envoyés à la destination indiquée.

La spécification JMS s'attend 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 extrait ces objets à l'aide de l'interface JNDI (Java Naming and Directory Interface). Le référentiel des objets gérés peut être un simple fichier ou un annuaire LDAP (Lightweight Directory Access Protocol).

WebSphere MQ classes for JMS prend en charge l'utilisation d'objets gérés. Une application peut utiliser toutes les fonctions de WebSphere MQ qui sont exposées via WebSphere MQ classes for JMS sans qu'aucune information spécifique à WebSphere MQne soit codée en dur dans l'application elle-même. Cette disposition offre à l'application un degré d'indépendance par rapport à la configuration WebSphere MQ sous-jacente. Pour parvenir à cette indépendance, l'application peut utiliser JNDI pour extraire les fabriques de connexions et les destinations stockées sous forme d'objets gérés et utiliser uniquement les interfaces définies dans le package javax.jms pour effectuer des opérations de messagerie. Un administrateur peut utiliser l'outil d'administration JMS WebSphere MQ ou IBM WebSphere MQ Explorer pour créer et gérer des objets gérés dans un référentiel central. Toutefois, un serveur d'applications fournit généralement son propre référentiel pour les objets gérés et ses propres outils pour la création et la maintenance des objets. Une application Java EE peut donc utiliser JNDI pour extraire des objets gérés du référentiel du serveur d'applications ou d'un référentiel central.

WebSphere MQ classes for JMS fournit également des extensions à l'API JMS. Les éditions précédentes de WebSphere MQ classes for JMS contiennent des extensions implémentées dans des objets MQConnectionFactory, MQQueue et MQTopic. Ces objets possèdent des propriétés et des méthodes spécifiques à WebSphere MQ. Les objets peuvent être des objets administrés ou une application peut créer les objets de façon dynamique lors de la phase d'exécution. Cette édition de WebSphere MQ classes for JMS gère ces extensions et vous pouvez continuer à utiliser, sans modification, toutes les applications qui utilisent ces extensions. Ces extensions sont appelées *WebSphere MQ extensions JMS*. Notez que dans cette documentation, les objets créés de manière dynamique par une application lors de la phase d'exécution *ne sont pas* considérés comme des objets gérés.

Outre les extensions JMS WebSphere MQ , cette édition de WebSphere MQ classes for JMS fournit un ensemble plus générique d'extensions à l'API JMS. Ces extensions sont appelées *IBM JMS*et ont les objectifs généraux suivants:

- Pour fournir un niveau de cohérence plus élevé entre les fournisseurs JMS IBM
- 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 JMS IBM à un autre

L'objectif principal de ces extensions concerne la création et la configuration dynamique des fabriques de connexions et des destinations lors de l'exécution, mais les extensions fournissent également des fonctions qui ne sont pas directement liées à la messagerie, telles que la fonction de détermination des incidents.

Une fabrique de connexions, une file d'attente ou un objet de rubrique créé à l'aide de l'interface javax.jms ou d'un ensemble d'extensions JMS peut être adressé à l'aide de l'une de ces API, c'est-à-dire qu'il peut être transtypé vers n'importe quelle interface. Pour conserver la portabilité des applications au plus haut niveau, utilisez l'API la plus générique qui convient à vos exigences.

Les applications Java SE et Java EE peuvent utiliser WebSphere MQ classes for JMS. Sur la plateforme Java EE , WebSphere MQ classes for JMS prend en charge deux types de communication entre un composant d'une application et un gestionnaire de files d'attente WebSphere MQ :

Communication sortante
A l'aide de l'API JMS directement, un composant d'application crée une connexion à un gestionnaire de files d'attente, puis envoie et reçoit des messages.

Par exemple, le composant d'application peut être un client d'application, un servlet, une page JavaServer (JSP), un bean enterprise Java (EJB) ou un bean géré par message (MDB). Dans ce type de communication, le conteneur du serveur d'applications fournit uniquement des fonctions de bas

niveau pour le support des opérations de messagerie, comme le regroupement de connexions et la gestion des unités d'exécution.

Communication entrante
Un message qui arrive à destination est distribué à un bean géré par message, qui le traite.

Les applications Java EE utilisent des beans gérés par message pour traiter les messages de manière asynchrone. Un bean géré par message agit en tant que programme d'écoute de message JMS et est implémenté par une méthode onMessage(), qui définit comment un message est traité. Un bean géré par message est déployé dans le conteneur d'EJB d'un serveur d'applications. La manière selon laquelle un bean géré par message est configuré varie en fonction du serveur d'applications utilisé, mais les informations de configuration doivent spécifier à quel gestionnaire de files d'attente se connecter, comment se connecter à ce dernier, quelle destination surveiller pour les messages et le comportement transactionnel du bean géré par message. Ces informations sont ensuite utilisées par le conteneur d'EJB. Lorsqu'un message répondant aux critères de sélection du bean géré par message arrive à la destination spécifiée, le conteneur d'EJB utilise WebSphere MQ classes for JMS pour extraire le message du gestionnaire de files d'attente, puis le distribue au bean géré par message en appelant sa méthode onMessage().

Classes IBM WebSphere MQ pour architecture JMS

IBM WebSphere MQ classes for JMS, tel qu'il est fourni dans IBM WebSphere MQ version 7.0et les éditions suivantes, contient un certain nombre d'améliorations par rapport aux éditions précédentes. Certaines de ces améliorations résultent de modifications apportées à l'implémentation de IBM WebSphere MQ classes for JMS et d'autres résultent de l'exploitation par IBM WebSphere MQ classes for JMS des modifications apportées à la fonction IBM WebSphere MQ sous-jacente.

Les sections suivantes résument les principales améliorations.

Une architecture en couches

Dans les éditions précédentes de WebSphere MQ, l'implémentation de WebSphere MQ classes for JMS était entièrement spécifique à WebSphere MQ. D'autres produits IBM qui fournissent des systèmes de messagerie ont également inclus des fournisseurs JMS, mais ces derniers n'ont que très peu ou rien en commun avec l'implémentation des classes WebSphere MQ for JMS.

Depuis WebSphere MQ V7.0, WebSphere MQ classes for JMS possède une architecture en couches. La couche supérieure du code est une couche commune qui peut être utilisée par n'importe quel fournisseur JMS IBM . Lorsqu'une application appelle une méthode JMS, tout traitement de l'appel qui n'est pas spécifique à un système de messagerie est effectué par la couche commune, qui fournit également une réponse cohérente à l'appel. Les traitements de l'appel spécifiques à un système de messagerie sont délégués à une couche inférieure. La Figure 125, à la page 827 présente l'architecture en couches.

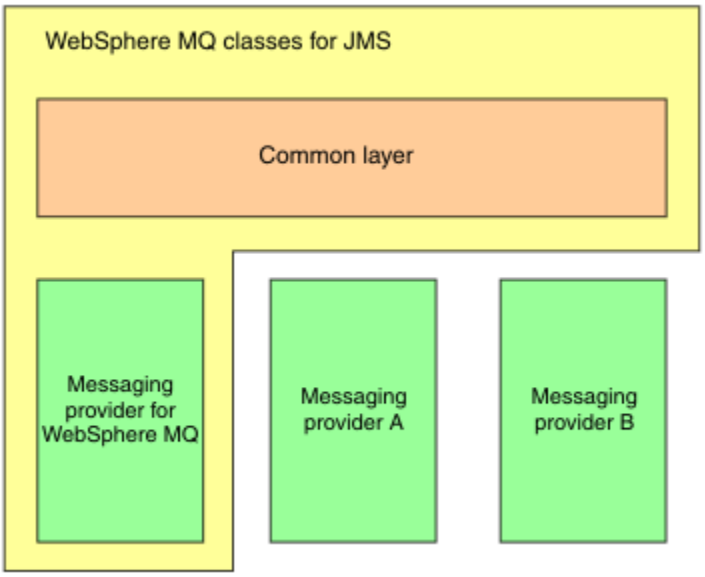


Figure 125. Architecture en couches pour les fournisseurs JMS IBM

Le passage à une architecture en couches a les objectifs suivants:

- Pour améliorer la cohérence du comportement des différents fournisseurs JMS IBM
- 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 JMS IBM à un autre

Cette implémentation des classes WebSphere MQ pour JMS introduit également un nouvel ensemble d'extensions à l'API JMS. Ces extensions sont appelées *IBM JMS*. L'objectif principal de ces extensions concerne la création et la configuration de fabriques de connexions et de destinations de manière dynamique lors de l'exécution.

Une application utilisant les extensions JMS IBM commence par créer un objet de fabrique JmsFactory, en spécifiant comme paramètre une constante qui identifie le système de messagerie choisi. L'application utilise l'objet JmsFactoryFactory pour créer des fabriques de connexions et des destinations comportant les classes spécialisées correctes pour le système de messagerie choisi.

L'application peut ensuite configurer les fabriques de connexions et les destinations en définissant leurs propriétés. Les extensions JMS IBM fournissent un ensemble de méthodes permettant de définir des propriétés. Ces méthodes sont indépendantes de tout système de messagerie. Chaque type de données possède sa propre méthode de définition et chaque propriété est identifiée à l'aide d'un nom défini en tant que membre final statique de la classe WMQConstants. Lorsqu'une application appelle l'une de ces méthodes, un des paramètres de l'appel correspond au nom de la propriété et l'autre à la valeur de la propriété.

Par exemple, si WebSphere MQ est le système de messagerie, l'une des propriétés d'une fabrique de connexions est le nom du gestionnaire de files d'attente auquel se connecter. A l'aide des extensions JMS IBM , une application définit le nom du gestionnaire de files d'attente sur JUPITER en appelant la méthode suivante:

```
JmsConnectionFactory myCF;
...
myCF.setStringProperty(WMQConstants.WMQ_QUEUE_MANAGER, "JUPITER");
```

En revanche, une application peut exécuter la même fonction en appelant la méthode suivante:

```
MQConnectionFactory myCF;
...
myCF.setQueueManager("JUPITER");
```

Cette méthode est une extension JMS WebSphere MQ et est spécifique à WebSphere MQ en tant que système de messagerie. L'utilisation de cette méthode rend donc l'application potentiellement moins facile à porter vers un autre fournisseur IBM JMS.

Relation entre WebSphere MQ classes for JMS et WebSphere MQ classes for Java

Dans les éditions de WebSphere MQ, antérieures à la version 7.0, WebSphere MQ classes for JMS était implémenté presque entièrement en tant que couche de code sur WebSphere MQ classes for Java. Cette disposition a provoqué une certaine confusion parmi les développeurs d'applications car la définition de zones ou l'appel de méthodes dans la classe MQEnvironment peut avoir des effets indésirables et inattendus sur le comportement d'exécution du code écrit à l'aide de WebSphere MQ classes for JMS. En outre, l'implémentation des classes WebSphere MQ pour JMS comportait certaines contraintes dans les zones où l'API JMS ne correspond pas à la configuration naturelle des classes WebSphere MQ pour Java. Ces contraintes ont entraîné des problèmes de performances d'exécution.

A partir de WebSphere MQ V7.0, l'implémentation de WebSphere MQ classes for JMS ne dépend plus de WebSphere MQ classes for Java. WebSphere MQ classes for Java et WebSphere MQ classes for JMS sont désormais des homologues qui utilisent une interface Java commune à l'interface MQI. Cette disposition permet d'optimiser les performances et signifie que la définition de zones ou l'appel de méthodes dans la classe MQEnvironment n'a aucun effet sur le comportement d'exécution du code écrit à l'aide de WebSphere MQ classes for JMS. Figure 126, à la page 828 présente la relation entre WebSphere MQ classes for JMS et WebSphere MQ classes for Java dans les éditions précédentes de WebSphere MQ et dans WebSphere MQ V7.0 et les éditions suivantes.

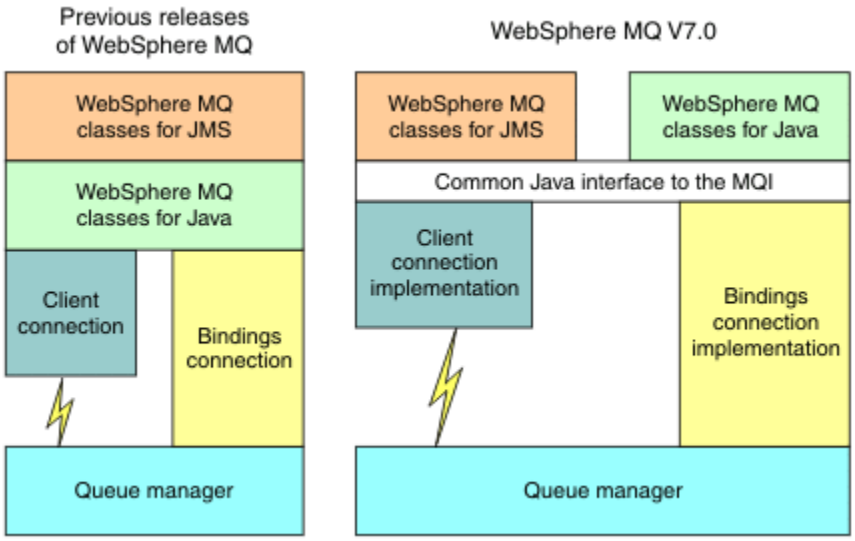


Figure 126. Relation entre WebSphere MQ classes for JMS et WebSphere MQ classes for Java

Afin de maintenir la compatibilité avec les versions antérieures, les classes d'exit de canal écrites en Java peuvent toujours utiliser les classes WebSphere MQ pour les interfaces Java, même si les classes d'exit de canal sont appelées à partir de WebSphere MQ classes for JMS. Toutefois, l'utilisation des interfaces WebSphere MQ pour Java signifie que vos applications dépendent toujours du fichier JAR WebSphere MQ classes for Java, com.ibm.mq.jar. Si vous ne souhaitez pas que com.ibm.mq.jar figure dans votre chemin d'accès aux classes, vous pouvez utiliser le nouvel ensemble d'interfaces dans le package com.ibm.mq.exits à la place.

Vous pouvez désormais créer et configurer des objets gérés JMS avec WebSphere MQ Explorer.

Messagerie de type publication/abonnement

WebSphere MQ V7.0, et les éditions suivantes, contiennent une fonction de publication / abonnement intégrée. Cette fonction remplace WebSphere MQ Publish / Subscribe, qui a été fourni avec WebSphere MQ V6.0.

Les applications WebSphere MQ classes for JMS peuvent utiliser la fonction de publication / abonnement intégrée et l'utiliser au lieu d'utiliser WebSphere Event Broker ou WebSphere Message Broker pour la messagerie de publication / abonnement avec WebSphere MQ comme transport. La configuration de WebSphere MQ classes for JMS pour utiliser la nouvelle fonction est plus simple que la configuration de WebSphere MQ classes for JMS pour utiliser WebSphere MQ Publish / Subscribe, WebSphere Event Broker ou WebSphere Message Broker. Les administrateurs et développeurs d'application n'ont plus besoin de gérer les files d'attente de publication, les files d'attente d'abonnement, les magasins d'abonnement et les nettoyages d'abonné. En outre, les objets ConnectionFactory et Topic possèdent un plus petit nombre de propriétés.

La fonction de publication/abonnement intégrée fournit également quelques fonctionnalités supplémentaires telles que les publications conservées et un choix de deux schémas de caractères génériques permettant de spécifier une série de rubriques auxquelles une application souhaite s'abonner.

Une application peut toujours utiliser une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker pour la messagerie de publication / abonnement. Ce support est inchangé.

Les applications utilisant WebSphere MQ Publish / Subscribe peuvent utiliser la fonction de publication / abonnement intégrée sans modification lors de la mise à niveau du gestionnaire de files d'attente auquel elles sont connectées. Les propriétés définies par une application, mais qui ne sont pas requises par la fonction de publication/abonnement imbriquée, sont ignorées.

Fournisseur de messagerie WebSphere MQ

Le fournisseur de messagerie WebSphere MQ dispose de deux modes de fonctionnement:

- *WebSphere MQ mode normal du fournisseur de messagerie*
- *WebSphere MQ mode de migration du fournisseur de messagerie*

Le mode normal du fournisseur de messagerie WebSphere MQ utilise toutes les fonctions de WebSphere MQ Version 7.0, ainsi que les gestionnaires de files d'attente d'édition suivants pour implémenter JMS. Ce mode est utilisé uniquement pour la connexion à un gestionnaire de files d'attente WebSphere MQ et peut se connecter à WebSphere MQ Version 7.0et aux gestionnaires de files d'attente d'édition suivants en mode client ou en mode liaisons. Ce mode est optimisé pour utiliser la nouvelle fonction WebSphere MQ Version 7.0 et les éditions suivantes.

Le mode de migration du fournisseur de messagerie WebSphere MQ est basé sur la fonction WebSphere MQ Version 6.0 et utilise uniquement les fonctions qui étaient disponibles dans le gestionnaire de files d'attente WebSphere MQ Version 6.0 pour implémenter JMS. Vous pouvez vous connecter à un WebSphere MQ version 7.0 et aux gestionnaires de files d'attente d'édition suivants à l'aide du mode de migration du fournisseur de messagerie WebSphere MQ , mais vous ne pouvez pas utiliser les optimisations de la version 7.0 . Ce mode autorise les connexions à l'une des versions de gestionnaire de files d'attente suivantes :

1. WebSphere MQ Version 7.0, et versions ultérieures, gestionnaire de files d'attente en mode liaisons ou client, mais ce mode utilise uniquement les fonctions qui étaient disponibles pour un gestionnaire de files d'attente WebSphere MQ Version 6.0
2. WebSphere MQ version 6.0 ou antérieure en mode client

Si vous souhaitez vous connecter à WebSphere Event Broker ou WebSphere Message Broker à l'aide de WebSphere MQ Enterprise Transport, utilisez le mode de migration du fournisseur de messagerie WebSphere MQ . Si vous utilisez WebSphere MQ Real-Time Transport, le mode de migration du fournisseur de messagerie WebSphere MQ est automatiquement sélectionné, car vous avez explicitement sélectionné des propriétés dans l'objet de fabrique de connexions. La connexion à WebSphere Event Broker ou WebSphere Message Broker à l'aide de WebSphere MQ Enterprise Transport respecte les règles générales de sélection de mode décrites dans Règles de sélection du mode de fournisseur de messagerie WebSphere MQ .

Consommation de messagerie asynchrone

WebSphere MQ V7.0 et les éditions suivantes prennent en charge la consommation de messages asynchrones. Une application peut enregistrer une fonction de rappel pour une destination. Lorsqu'un message approprié est envoyé à la destination, WebSphere MQ appelle la fonction et transmet le message en tant que paramètre. La fonction traite ensuite le message de manière asynchrone. Dans les éditions précédentes de WebSphere MQ, cette fonction était disponible uniquement lors de l'utilisation de WebSphere MQ classes for JMS.

WebSphere MQ classes for JMS a été modifié pour exploiter cette nouvelle fonction dans WebSphere MQ V7.0 et toute édition ultérieure. L'implémentation des programmes d'écoute de messages JMS est désormais plus naturelle avec WebSphere MQet WebSphere MQ classes for JMS n'a plus à interroger une destination pour vérifier si un message approprié a été envoyé à la destination. Les performances des programmes d'écoute de messages JMS s'en trouvent améliorées, en particulier lorsqu'une application utilise plusieurs programmes d'écoute de messages dans une session pour surveiller plusieurs destinations. Le débit des messages est augmenté et le temps nécessaire à la distribution d'un message à un programme d'écoute de message après son arrivée à une destination est réduit.

Les beans gérés par message (MDB) comportent des améliorations de performances similaires. En outre, en raison d'une autre amélioration de la fonction WebSphere MQ , plusieurs beans gérés par message qui consomment des messages provenant de la même destination subissent désormais un conflit réduit sur les messages.

Sélection de messages

A l'exception de la sélection des messages par identificateur de message ou par identificateur de corrélation, toutes les sélections de message effectuées dans les versions de WebSphere MQ antérieures à la version 7.0 ont été effectuées par WebSphere MQ classes for JMS. Dans WebSphere MQ V7.0et toute édition ultérieure, la sélection de tous les messages est effectuée par le gestionnaire de files d'attente.

Par conséquent, le débit de message est amélioré pour les applications consommant des messages à l'aide de la sélection de message. L'amélioration des performances est plus importante pour une application qui se connecte en mode client car seuls les messages qui répondent aux critères de sélection sont transportés sur le réseau et les classes WebSphere MQ pour JMS ne traitent que les messages qu'elle distribue à l'application.

Partage d'une connexion de communications

Dans les éditions précédentes de WebSphere MQ, si une application client WebSphere MQ était connectée à un gestionnaire de files d'attente plusieurs fois à l'aide du même canal MQI, chaque instance du canal MQI nécessitait une connexion TCP distincte. Dans WebSphere MQ V7.0 et toute édition ultérieure, chaque connexion au gestionnaire de files d'attente à l'aide du même canal MQI peut partager une connexion TCP unique. Cette disposition signifie que moins de ressources réseau sont requises et que le temps total nécessaire à la création de plusieurs connexions au gestionnaire de files d'attente est réduit, en particulier lors de l'utilisation de SSL, car l'établissement de liaison SSL n'a lieu qu'une seule fois au début de la connexion TCP.

WebSphere MQ classes for JMS exploite cette amélioration. Pour une application qui se connecte à un gestionnaire de files d'attente en mode client, WebSphere MQ classes for JMS peut créer plusieurs connexions à un gestionnaire de files d'attente à l'aide du canal MQI avec le nom spécifié comme propriété de l'objet ConnectionFactory . Chacune de ces connexions au gestionnaire de files d'attente peut désormais partager une connexion TCP unique.

Lecture anticipée sur des connexions client

Si une application utilise une connexion client pour consommer des messages non persistants à partir d'une destination, la destination peut être configurée de sorte que WebSphere MQ classes for JMS utilise une mémoire tampon pour stocker les messages qui vous intéressent avant de les distribuer à l'application. Cette optimisation est appelée *lecture anticipée* et peut être utilisée par les applications qui

consomment les messages de façon asynchrone en appelant la méthode receive() et par les programmes d'écoute de message et les MDB qui consomment les messages de façon asynchrone. La lecture anticipée est particulièrement efficace pour les destinations avec un grand nombre de messages devant être consommés rapidement.

La lecture anticipée ne s'applique pas aux messages persistants car, si ces derniers sont lus dans une mémoire tampon, le gestionnaire de files d'attente ne peut plus récupérer les messages à la suite d'un incident. Cependant, une application consommant des messages à partir d'une destination avec une combinaison de messages persistants et non persistants peut toujours la lecture anticipée. L'ordre des messages est préservé mais les avantages en termes d'exécution de la lecture anticipée ne s'appliquent qu'aux messages non persistants.

Lorsque vous décidez d'utiliser la lecture anticipée, tenez compte des points suivants:

- Si une application consomme des messages à partir d'une destination configurée pour la lecture anticipée et que l'application prend fin pour une raison quelle qu'elle soit, tous les messages non persistants stockés dans la mémoire tampon sont supprimés.
- Si toutes les conditions suivantes sont satisfaites, les messages envoyés à une file d'attente dans une session risquent de ne pas être reçus dans l'ordre dans lequel ils ont été envoyés :
  - Une application utilise deux consommateurs de message dans la même session pour consommer les messages à partir de la file d'attente.
  - Chaque consommateur de message utilise un objet Destination différent pour la file d'attente.
  - L'un des objets Destination ou les deux sont configurés pour la lecture anticipée.

Envoi de messages

Lorsqu'une application envoie des messages à une destination, la destination peut être configurée de sorte que, lorsque l'application appelle send (), WebSphere MQ classes for JMS transfère le message au gestionnaire de files d'attente et renvoie le contrôle à l'application sans déterminer si le gestionnaire de files d'attente a reçu le message en toute sécurité. WebSphere MQ classes for JMS peut fonctionner de cette manière uniquement pour les messages non persistants et pour les messages persistants envoyés dans une session transactionnelle.

Pour les messages persistants envoyés dans une session transactionnelle, l'application détermine en fin de compte si le gestionnaire de files d'attente a reçu les messages en toute sécurité lorsqu'il appelle commit (). Pour les messages 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 WebSphere MQ classes for JMS ne vérifie que le gestionnaire de files d'attente a reçu les messages en toute sécurité.

Cette optimisation profite particulièrement à une application qui se connecte à un gestionnaire de files d'attente en mode client et qui doit envoyer une séquence de messages à un intervalle très rapproché mais qui ne demande pas un retour immédiat d'informations de la part du gestionnaire de files d'attente pour chaque message envoyé.

Exits de canal

Lorsqu'ils sont appelés à partir de WebSphere MQ classes for JMS, les programmes d'exit de canal écrits en C ou C++ se comportent désormais de la même manière que lorsqu'ils sont appelés à partir d'un client WebSphere MQ MQI. Les performances des classes d'exit de canal écrites en Java ont été améliorées et vous pouvez désormais écrire des classes d'exit de canal à l'aide d'un nouvel ensemble d'interfaces dans le package com.ibm.mq.exits au lieu d'utiliser les interfaces dans WebSphere MQ classes for Java.

Propriétés des messages

Un message JMS se compose d'un ensemble de zones d'en-tête, d'un ensemble de propriétés et d'un corps contenant les données d'application. Au minimum, un message WebSphere MQ se compose d'un descripteur de message et des données d'application.

Lorsqu'une application WebSphere MQ classes for JMS envoie un message JMS, WebSphere MQ classes for JMS mappe le message JMS dans un message WebSphere MQ . Certaines des zones et propriétés d'en-tête JMS sont mappées dans des zones du descripteur de message, et d'autres sont mappées dans des zones d'un en-tête WebSphere MQ supplémentaire appelé en-tête MQRFH2 . Lorsqu'une application WebSphere MQ classes for JMS reçoit un message JMS, WebSphere MQ classes for JMS effectue le mappage inverse.

Une application qui utilise l'interface MQI pour recevoir des messages d'une application WebSphere MQ for JMS doit donc pouvoir gérer un en-tête MQRFH2 . Si l'application ne peut pas traiter un en-tête MQRFH2 , la propriété TARGCLIENT de l'objet Destination peut être définie pour indiquer à WebSphere MQ classes for JMS de ne pas inclure d'en-tête MQRFH2 dans les messages WebSphere MQ . Toutefois, en excluant l'en-tête MQRFH2 , les informations contenues dans certaines des zones et propriétés d'en-tête JMS sont perdues.

De même, une application qui utilise l'interface MQI pour envoyer des messages à une application WebSphere MQ pour JMS doit inclure un en-tête MQRFH2 dans chaque message. Si un en-tête MQRFH2 n'est pas inclus, WebSphere MQ classes for JMS peut définir uniquement les zones d'en-tête JMS et les propriétés qui peuvent être dérivées des zones d'un descripteur de message.

WebSphere MQ V7.0 fournit une prise en charge supplémentaire pour les applications qui utilisent l'interface MQI pour recevoir et envoyer des messages à WebSphere MQ classes for JMS applications.

Lorsqu'une application appelle MQGET pour recevoir un message d'une application WebSphere MQ classes for JMS, l'application peut choisir de recevoir le message de l'une des manières suivantes:

1. Le message est distribué avec un descripteur de message, un en-tête MQRFH2 qui contient les données dérivées des zones d'en-tête et des propriétés JMS, ainsi que les données d'application.
2. Le message est distribué à l'aide d'un descripteur de message, des données d'application et d'un ensemble de propriétés de message.

Dans l'option 2, chaque propriété de message représente une zone d'en-tête JMS ou une propriété qui a été mappée à l'origine par WebSphere MQ classes for JMS dans une zone dans un en-tête MQRFH2 . A la suite de l'appel MQGET, l'application peut utiliser l'appel MQINQMP pour obtenir les valeurs des propriétés de message. L'utilisation de l'option 2 à la place de l'option 1 pour la réception d'un message simplifie la logique d'application des manières suivantes :

- L'application n'a pas besoin d'analyser la partie variable de l'en-tête MQRFH2 , qui contient la zone d'en-tête JMS et les données de propriété codées dans un format de type XML.
- L'application n'a pas besoin de convertir les données de type caractère dans la partie variable de l'en-tête MQRFH2.

En conséquence, avant qu'une application appelle MQPUT pour envoyer un message à une application WebSphere MQ pour JMS, l'application peut utiliser l'appel MQSETMP pour définir les valeurs des propriétés de message au lieu de construire un en-tête MQRFH2 .

Maintenabilité

WebSphere MQ classes for JMS contient un certain nombre d'améliorations liées à la maintenabilité:

- Traçage.

  WebSphere MQ classes for JMS contient une classe qu'une application peut utiliser pour contrôler le traçage. Une application peut démarrer et arrêter le traçage, spécifier le niveau de détail requis dans une trace et personnaliser la sortie de trace de différentes manières.

- Consignation.

  WebSphere MQ classes for JMS gère un fichier journal qui contient des messages sur les erreurs que vous devez corriger. Les messages sont écrits sous forme de texte normal. WebSphere MQ classes for JMS contient une classe qu'une application peut utiliser pour spécifier l'emplacement du fichier journal et sa taille maximale.

- First Failure Support Technology (FFST).

En cas d'échec grave, WebSphere MQ classes for JMS génère un rapport FFST dans un fichier FDC. Le rapport FFST contient des informations que le service IBM peut utiliser pour diagnostiquer le problème plus rapidement.

- Informations sur la version.

  WebSphere MQ classes for JMS contient une classe qu'une application peut utiliser pour interroger la version de WebSphere MQ classes for JMS.

- Messages d'exception.

  Les messages d'exception ont été améliorés pour fournir plus d'informations sur les causes des erreurs et les actions requises pour corriger les erreurs.

- Serveurs d'applications.

  L'intégration des fonctions de serviçabilité de WebSphere MQ classes for JMS avec celles de WebSphere Application Server a été améliorée.

MQC est remplacé par MQConstants

Un nouveau package, com.ibm.mq.constants, est fourni avec IBM WebSphere MQ Version 7.0. Ce module comprend la classe MQConstants qui applique un certain nombre d'interfaces. MQConstants comprend les définitions de toutes les constantes se trouvant dans l'interface MQC ainsi qu'un certain nombre de nouvelles constantes. Les interfaces de ce package suivent de près les noms des fichiers d'en-tête de constantes utilisés dans IBM WebSphere MQ.

Par exemple, l'interface CMQC comprend une constante MQOO_INPUT_SHARED ; cette interface et cette constante correspondent au fichier d'en-tête `cmqc.h` et à la constante MQOO_INPUT_SHARED.

com.ibm.mq.constants peut être utilisé avec les classes IBM WebSphere MQ pour Java et IBM WebSphere MQ pour JMS.

MQC est toujours présent et possède les constantes qu'il avait précédemment. En revanche, pour de nouvelles applications, vous devez utiliser le package com.ibm.mq.constants.

Ecriture des classes WebSphere MQ pour les applications JMS

Après une brève introduction au modèle JMS, cette rubrique fournit des conseils détaillés sur la façon d'écrire des classes WebSphere MQ pour les applications JMS.

Le modèle JMS

Le modèle JMS définit un ensemble d'interfaces que les applications Java peuvent utiliser pour effectuer des opérations de messagerie. WebSphere MQ classes for JMS, en tant que fournisseur JMS, définit comment les objets JMS sont liés aux concepts WebSphere MQ . La spécification JMS s'attend à ce que certains objets JMS soient des objets gérés.

La spécification JMS et le package javax.jms définissent un ensemble d'interfaces que les applications Java peuvent utiliser pour effectuer des opérations de messagerie. La liste suivante récapitule les principales interfaces JMS:

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 127, à la page 834 présente ces objets et leurs relations.

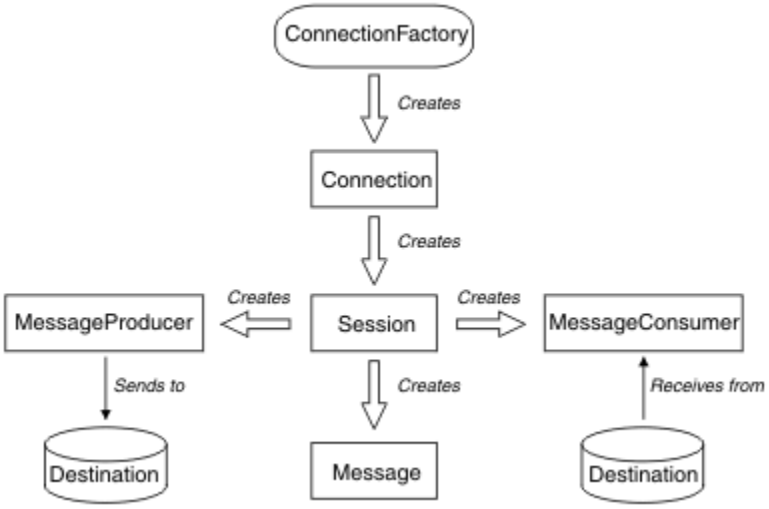


Figure 127. Objets JMS et leurs relations

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.

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. Avec 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 103, à la page 835 répertorie les interfaces indépendantes du domaine JMS et les interfaces spécifiques au domaine correspondantes.

Tableau 103. Interfaces indépendantes du domaine JMS et spécifiques au domaine		
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 1.1 conserve toutes les interfaces spécifiques au domaine, de sorte que les applications existantes peuvent toujours utiliser ces interfaces. Pour les nouvelles applications, cependant, pensez à utiliser les interfaces indépendantes du domaine.

Dans WebSphere MQ classes for JMS, les objets JMS sont liés aux concepts WebSphere MQ des manières suivantes:

- 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 un descripteur de connexion WebSphere MQ , qui définit donc la portée transactionnelle de la session.
- Un objet MessageProducer et un objet MessageConsumer encapsulent chacun un descripteur d'objet WebSphere MQ .

Lorsque vous utilisez WebSphere MQ classes for JMS, toutes les règles normales de WebSphere 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 des 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 WebSphere MQ classes for JMS, l'implémentation de l'interface Destination est une superclasse abstraite de Queue et Topic. Par conséquent, 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.

Dans les classes WebSphere MQ pour JMS, les objets des types suivants peuvent être des objets gérés:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- File d'attente
- Topic
- XAConnectionFactory
- XAQueueConnectionFactory

- XATopicConnectionFactory

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 plusieurs types de corps de message qui couvrent la majorité des styles de messagerie actuellement utilisés.

JMS définit cinq types de corps de message:

Flux
Flux de valeurs primitives Java. Il est rempli et lu séquentiellement.

Mapper
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 en son nom, 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.

Un sélecteur 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 compris dans la plage des nombres longs Java 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 dans la plage des doubles Java sont pris en charge.
  - Les littéraux booléens TRUE et FALSE.
- Identificateurs :
  - Un identificateur est une séquence de longueur illimitée de lettres et de chiffres Java, le premier devant 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.
- L'espace est identique à celui défini 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.

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"
```

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.

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 vers des messages WebSphere MQ

Les messages WebSphere 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 WebSphere MQ .

Cette rubrique décrit comment la structure de message JMS décrite dans la première partie de cette section est mappée à un message WebSphere MQ . Il est intéressant pour les programmeurs qui souhaitent transmettre des messages entre JMS et les applications WebSphere MQ traditionnelles. Elle est également intéressante pour les personnes qui souhaitent manipuler des messages transmis entre deux applications JMS, par exemple, dans une implémentation WebSphere Message Broker.

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 aucun message WebSphere MQ n'est impliqué.

Les messages WebSphere MQ sont composés de trois composants:

- Le descripteur de message WebSphere MQ (MQMD)

- En-tête WebSphere MQ MQRFH2
- Corps du message.

MQRFH2 est facultatif et son inclusion dans un message sortant est régie par un indicateur dans la classe de destination JMS. Vous pouvez définir cet indicateur à l'aide de l'outil d'administration JMS WebSphere MQ . 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 non JMS. En effet, une telle application n'attend pas de MQRFH2 dans son message WebSphere 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 128, à la page 840 montre comment la structure d'un message JMS est transformée en message WebSphere MQ et en retour:

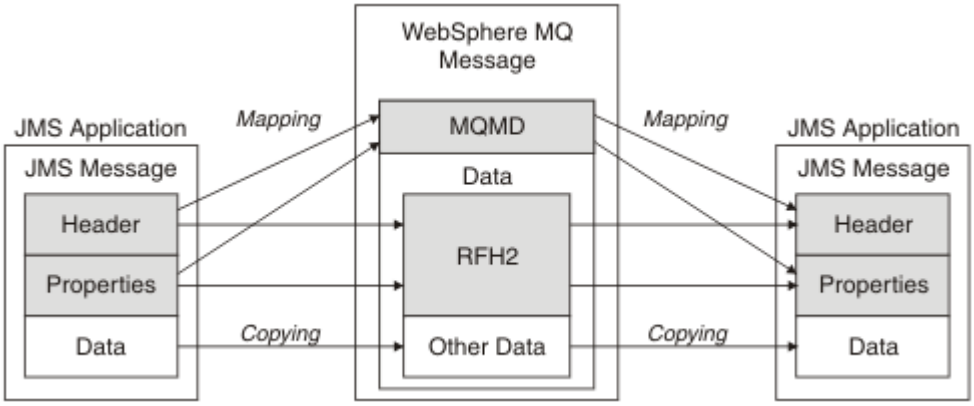


Figure 128. Comment les messages sont transformés entre JMS et WebSphere MQ à l'aide de l'en-tête MQRFH2

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 non JMS.

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.

En-tête MQRFH2 et JMS
Cette collection de rubriques décrit l'en-tête MQRFH version 2, qui contient les données spécifiques à JMS associées au contenu du message. MQRFH2 version 2 est un en-tête extensible qui 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* WebSphere 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 104. Valeurs possibles pour la zone NameValueCCSID

Valeur	Explication
1 200	UCS2 ouvert
1208	UTF8
13488	UCS2 2.0 Sous-ensemble
17584	UCS2 2.1 Sous-ensemble (inclut le symbole Euro)

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:

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 message MQRFH2 envoyé par WebSphere Message Broker. Il décrit le domaine, le format, le type et l'ensemble de messages d'un message.

Tableau 105. 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.

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 existe aussi dans un dossier JMS MQRFH2.

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.

Dossier < mqext>

mqext contient des propriétés utilisées uniquement par WebSphere Application Server. Le dossier est présent uniquement si l'application a défini au moins l'une des propriétés IBM définies.

Tableau 106. 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>

N'ajoutez pas vos propres propriétés dans le dossier mqext.

Dossier < mqps>

mqps contient des propriétés qui sont utilisées uniquement par la publication/l'abonnement IBM WebSphere MQ. Le dossier est présent uniquement si l'application a défini au moins l'une des propriétés de publication/abonnement intégré.

Tableau 107. 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 108, à la page 843 affiche la liste complète des noms de propriété.

Tableau 108. Dossiers et propriétés MQRFH2 utilisés par JMS				
Nom de zone JMS	**Type Java**	**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	quelconque

Tableau 108. Dossiers et propriétés MQRFH2 utilisés par JMS (suite)				
Nom de zone JMS	**Type Java**	**Nom de dossier MQRFH2**	**Nom de la propriété**	**Type / valeurs**
		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 108, à la page 843.

datatype
doit correspondre, après pliage, à l'un des types de données répertoriés dans le Tableau 109, à la page 844.

value
est une représentation de chaîne de la valeur à transmettre, à l'aide des définitions dans Tableau 109, à la page 844.

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 109. 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

Tableau 109. Types de données de propriété (suite)	
Type de données	**Définition**
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 110, à la page 845 répertorie les zones d'en-tête JMS et Tableau 111, à la page 846 répertorie les propriétés JMS qui sont mappées directement aux zones MQMD. Tableau 112, à la page 846 répertorie les propriétés spécifiques au fournisseur et les zones MQMD auxquelles elles sont mappées.

Tableau 110. Mappage des zones d'en-tête JMS aux zones MQMD			
Zone d'en-tête JMS	**Type Java**	**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

Tableau 110. Mappage des zones d'en-tête JMS aux zones MQMD (suite)			
Zone d'en-tête JMS	**Type Java**	**Zone MQMD**	**Type sté**
JMSCorrelationID	String	CorrelId	MQBYTE24

Tableau 111. Mappage des propriétés JMS aux zones MQMD			
propriété JMS	**Type Java**	**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 112. Mappage des propriétés spécifiques au fournisseur JMS vers les zones MQMD			
Propriété spécifique au fournisseur JMS	**Type Java**	**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
JMS_IBM_Format	String	Format«1», à la page 847	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Codage	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId«2», à la page 847	MQLONG
JMS_IBM_PutDate	String	PutDate	MQCHAR8
JMS_IBM_PutTime	String	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

Tableau 112. Mappage des propriétés spécifiques au fournisseur JMS vers les zones MQMD (suite)			
Propriété spécifique au fournisseur JMS	**Type Java**	**Zone MQMD**	**Type sté**
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 WebSphere 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 sur des zones WebSphere 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 ().

La Tableau 113, à la page 847 montre comment les zones d'en-tête JMS sont mappées dans des zones MQMD/RFH2 au moment de l'envoi () ou de la publication (). Tableau 114, à la page 848 montre comment les propriétés JMS sont mappées dans les zones MQMD/RFH2 au moment de l'envoi () ou de la publication (). Tableau 115, à la page 848 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 correspond à 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'exécution de la fonction send () ou publish () (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 113. 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
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

Tableau 113. Mappage de zones de message sortant (suite)

Nom de zone d'en-tête JMS	Zone MQMD utilisée pour la transmission	En-tête	Définie par
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 114. Mappage de propriétés JMS de message sortant

Nom de 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

Tableau 115. Mappage de propriété spécifique au fournisseur JMS du 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 lors de l'envoi () ou de la publication ()
Ces remarques concernent le mappage des zones JMS à send () ou publish ().

JMSDestination vers MQRFH2
Il est stocké sous la forme d'une chaîne qui sérialise les caractéristiques principales de l'objet de destination, de sorte qu'un JMS récepteur 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 909).

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 909 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 WebSphere MQ. La valeur affectée est renvoyée dans MQMD.MessageId après l'appel MQPUT et est retransmise à l'application dans la zone JMSMessageID . WebSphere MQ messageId est une valeur binaire de 24 octets, alors 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 JMS WebSphere MQ décrites dans «Lecture et écriture du descripteur de message à partir d'une application WebSphere MQ classes for JMS», à la page 926.

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 . Voir aussi «Utilisation d'une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker», à la page 954.

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 *ID:* Le préfixe est supprimé, les caractères restants sont convertis en caractères binaires, puis ils sont définis dans MQMD.CorrelId . Aucune valeur CorrelId n'est codée dans MQRFH2.

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 WebSphere 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 WebSphere MQ .

JMS_IBM_Report_ < nom > vers le rapport 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. L'application doit définir la valeur de ces propriétés sur les constantes IBM WebSphere MQ MQRO_ standard (incluses dans com.ibm.mq.MQC). Par exemple, pour demander

un DCO avec des données complètes, l'application doit définir JMS_IBM_Report_COD sur la valeur CMQC.MQRO_COD_WITH_FULL_DATA.

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

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 WebSphere MQ , MSGType est défini sur la valeur MQMT_REQUEST
- Si JMSReplyTo n'est pas défini ou s'il est défini sur une autre destination de file d'attente IBM WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ existante que ce message est le dernier d'un groupe. Cette propriété est ignorée pour la messagerie de publication / abonnement.

Mappage de zones WebSphere MQ sur 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 au moment de l'obtention () ou de la réception ().

La Tableau 116, à la page 852 montre comment les zones d'en-tête JMS sont mappées aux zones MQMD/MQRFH2 au moment de l'obtention () ou de la réception (). Tableau 117, à la page 853 montre comment les zones de propriété JMS sont mappées aux zones MQMD/MQRFH2 au moment de l'obtention () ou de la réception (). Tableau 118, à la page 853 montre comment les propriétés spécifiques au fournisseur JMS sont mappées.

Tableau 116. 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 853
JMSDeliveryMode	Persistance«2», à la page 853	jms.Dlv«2», à la page 853
JMSExpiration		jms.Exp
JMSPriority	Priorité	
JMSMessageID	MsgID	
JMSTimestamp	PutDate«2», à la page 853 PutTime«2», à la page 853	jms.Tms«2», à la page 853
JMSCorrelationID	CorrelId«2», à la page 853	jms.Cid«2», à la page 853
JMSReplyTo	ReplyToQ«2», à la page 853 ReplyToGestionnaire de files d'attente«2», à la page 853	jms.Rto«2», à la page 853
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Tableau 116. Mappage de zones d'en-tête JMS de message entrant (suite)

Nom de zone d'en-tête JMS	Zone MQMD extraite de	Zone MQRFH2 extraite de

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 117. Mappage des propriétés de message entrant

Nom de propriété JMS	Zone MQMD extraite de	Zone MQRFH2 extraite de
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId«1», à la page 853	jms.Gid«1», à la page 853
JMSXGroupSeq	MsgSeqNuméro«1», à la page 853	jms.Seq«1», à la page 853

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 118. Mappage des propriétés JMS spécifiques au fournisseur de messages entrants

Nom de 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	

Tableau 118. Mappage des propriétés JMS spécifiques au fournisseur de messages entrants (suite)		
Nom de propriété JMS	**Zone MQMD extraite de**	**Zone MQRFH2 extraite de**
JMS_IBM_Encoding «1», à la page 854	Codage	
JMS_IBM_Character_Set «1», à la page 854	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 WebSphere MQ traditionnelle
Cette rubrique décrit ce qui se passe lorsqu'une application JMS échange des messages avec une application WebSphere MQ traditionnelle qui ne peut pas traiter l'en-tête MQRFH2 .

. Figure 129, à la page 854 montre le mappage.

L'administrateur indique que l'application JMS communique avec une application WebSphere 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 WebSphere MQ traditionnelle est identique au mappage de JMS vers MQMD destiné à une application JMS. Si WebSphere MQ classes for JMS reçoit un message WebSphere MQ avec la zone MQMD *Format* définie sur une valeur autre que MQFMT_RFH2, les données sont reçues d'une application non JMS. Si le format est MQFMT_STRING, le message est reçu en tant que message texte JMS. Sinon, il est reçu en tant que 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 WebSphere MQ classes for JMS reçoit un message qui ne possède 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.

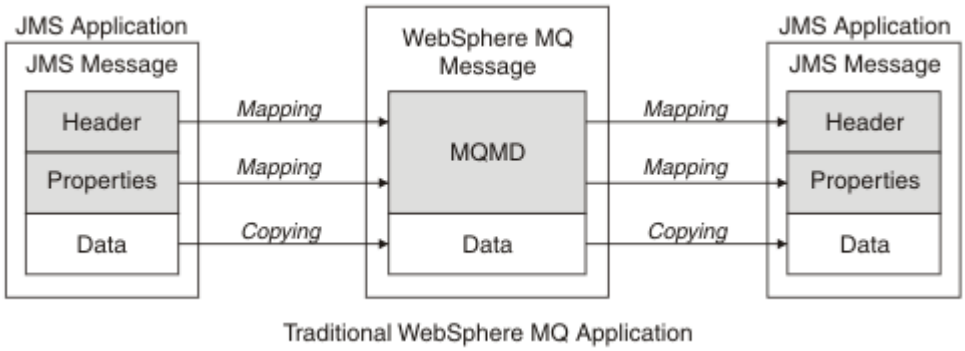


Figure 129. Comment les messages JMS sont transformés en messages WebSphere MQ sans en-tête MQRFH2

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 WebSphere MQ classes for JMS. Ces jeux de caractères sont utilisés principalement lorsque vous envoyez un message à une application non JMS.

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 WebSphere 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 dans S/390 au lieu du 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 WebSphere MQ (ou MQMD s'il n'y a 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 WebSphere MQ entrant n'a aucun effet sur l'interprétation de ce message en tant que BytesMessageJMS.

Il est peu probable que les applications non Java 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 109, à la page 844. 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 WebSphere MQ classes for JMS antérieures à la version 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>
```

La version 5.3 et les versions ultérieures de WebSphere MQ classes for JMS peuvent interpréter l'un ou l'autre format, mais les versions de WebSphere MQ classes for JMS antérieures à la version 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 WebSphere MQ classes for JMS antérieure à la version 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` est l'un des types de données répertoriés dans le Tableau 109, à la page 844. 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.

Conversion de messages JMS

La conversion des données de message dans JMS est effectuée lors de l'envoi et de la réception de messages. WebSphere MQ effectue la plupart des conversions de données automatiquement. Il convertit du texte et des données numériques lors du transfert d'un message entre des applications JMS. Le texte est converti lors de l'échange d'un `JMSTextMessage` entre une application JMS et une application WebSphere 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 textuels entre une application WebSphere MQ et une application JMS.
- Echange de données texte au format octet.
- Conversion du texte dans votre application.

Données de message JMS

La conversion de données est nécessaire pour échanger du texte et des données numériques entre les 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. WebSphere MQ gère le codage du texte et des nombres dans les messages JMS, à l'exception de `JMSObjectMessage`, voir «JMSObjectMessage», à la page 863. Il utilise trois attributs de message. Les trois attributs sont `CodedCharacterSetId`, `Encoding`et `Format`.

Ces trois attributs de message sont normalement stockés dans les zones d'en-tête JMS, `MQRFH2`, 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 WebSphere MQ .

Propriétés de message JMS

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 WebSphere 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.

Conversion de messages JMS

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 n'échange que du texte ou des messages uniquement avec d'autres applications JMS, vous n'envisagez normalement pas la conversion de données. La conversion des données est effectuée automatiquement pour vous par WebSphere 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 WebSphere MQ , IBM WebSphere 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 857
2. «Conversion des données d'application», à la page 858
3. «Conversion des données du gestionnaire de files d'attente», à la page 859
4. «Conversion de données de canal de transmission de messages», à la page 860

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 860, 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

Client JMS[4]La conversion de données est la conversion de primitives et d'objets Java en octets dans un message JMS lorsqu'il est envoyé à une destination, et la conversion à nouveau, lorsqu'il est reçu. La conversion des données client JMS utilise les méthodes des classes JMSMessage . Les méthodes sont répertoriées par type de classe JMSMessage dans Tableau 119, à la page 861.

[4] "Client JMS" désigne les classes WebSphere MQ for JMS qui implémentent l'interface JMS, qui s'exécute en mode client ou en mode liaisons.

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 s'appliquer.

Pour obtenir les données d'un message, une application appelle les méthodes de lecture ou d'extraction 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 du client JMS, voir «Conversion et codage des messages du client JMS», à la page 870.

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 132, à la page 863 et Figure 133, à la page 863 . 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.[5]. 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 des formats de codage différents. 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 non JMS. 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 non JMS n'ait spécifié le format de message "MQSTR ", le message est

[5] Une exception: les données écrites à l'aide de `writeUTF` commencent par une zone de longueur de 2 octets

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 non JMS», à la page 878.

Conversion des données du gestionnaire de files d'attente

Dans WebSphere MQ V7.0, 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 131, à la page 862. 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 130, à la page 859.

Avant V7.0, les conversions étaient toujours effectuées par le client JMS. La conversion des données du client JMS est limitée à la conversion des séquences de nombres et de texte de type et de longueur connus du client JMS. Il ne peut pas convertir les structures de données ; voir «Echange d'un enregistrement formaté avec une application non JMS», à la page 878. Dans V7.0, jusqu'au groupe de correctifs 7.0.1.5, si la conversion peut être effectuée par le gestionnaire de files d'attente, elle est toujours effectuée par le gestionnaire de files d'attente. A partir de 7.0.1.5 , le comportement de conversion par défaut est identique à celui de V6.0et toutes les conversions sont effectuées par le client JMS. A partir de 7.0.1.5ou de 7.0.1.4 avec l'APAR IC72897, vous pouvez définir une nouvelle option de destination, WMQ_RECEIVE_CONVERSION, pour contrôler où la conversion est effectuée et WMQ_RECEIVE_CCSID, pour définir la page de codes cible ; voir Figure 130, à la page 859.

```
((MQDestination)destination).setIntProperty(
              WMQConstants.WMQ_RECEIVE_CONVERSION,
              WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Ou,

```
((MQDestination)destination).setReceiveConversion
             (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
```

Figure 130. Activer la conversion des données du gestionnaire de files d'attente

Le principal avantage de la conversion de gestionnaire de files d'attente réside dans l'échange de messages avec des applications non JMS. 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 WebSphere MQ prédéfinis, tels qu'un en-tête de pont CICS (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 non JMS 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 d'envoi et un récepteur non JMS, l'exemple s'applique à IBM WebSphere MQ pre- et post-V7.0. Avec IBM WebSphere MQ V7.0, l'exit de conversion peut également être appelé pour un programme JMS de réception.

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 non JMS», à la page 878.

Conversion de données de canal de transmission de messages

WebSphere MQ Sender, Server, Cluster-receiver et Cluster-sender 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 WebSphere MQ pour la conversion de code est la suivante: "le récepteur est efficace". "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, s'il ne s'agit pas d'un client JMS, peut supprimer la nécessité pour le destinataire non JMS 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 des propriétés de destination ou de message, vous pouvez définir le jeu de caractères et le codage utilisés par WebSphere MQ pour coder les numéros et le texte dans les messages. Normalement, vous devez conserver le jeu de caractères `1208` et le codage `Native`.

WebSphere 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. Le `Charset` par défaut est généralement `windows-1252` sous Windowset `UTF-8` sous UNIX. `windows-1252` est un jeu de caractères à un octet et `UTF-8` est un jeu de caractères à plusieurs octets.

En règle générale, conservez les valeurs par défaut du jeu de caractères de destination et des propriétés de codage `UTF-8` et `Native` 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 non JMS 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 les classes IBM WebSphere MQ pour JMS n'ajoutent 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 non JMS», à la page 878. pour des exemples de conversion de messages échangés avec des applications non JMS.

Exemples

Tableau des types de message et des types de conversion

Tableau 119. 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`

Tableau 119. 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

Appel de la conversion de données à partir d'un programme C

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                 */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Figure 131. Fragment de code de amqsget0.c

Envoi et réception de texte dans un JMSBytesMessage

Le code dans Figure 132, à la page 863 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 133, à la page 863. 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 132. 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 133. 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 non JMS recevant des messages des clients JMS. Depuis la V7.0, les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente. Depuis 7.0.1.5ou 7.0.1.4 avec l'APAR IC72897, la conversion des données du gestionnaire de files d'attente est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application non JMS
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 non JMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application non JMS 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-8`et 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 non JMS.

La «Envoi et réception d'un JMSObjectMessage», à la page 866 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 WebSphere MQ. Les zones `CodedCharacterSetId` et `Format` sont définies dans le descripteur de message, `MQMD`, ou dans les zones JMS d'une `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. Sinon, le message possède un RFH2 JMS 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 866.

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 à partir 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_CCSID`différent. 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 866.

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 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 869. Les détails sont décrits dans «Conversion et codage des messages du client JMS», à la page 870. Chaque primitive est balisée ; voir «Corps du message JMS», à la page 854.

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 138, à la page 867, 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 `JMSMapMessage`ou 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. Pour plus d'informations, voir Class JMSStreamMessage . L'exemple dans «Envoi de données dans JMSStreamMessage et JMSMapMessage», à la page 867 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 JMS et 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 869. Les détails sont décrits dans «Types de message JMS et conversion», à la page 863.

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 les classes WebSphere MQ for JMS effectuant une conversion ; voir «Envoi et réception de texte dans un JMSBytesMessage», à la page 867.

Un `JMSBytesMessage` envoyé à une application MQ est généralement envoyé 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 868, 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 134. 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 135. 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 136. 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 137. 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 138. Envoi de données dans JMSStreamMessage et JMSMapMessage

Envoi et réception de texte dans un JMSBytesMessage

Le code dans Figure 139, à la page 867 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 140, à la page 868. 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 139. 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 140. 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 141, à la page 868 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 141. 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 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 142, à la page 868 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 142. 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 120. 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
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 n'échange que du texte ou des messages uniquement avec d'autres applications JMS, vous n'envisagez normalement pas la conversion de données. La conversion des données est effectuée automatiquement pour vous par WebSphere 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 non JMS recevant des messages des clients JMS. Depuis la V7.0, les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente. Depuis 7.0.1.5ou 7.0.1.4 avec l'APAR IC72897, la conversion des données du gestionnaire de files d'attente est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application non JMS
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 non JMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application non JMS 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 des primitives ou des objets Java sont lus ou écrits dans et depuis 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 des données sont lues ou écrites dans un message JMS. Le texte est converti vers et depuis la représentation Unicode 16 bits interne[6]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 dépendent du type de message JMS et de l'opération de lecture ou d'écriture.

Tableau 121, à la page 870 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 121. Types de message et types de conversion

	Type de conversion			
Type de message	**Texte**	**Numérique**	**Autre**	**Aucun**
`JMSObjectMessage`				`getObject` `setObject`
`JMSTextMessage`	`getText` `setText`			

[6] Certaines représentations Unicode nécessitent plus de 16 bits. Voir une référence Java SE.

Tableau 121. 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

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[7].

Les fragments de code de «Envoi et réception d'un JMSTextmessage», à la page 874 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.

[7] Lors de la réception d'un message, JMS_IBM_CHARACTER_SET est un nom de page de codes Java Charset .

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 147, à la page 874 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 WebSphere MQ Explorer. Parcourez la file d'attente et affichez les propriétés du message avant de l'extraire.

Comparez le fragment de code dans Figure 146, à la page 874 avec le fragment de code incorrect dans Figure 143, à la page 872. Dans le fragment incorrect, la chaîne de texte est convertie deux fois, une fois par l'application, puis à nouveau par WebSphere MQ.

```
TextMessage tmo = session.createTextMessage();
tmo.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, 37);
tmo.setText(new String("Sent in EBCDIC character set 37".getBytes(CCSID.getCodepage(37))));
producer.send(tmo);
```

Figure 143. 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. Lors de 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 148, à la page 874 montre une application codant un nombre au format de destination et l'envoyant dans un `JMSStreamMessage`. Comparez l'exemple dans Figure 148, à la page 874 à l'exemple dans Figure 149, à la page 875. 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 WebSphere 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, JMSStreamMessageet 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, writeByteet 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 875.

WebSphere MQ n'effectue aucune conversion de page de codes dans le client car le message est un JMSBytesMessageet 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. Les classes WebSphere MQ pour Java utilisent JMS_IBM_CHARACTER_SET pour définir la zone CodedCharacterSetId dans les propriétés JMS dans MQRFH2ou dans le descripteur de message, MQMD:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);[8]
```

Si un tableau d'octets est écrit dans un JMSStringMessage ou JMSMapMessage, WebSphere MQ classes for JMS n'effectue pas de conversion de données, car les octets sont saisis sous forme de données hexadécimales et non sous forme de 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 144, à la page 874 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 possède généralement une valeur par défaut Charset windows-1252et UNIX, UTF-8. L'échange entre Windows et UNIX nécessite que vous sélectionniez une page de codes explicite pour l'échange de texte sous forme d'octets.

[8] SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage) currently accepts only numeric character set identifiers.

```
StreamMessage smo = producer.session.createStreamMessage();
smo.writeBytes("123".getBytes(CCSID.getCodepage(((MQDestination) destination)
           .getIntProperty(WMQConstants.WMQ_CCSID))));
```

Figure 144. Ecriture d'octets représentant une chaîne dans un JMS`StreamMessage` à l'aide du jeu de caractères de destination

Exemples

Envoi et réception d'un JMS`Textmessage`

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 145. 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 146. 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 147. 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 148. Envoi d'un nombre à l'aide du codage de destination dans un JMS`StreamMessage`

```
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 149. 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 150, à la page 875 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 151, à la page 875. 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 150. 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 151. 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 n'échange que du texte ou des messages uniquement avec d'autres applications JMS, vous n'envisagez normalement pas la conversion de données. La conversion des données est effectuée automatiquement pour vous par WebSphere 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 non JMS recevant des messages des clients JMS. Depuis la V7.0, les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente. Depuis 7.0.1.5ou 7.0.1.4 avec l'APAR IC72897, la conversion des données du gestionnaire de files d'attente est facultative.

Tâches associées

Echange d'un enregistrement formaté avec une application non JMS
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 non JMS à l'aide de

JMSBytesMessage. L'échange d'un message formaté avec une application non JMS 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 non JMS recevant des messages des clients JMS. Depuis la V7.0, les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente. Depuis 7.0.1.5ou 7.0.1.4 avec l'APAR IC72897, la conversion des données du gestionnaire de files d'attente 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 non JMS, la fonction de conversion a toujours été disponible en définissant l'option GetMessage, GMO_CONVERT. La fonction de conversion du gestionnaire de files d'attente n'est pas disponible pour une application JMS recevant un message jusqu'à la V7.0.

Vous pouvez utiliser la conversion de gestionnaire de files d'attente, avant V7.0, avec une application client JMS qui envoie un message. Le client JMS génère un enregistrement formaté, définit les attributs CodedCharacterSetId, Encodinget Format correspondant aux données placées dans le message. Une application de réception non JMS lit le message à l'aide de GMO_CONVERTet appelle un exit de conversion de données écrit par l'utilisateur. L'exit de conversion de données est une bibliothèque partagée dont le nom est défini dans la zone Format .

Depuis la V7.0, le gestionnaire de files d'attente peut convertir les messages envoyés aux clients JMS. De 7.0.0.0 à 7.0.1.4 inclus, la conversion du gestionnaire de files d'attente est toujours appelée pour les clients JMS. A partir de 7.0.1.5ou de 7.0.1.4 avec application de l'APAR IC72897 , 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. WMQ_RECEIVE_CONVERSION_CLIENT_MSG est le paramètre par défaut, qui correspond au comportement de WebSphere MQ V6.0, qui ne prend pas en charge la conversion des données du gestionnaire de files d'attente pour les clients JMS. 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 152. 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, créant ainsi 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 153. 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 dans le code C de Figure 154, à la page 877,

```
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 154. 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 WebSphere 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 n'échange que du texte ou des messages uniquement avec d'autres applications JMS, vous n'envisagez normalement pas la conversion de données. La conversion des données est effectuée automatiquement pour vous par WebSphere 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 433
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 non JMS
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 non JMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application non JMS 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`.

Echange d'un enregistrement formaté avec une application non JMS
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 non JMS à l'aide de `JMSBytesMessage`. L'échange d'un message formaté avec une application non JMS 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 non JMS 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`, `JMSStreamMessageou` `JMSObjectMessage`, WebSphere MQ s'occupe des détails du formatage du message. WebSphere 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 non JMS. Pour ce faire, vous devez comprendre comment ces messages sont générés par les classes WebSphere MQ pour JMS. Il se peut que vous puissiez modifier l'application non JMS pour interpréter les messages ; voir «Mappage de messages JMS vers des messages WebSphere MQ», à la page 839.

Un avantage de l'utilisation de l'un de ces types de message est que 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 pouvant traiter 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 882comme point de départ pour la génération d'un client JMS pouvant échanger des enregistrements formatés avec des applications non JMS.

Procédure

1. Définissez la présentation de l'enregistrement ou utilisez l'une des classes d'en-tête WebSphere MQ prédéfinies.

   Pour la gestion des en-têtes WebSphere MQ prédéfinis, voir Gestion des en-têtes de message WebSphere MQ.

Figure 155, à la page 880 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 882, nommez l'exit de conversion de données MYRECORD.

3. Ecrivez des 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 882.
   - 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», à la page 860.

5. Configurez la destination pour échanger des 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 882 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 non JMS

   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 156, à la page 881. 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 882.
   - L'exemple de programme de navigation WebSphere MQ , `amqsbcg0.c`, est utile pour examiner le contenu de l'en-tête de message, l'en-tête JMS, MQRFH2et 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 155. 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 156. 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 n'échange que du texte ou des messages uniquement avec d'autres applications JMS, vous n'envisagez normalement pas la conversion de données. La conversion des données est effectuée automatiquement pour vous par WebSphere 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 non JMS recevant des messages des clients JMS. Depuis la V7.0, les clients JMS qui reçoivent des messages utilisent également la conversion des données du gestionnaire de files d'attente. Depuis 7.0.1.5ou 7.0.1.4 avec l'APAR IC72897, la conversion des données du gestionnaire de files d'attente est facultative.

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`.

Utilitaire permettant de créer un code de sortie de conversion

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 cette 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 non JMS. 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 WebSphere 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 sur 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 V7.0.
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 WebSphere MQ Explorer pour développer et exécuter les classes Java. Vous pouvez également utiliser votre propre environnement de développement Java.
4. L'utilisation de IBM WebSphere 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 WebSphere 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 WebSphere 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 889 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 JMS, la destination et la session 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 JMS Connection, Destination, Sessionet MessageProducer dans une seule classe. MyConsumer, utilisé ultérieurement, encapsule JMS Connection, Destination, Sessionet 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 . Elle n'est pas définie dans le message car elle n'est pas utilisée par les classes IBM WebSphere MQ pour 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 utiliser un client IBM WebSphere MQ MQI. 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 WebSphere MQ sur une valeur non spécifiée, s'il doit remplacer un paramètre de JMS.

Remarque :

Les classes IBM WebSphere MQ pour 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 WebSphere 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 les classes IBM WebSphere MQ pour JMS construisent 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 javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;
```

6. 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; }Â .
```

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 javax.jms.BytesMessage;
```

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

Résultats

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* à l'aide 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 156, à la page 881 dans «Echange d'un enregistrement formaté avec une application non JMS», à la page 878.
  - 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* à l'aide 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 WebSphere MQet exécutez le programme principal, `TryMyRecord`; voir Figure 157, à la page 889.

1. Configurez IBM WebSphere MQ et JMS pour exécuter l'exemple. Les instructions permettent d'exécuter l'exemple sous Windows.

   1. Création d'un gestionnaire de files d'attente

```
crtmqm -sa -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

   2. Création d'une file d'attente

```
echo DEFINE QL('Q1') REPLACE | runmqsc QM1
```

   3. Créez un répertoire JNDI

```
cd c:\
md JNDI-Directory
```

4. **Passez dans le répertoire bin JMS**
   Le programme d'administration JMS doit être exécuté à partir d'ici. Le chemin est *MQ_INSTALLATION_PATH*\java\bin.
5. **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
```

6. **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 WebSphere MQ .
3. Exécutez `TryMyRecord`.

Classes utilisées pour exécuter l'exemple

Les classes répertoriées dans les figures Figure 157, à la page 889 à Figure 162, à la page 893 sont également disponibles dans un fichier compressé ; téléchargez jm25529_.zip ou jm25529_.tar.gz.

```
package com.ibm.mq.id;
public class TryMyRecord {
    public static void main(String[] args) throws Exception {
        MyProducer producer = new MyProducer();
        MyRecord outrec = new MyRecord();
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        outrec.put(producer);
        System.out.println("Out flags " + outrec.getFlags() + " text "
                + outrec.getRecordData() + " Encoding "
                + producer.getEncoding() + " CCSID " + producer.getCCSID()
                + " MQ " + producer.getMQDest());
        MyRecord inrec = MyRecord.get(new MyConsumer());
        System.out.println("In  flags " + inrec.getFlags() + " text "
                + inrec.getRecordData() + " Encoding "
                + inrec.getMessageEncoding() + " CCSID "
                + inrec.getMessageCharset());
    }
}
```

Figure 157. `TryMyRecord`

```
package com.ibm.mq.id;
import java.io.IOException;
import java.io.Serializable;
import java.io.UnsupportedEncodingException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.constants.MQConstants;
import com.ibm.mq.headers.MQDataException;
import com.ibm.msg.client.wmq.WMQConstants;

public abstract class RECORD implements Serializable {
    private static final long serialVersionUID = -1616617232750561712L;
    protected final static int UTF8 = 1208;
    protected final static int MQLONG_LENGTH = 4;
    protected final static int RECORD_STRUCT_ID_LENGTH = 4;
    protected final static int RECORD_VERSION_1 = 1;
    protected final String RECORD_STRUCT_ID = "BLNK";
    protected final String RECORD_TYPE = "BLANK   ";
    private String structID = RECORD_STRUCT_ID;
    private int version = RECORD_VERSION_1;
    private int structLength = RECORD_STRUCT_ID_LENGTH + MQLONG_LENGTH * 2;
    private int headerEncoding = WMQConstants.WMQ_ENCODING_NATIVE;
    private String headerCharset = "UTF-8";
    private String headerFormat = RECORD_TYPE;

    public RECORD() {
        super();
    }

    public RECORD(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super();
        setHeaderCharset(message.getStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET));
        setHeaderEncoding(message.getIntProperty(WMQConstants.JMS_IBM_ENCODING));
        byte[] structID = new byte[RECORD_STRUCT_ID_LENGTH];
        message.readBytes(structID, RECORD_STRUCT_ID_LENGTH);
        setStructID(new String(structID, getMessageCharset()));
        setVersion(message.readInt());
        setStructLength(message.readInt());
    }

    public String getHeaderFormat() { return headerFormat; }
    public int getHeaderEncoding() { return headerEncoding;    }
    public String getMessageCharset() { return headerCharset; }
    public int getMessageEncoding() { return headerEncoding; }
    public String getStructID() { return structID; }
    public int getStructLength() { return structLength; }
    public int getVersion() { return version; }


    protected BytesMessage put(MyProducer myProducer) throws IOException,
            JMSException, UnsupportedEncodingException {
        setHeaderEncoding(myProducer.getEncoding());
        setHeaderCharset(myProducer.getCharset());
        myProducer.setMQClient(true);
        BytesMessage bytes = myProducer.session.createBytesMessage();
        bytes.setStringProperty(WMQConstants.JMS_IBM_FORMAT, getHeaderFormat());
        bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getHeaderEncoding());
        bytes.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET,
                             myProducer.getCCSID());
        bytes.writeBytes(String.format("%1$-" + RECORD_STRUCT_ID_LENGTH + "."
                         + RECORD_STRUCT_ID_LENGTH + "s", getStructID())
                         .getBytes(getMessageCharset()), 0, RECORD_STRUCT_ID_LENGTH);
        bytes.writeInt(getVersion());
        bytes.writeInt(getStructLength());
        return bytes;
    }

    public void setHeaderCharset(String charset) {
        this.headerCharset = charset; }
    public void setHeaderEncoding(int encoding) {
        this.headerEncoding = encoding; }
    public void setHeaderFormat(String headerFormat) {
        this.headerFormat = headerFormat; }
    public void setStructID(String structID) {
        this.structID = structID; }
    public void setStructLength(int structLength) {
        this.structLength = structLength; }
    public void setVersion(int version) {
        this.version = version; }
}
```

Figure 158. RECORD

```
package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;

public class MyRecord extends RECORD {
    private static final long serialVersionUID = -370551723162299429L;
    private final static int FLAGS = 1;
    private final static String STRUCT_ID = "MYRD";
    private final static int DATA_LENGTH = 32;
    private final static String FORMAT = "MYRECORD";
    private int flags = FLAGS;
    private String recordData = "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345";

    public MyRecord() {
        super();
        super.setStructID(STRUCT_ID);
        super.setHeaderFormat(FORMAT);
        super.setStructLength(super.getStructLength() + MQLONG_LENGTH
                + DATA_LENGTH);
    }

    public MyRecord(BytesMessage message) throws JMSException, IOException,
            MQDataException {
        super(message);
        setFlags(message.readInt());
        byte[] recordData = new byte[DATA_LENGTH];
        message.readBytes(recordData, DATA_LENGTH);
        setRecordData(new String(recordData, super.getMessageCharset()));
    }

    public static MyRecord get(MyConsumer myConsumer) throws JMSException,
            MQDataException, IOException {
        BytesMessage message = (BytesMessage) myConsumer.receive();
        return new MyRecord(message);
    }

    public int getFlags() { return flags; }
    public String getRecordData() { return recordData; }Â .


    public BytesMessage put(MyProducer myProducer) throws JMSException,
            IOException {
        BytesMessage bytes = super.put(myProducer);
        bytes.writeInt(getFlags());
        bytes.writeBytes(String.format("%1$-" + DATA_LENGTH + "."
                        + DATA_LENGTH + "s",getRecordData())
                        .getBytes(super.getMessageCharset()), 0, DATA_LENGTH);
        myProducer.send(bytes);
        return bytes;
    }

    public void setFlags(int flags) {
        this.flags = flags; }
    public void setRecordData(String recordData) {
        this.recordData = recordData; }
}
```

Figure 159. MyRecord

```
package com.ibm.mq.id;
import java.io.UnsupportedEncodingException;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.ibm.mq.headers.CCSID;
import com.ibm.mq.jms.MQDestination;
import com.ibm.msg.client.wmq.WMQConstants;
public abstract class EndPoint {
    public Context ctx;
    public ConnectionFactory cf;
    public Connection connection;
    public Destination destination;
    public Session session;
    protected EndPoint() throws NamingException, JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup("QM1");
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup("Q1");
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); }
    protected EndPoint(String cFactory, String dest) throws NamingException,
            JMSException {
        System.setProperty("java.naming.provider.url", "file:/C:/JNDI-Directory");
        System.setProperty("java.naming.factory.initial",
                           "com.sun.jndi.fscontext.RefFSContextFactory");
        ctx = new InitialContext();
        cf = (ConnectionFactory) ctx.lookup(cFactory);
        connection = cf.createConnection();
        destination = (Destination) ctx.lookup(dest);
        ((MQDestination)destination).setReceiveConversion
                                    (WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR);
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  }
    public int getCCSID() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_CCSID));  }
    public String getCharset() throws UnsupportedEncodingException,
            JMSException {
        return CCSID.getCodepage(getCCSID());  }
    public int getEncoding() throws JMSException {
        return (((MQDestination) destination)
                .getIntProperty(WMQConstants.WMQ_ENCODING));  }
    public boolean getMQDest() throws JMSException {
        if ((((MQDestination) destination).getMessageBodyStyle()
               == WMQConstants.WMQ_MESSAGE_BODY_MQ)
            || ((((MQDestination) destination).getMessageBodyStyle()
                  == WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED)
               && (((MQDestination) destination).getTargetClient()
                   == WMQConstants.WMQ_TARGET_DEST_MQ)))
            return true;
        else
            return false; }
    public void setCCSID(int ccsid) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_CCSID,
                ccsid); }
    public void setEncoding(int encoding) throws JMSException {
        ((MQDestination) destination).setIntProperty(WMQConstants.WMQ_ENCODING,
                encoding); }
    public void setMQBody() throws JMSException {
        ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);  }
    public void setMQBody(boolean mqbody) throws JMSException {
        if (mqbody)  ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
        else         ((MQDestination) destination)
                    .setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_JMS);  }
    public void setMQClient(boolean mqclient) throws JMSException {
       if (mqclient){
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
            if (!getMQDest()) setMQBody();
       }
       else
            ((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_JMS); }
    }
```

Figure 160. EndPoint

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.naming.NamingException;
public class MyProducer extends EndPoint {
    public MessageProducer producer;
    public MyProducer() throws NamingException, JMSException {
        super();
        producer = session.createProducer(destination); }
    public MyProducer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        producer = session.createProducer(destination); }
    public void send(Message message) throws JMSException {
        producer.send(message); }
}
```

Figure 161. MyProducer

```
package com.ibm.mq.id;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.naming.NamingException;
public class MyConsumer extends EndPoint {
    public MessageConsumer consumer;
    public MyConsumer() throws NamingException, JMSException {
        super();
        consumer = session.createConsumer(destination);
        connection.start();  }
    public MyConsumer(String cFactory, String dest) throws NamingException,
            JMSException {
        super(cFactory, dest);
        consumer = session.createConsumer(destination);
        connection.start();  }
    public Message receive() throws JMSException {
        return consumer.receive();  }
}
```

Figure 162. MyConsumer

Création et configuration de fabriques de connexions et de destinations dans une application WebSphere MQ classes for JMS

Une application WebSphere MQ classes for JMS 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 JMS IBM ou à l'aide des extensions JMS WebSphere MQ . Une application peut également utiliser les extensions JMS IBM ou WebSphere MQ 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. 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 JMS WebSphere MQ ou WebSphere 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 JMS IBM ou WebSphere MQ .

Utilisation des extensions JMS IBM
Une application peut utiliser les extensions JMS IBM 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 JMS WebSphere MQ
Une application peut également utiliser les extensions JMS WebSphere MQ 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.

Utilisation de JNDI pour extraire des objets gérés dans une application JMS

Pour extraire des objets gérés à partir d'un espace de nom JNDI (Java Naming and Directory Interface), une application JMS 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. L'administrateur peut utiliser l'outil d'administration JMS de WebSphere MQ ou WebSphere MQ Explorer pour créer et maintenir les objets gérés dans un espace de nom JNDI. Pour plus d'informations sur l'utilisation de l'outil d'administration JMS WebSphere MQ , voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965. Pour plus d'informations sur l'utilisation de WebSphere MQ Explorer, voir l'aide fournie avec WebSphere MQ Explorer. Toutefois, un serveur d'applications fournit généralement son propre référentiel pour les objets gérés et ses propres outils pour la création et la maintenance 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:

```
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

Utilisation des extensions JMS IBM

WebSphere MQ classes for JMS contient un ensemble d'extensions de l'API JMS appelé IBM extensions 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 WebSphere MQ classes for JMS. Les extensions peuvent être utilisées avec n'importe quel fournisseur de messagerie.

Les extensions JMS IBM sont un ensemble d'interfaces et de classes dans les packages suivants:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Les packages se trouvent dans `com.ibm.mqjms.jar` , qui se trouve dans `<MQ_Install_Dir>/java/lib`.

Ces extensions fournissent la fonction suivante:

- Mécanisme basé sur une fabrique 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 classes WebSphere MQ pour les objets JMS
- 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 WebSphere MQ classes for JMS

En ce qui concerne 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 JMS IBM fournissent un ensemble alternatif d'interfaces aux extensions JMS WebSphere MQ . Toutefois, alors que les extensions JMS WebSphere MQ sont spécifiques au fournisseur de messagerie WebSphere MQ , les extensions JMS IBM ne sont pas spécifiques à WebSphere MQ et peuvent être utilisées avec n'importe quel fournisseur de messagerie au sein de l'architecture en couches décrite dans «Une architecture en couches», à la page 826.

L'interface com.ibm.msg.client.wmq.WMQConstants contient les définitions de constantes qu'une application peut utiliser lors de la définition des propriétés des objets WebSphere MQ classes for JMS à l'aide des extensions JMS IBM . L'interface contient des constantes pour le fournisseur de messagerie WebSphere 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 ont été exécutées:

```
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 JMS IBM , 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:

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Le paramètre de l'appel getInstance() est une constante qui identifie le fournisseur de messagerie WebSphere MQ en tant que 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 WebSphere MQ classes for JMS», à la page 897.

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 WebSphere 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 WebSphere MQ et, éventuellement, 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 WebSphere 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 909. Pour une autre méthode de définition des propriétés d'un objet JmsQueue , voir «Définition des propriétés des objets WebSphere MQ classes for JMS», à la page 897.

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 909. Pour une autre méthode de définition des propriétés d'un objet JmsTopic , voir «Définition des propriétés des objets WebSphere MQ classes for JMS», à la page 897.

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 WebSphere MQ classes for JMS

Pour définir les propriétés des objets WebSphere MQ classes for JMS à l'aide des extensions JMS IBM , une application utilise les méthodes de l'interface com.ibm.msg.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 du package com.ibm.mq.jms héritent également des méthodes de l'interface de contexte JmsProperty. Une application peut donc utiliser ces méthodes pour définir les propriétés des objets MQConnectionFactory, MQQueue et MQTopic.

Lorsqu'une application crée un objet WebSphere MQ classes for JMS, 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 valide pour la propriété, WebSphere MQ classes for JMS 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. WebSphere MQ classes for JMS émet une exception NumberFormatException pour les types de données primitives et renvoie la valeur null pour les types de données référencés.

Outre les propriétés prédéfinies d'un objet WebSphere MQ classes for JMS, une application peut définir ses propres propriétés. Ces propriétés définies par l'application sont ignorées par les classes WebSphere MQ pour JMS.

Pour plus d'informations sur les propriétés des objets WebSphere MQ classes for JMS, voir Propriétés des objets IBM WebSphere MQ classes for JMS.

Le code suivant est un exemple de définition des propriétés à l'aide des extensions JMS IBM . 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 de 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 WebSphere MQ classes for JMS, 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 WebSphere MQ classes for JMS sont présentées dans Tableau 122, à la page 899.

Tableau 122. 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
Caractère	String
double	String
float	double, chaîne
int	long, chaîne
long	String
court	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 (à l'exception de `char`) à condition que le format de la chaîne soit correct pour la conversion. Si une application tente de convertir une chaîne dont le format n'est pas correct, WebSphere MQ classes for JMS émet une exception NumberFormat.
- Si une application tente une conversion qui n'est pas prise en charge, WebSphere MQ classes for JMS émet 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, WebSphere MQ classes for JMS émet 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, WebSphere MQ classes for JMS émet une exception NumberFormat.

- Lors de la conversion d'une valeur numérique (y compris une valeur avec le type de données `byte`) en chaîne, la valeur est convertie en représentation de chaîne de la valeur sous forme de nombre décimal, et non en chaîne contenant le caractère ASCII de 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 WebSphere MQ classes for JMS», à la page 897. 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 JmsPropertypour définir et obtenir les propriétés des objets WebSphere MQ classes for JMS, l'application 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 JMS WebSphere MQ

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, WebSphere MQ classes for JMS stocke les noms et les valeurs de propriété en tant que valeurs littérales définies dans l'interface com.ibm.msg.client.wmq.WMQConstants . 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, WebSphere MQ classes for JMS doit 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, WebSphere MQ classes for

JMS doit convertir les noms du format d'entrée spécifié au format canonique et convertir les valeurs du format canonique au 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 WebSphere MQ classes for JMS 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 Map pour accéder aux propriétés d'un objet WebSphere MQ classes for JMS.

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 JMS WebSphere MQ

WebSphere MQ classes for JMS contient un ensemble d'extensions de l'API JMS appelé WebSphere MQ extensions 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.

WebSphere 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 JMS WebSphere MQ . 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.*;
```

Une application peut utiliser les extensions JMS WebSphere MQ 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 903.

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.

Pour une connexion en temps réel à un courtier, une application peut utiliser le code suivant:

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

Ce code suppose que le courtier s'exécute sur un système dont le nom d'hôte est HOST2 et qu'il est à l'écoute sur le port numéro 1507.

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 de WebSphere MQ classes for JMS objects .

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 WebSphere 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 WebSphere 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 WebSphere 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, WebSphere 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 WebSphere MQ et, éventuellement, 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 WebSphere MQ appelée Q3 appartenant 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 909. Pour une autre méthode de définition des propriétés d'un objet MQQueue, voir «Définition des propriétés des destinations», à la page 904.

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 909. Pour une autre méthode de définition des propriétés d'un objet MQTopic, voir «Définition des propriétés des destinations», à la page 904.

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 WebSphere MQ classes for JMS.

Génération d'une connexion dans une application JMS

Pour établir une connexion, 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 WebSphere 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();
```

Création d'une session dans une application JMS

Pour créer une session, 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 WebSphere 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.

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 WebSphere MQ ou une rubrique et, en option, 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 WebSphere 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 WebSphere 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. 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 Queue créé par cette instruction représente une file d'attente WebSphere 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, WebSphere 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 909.

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, WebSphere 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 909.

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, WebSphere MQ classes for JMS crée une rubrique dont le nom commence par les caractères TEMP/*tempTopicPrefix*, où *tempTopicPrefix* correspond à 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 WebSphere 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, WebSphere

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 WebSphere 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 WebSphere MQ .

topicName
Nom de la rubrique.

Pour connaître les règles de création de noms de rubrique, voir Règles de dénomination des objets IBM WebSphere 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 Utilisation 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 123, à la page 910 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 123, à la page 910 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 123, à la page 910 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 JMS WebSphere MQ utilise des constantes symboliques pour les valeurs des propriétés, les URI ne peuvent pas contenir de constantes symboliques.

Tableau 123. 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 caractère dans le corps d'un message lorsque WebSphere MQ classes for JMS transmet le message à la destination	• Tout identificateur de jeu de caractères codés pris en charge par WebSphere MQ.
codage	Représentation des données numériques dans le corps d'un message lorsque WebSphere MQ classes for JMS transfère le message à la destination	• Toute valeur valide pour la zone *Codage* dans un descripteur de message WebSphere MQ .

Tableau 123. 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**
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. A chaque valeur est associée la valeur correspondante de la propriété MULTICAST telle qu'elle est utilisée dans l'outil d'administration JMS WebSphere MQ . Pour une description de la propriété MULTICAST et de ses valeurs valides, voir Propriétés des objets IBM WebSphere MQ classes for JMS. • -1-ASCF • 0 - désactivé • 3-NOTR • 5-FIABLE • 7-ACTIVE
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 rubrique WebSphere MQ . • 1-Non persistant. • 2-Persistant. • 3-Equivalent à la valeur HIGH pour la propriété PERSISTENCE telle qu'elle est utilisée dans l'outil d'administration JMS WebSphere MQ . Pour une explication de cette valeur, voir «Messages persistants JMS», à la page 935.

Tableau 123. 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
priorité	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 indiqué par l'attribut *DefPriority* de la file d'attente ou de la rubrique WebSphere 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 WebSphere 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 WebSphere 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», à la page 894
- «Utilisation des extensions JMS IBM», à la page 895
- «Utilisation des extensions JMS WebSphere MQ», à la page 902
- «Création de destinations dans une application JMS», à la page 907

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
- Mapper
- 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 836.

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, WebSphere 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 WebSphere MQ classes for JMS», à la page 951.

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», à la page 894
- «Utilisation des extensions JMS IBM», à la page 895
- «Utilisation des extensions JMS WebSphere MQ», à la page 902
- «Création de destinations dans une application JMS», à la page 907

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 WebSphere MQ classes for JMS, un abonné à une rubrique non durable reçoit également les publications conservées, mais pas lors de l'utilisation d'une connexion en temps réel à un courtier.

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 WebSphere 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 836.

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:

```
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)
- 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 WebSphere MQ classes for JMS», à la page 919.

V 7.5.0.8 Extraction des données utilisateur d'abonnement

Si les messages qu'une application IBM WebSphere 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.

Depuis la Version 7.5.0, Fix Pack 8, 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 WebSphere MQ classes for JMS . Pour permettre 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 javax.jms.Message.getStringProperty(java.lang.String) pour obtenir les données utilisateur d'abonnement.

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 WebSphere MQ classes for JMS peut appeler:

```
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

«En-tête MQRFH2 et JMS», à la page 840

Tâches associées

Définition d'un abonnement d'administration

Référence associée

DEFINE SUB

Interface JmsConstants

Fermeture d'une application WebSphere MQ classes for JMS

Il est important qu'une application WebSphere 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 classes WebSphere MQ pour JMS et WebSphere MQ de manière opportune, 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 des 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.

Traitement des messages incohérents dans IBM WebSphere MQ classes for JMS

Un message incohérent est un message qui ne peut pas être traité par une application MDB. Si un message incohérent est détecté, les objets JMS MessageConsumer et ConnectionConsumer peuvent le remettre en file d'attente en fonction de deux propriétés de file d'attente, BOQNAME et BOTHRESH.

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 *messages incohérents*. L'objet JMS MessageConsumer détecte les messages incohérents et les redirige vers une autre destination.

Le gestionnaire de files d'attente IBM WebSphere MQ conserve 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. Pour plus d'informations, voir «Suppression de messages de la file d'attente dans ASF», à la page 960.

Il existe une différence entre la manière dont les messages incohérents sont remis en file d'attente par MessageConsumers et ConnectionConsumers. Les ConnectionConsumers sont capables de remettre en file d'attente des messages incohérents sans affecter la distribution des messages. Le processus de remise en file d'attente se déroule en dehors de toute unité de travail associée à la distribution réelle des messages au code d'application. Cela est possible en raison de la nature à unités d'exécution multiples de l'opération ConnectionConsumer .

Les MessageConsumers, cependant, sont à unité d'exécution unique sous le niveau Session, et toute remise en file d'attente de messages incohérents a lieu dans l'unité de travail en cours. Cela n'affecte pas le fonctionnement de l'application, mais lorsque des messages incohérents sont remis en file d'attente

sous une session de transaction ou Client_accuse réception, l'action de remise en file d'attente elle-même n'est pas validée tant que l'unité de travail en cours n'est pas validée par le code d'application ou, le cas échéant, le code de conteneur d'application.

Les objets JMS ConnectionConsumer traitent les messages incohérents de la même manière et en utilisant 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 WebSphere 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 de publication / abonnement en mode normal du fournisseur de messagerie IBM WebSphere MQ , la file d'attente gérée de la rubrique est créée à partir de la file d'attente modèle.
- Pour la messagerie publication/abonnement en mode de migration de fournisseur de messagerie IBM WebSphere 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.

IBM WebSphere MQ classes for JMS interroge BackoutThreshold et BackoutRequeueQName de la file d'attente. Vous devez donc accorder l'accès en interrogation à la file d'attente à l'utilisateur exécutant l'application.

V 7.5.0.9 Si la file d'attente cible est une file d'attente de cluster, les droits requis dépendent de la version de IBM WebSphere MQ classes for JMS utilisée:

- Lors de l'utilisation de IBM WebSphere MQ classes for JMS for Version 7.5.0, Fix Pack 9 plus un correctif temporaire pour l'APAR IT26482, l'accès en consultation est requis.
- Pour toutes les autres versions, accordez des droits d'interrogation, de navigation et d'accès.

Pour définir les attributs BackoutThreshold et BackoutRequeueQName, entrez le commandes MQSC suivantes :

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold value) BOQNAME(your.backout.queue.name)
```

Si l'attribut BackoutThreshold est défini sur une valeur différente de zéro, pour éviter un comportement inattendu, définissez l'attribut QName BackoutRequeuesur un nom de file d'attente valide.

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 WebSphere MQ classes for JMS , SYSTEM.JMS.MODEL.QUEUE. Pour modifier ces paramètres, utilisez :

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold value) BOQNAME(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 accéder à la file d'attente d'annulation, le message est envoyé à la file d'attente des messages non livrés ou il est supprimé. 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. Pour plus de détails, voir «Suppression de messages de la file d'attente dans ASF», à la page 960.

Lorsqu'un message est replacé dans la file d'attente de remise en file d'attente d'annulation, certaines des valeurs de zone du descripteur de message (MQMD) du message changent. Voir MQMD-Descripteur de message pour plus de détails sur le format de MQMD.

Les zones MQMD suivantes changent de valeur lorsque le message est placé dans la file d'attente d'annulation.

- PutDate est mis à jour à la date à laquelle il est placé dans la file d'attente de remise en file d'attente des annulations.
- PutTime est mis à jour en fonction de l'heure à laquelle il est placé dans la file de remise en file d'attente d'annulation.
- Le nombre d'annulations est réinitialisé à zéro.
- L'expiration du message est mise à jour pour refléter l'expiration restante au moment où le message d'origine a été reçu par l'application JMS.

Les valeurs des zones suivantes restent les mêmes lorsque le message passe dans la file d'attente d'annulation:

- StructId
- Version
- Rapport
- MessageType
- Commentaires
- Codage
- CodedCharSetId
- MsgId
- CorrelId
- ReplyToQ
- ReplyToQMgr
- Format
- Persistance
- Priorité

Exceptions dans IBM WebSphere MQ classes for JMS

Une application IBM WebSphere MQ classes for JMS doit pouvoir gérer les exceptions émises par des appels d'API JMS ou distribuées à un gestionnaire d'exceptions.

IBM WebSphere MQ classes for JMS signale les problèmes d'exécution en générant des exceptions. JMSException est la classe racine des exceptions émises par les méthodes JMS, et l'interception des exceptions JMSException fournit un moyen générique de traiter toutes les exceptions liées à JMS.

Chaque exception JMSException encapsule les informations suivantes :

- Un message d'exception spécifique au fournisseur, obtenu par une application en appelant la méthode Throwable.getMessage().
- Un code d'erreur spécifique au fournisseur, obtenu par une application en appelant la méthode JMSException.getErrorCode().
- Une exception associée. Une exception émise par un appel d'API JMS est souvent le résultat d'un problème de niveau inférieur, qui est signalé par une autre exception liée à cette exception. Une application obtient une exception associée en appelant la méthode JMSException.getLinkedException() ou Throwable.getCause().

La plupart des exceptions émises par IBM WebSphere MQ classes for JMS sont des instances des sous-classes de JMSException. Ces sous-classes implémentent l'interface com.ibm.msg.client.jms.JmsExceptionDetail qui fournit les informations supplémentaires suivantes :

- Une explication du message d'exception, obtenue par une application en appelant la méthode JmsExceptionDetail.getExplanation().

- Une réponse d'utilisateur recommandée à l'exception, obtenue par une application en appelant la méthode JmsExceptionDetail.getUserAction().
- Les clés des insertions de message dans le message d'exception. Une application obtient un itérateur pour toutes les clés en appelant la méthode JmsExceptionDetail.getKeys().
- Les insertions de message dans le message d'exception. Par exemple, il se peut qu'une insertion de message corresponde au nom de la file d'attente à l'origine de l'exception et il convient peut-être qu'une application puisse accéder à ce nom. Une application obtient l'insertion de message correspondant à une clé spécifique en appelant la méthode JmsExceptionDetail.getValue().

Toutes les méthodes de l'interface JmsExceptionDetail risquent de renvoyer 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 WebSphere 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 WebSphere MQ reported an
              error.
User Action : Use the linked exception to determine the cause of this error. Check
              that the specified queue and queue manager are defined correctly.
```

L'exception émise, com.ibm.msg.client.jms.DetailedInvalidDestinationException, est une sous-classes de javax.jms.InvalidDestinationException et implémente l'interface com.ibm.msg.client.jms.JmsExceptionDetail.

Exceptions associées

Une exception associée fournit des informations supplémentaires sur un incident d'exécution. Par conséquent, pour chaque exception JMSException émise, une application doit vérifier l'exception associée. 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(). Pour une exception JMSException, la méthode getLinkedException() est réellement déléguée 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
                   |
                   +--->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 WebSphere MQ classes for JMS.
- L'exception suivante, une instance de com.ibm.mq.MQException, est émise par le fournisseur de messagerie IBM WebSphere MQ.
- L'exception suivante, une instance de com.ibm.mq.jmqi.JmqiException, est émise par l'interface Java commune à l'interface MQI.
- 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 WebSphere MQ classes for JMS, voir «Classes IBM WebSphere MQ pour architecture JMS», à la page 826.

A l'aide d'un similaire au code suivant, une application peut parcourir cette chaîne pour extraire toutes les informations appropriées :

```
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();
    }
}
```

Il est à noter qu'une application doit systématiquement 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 WebSphere MQ sur un problème

Les instances de com.ibm.mq.MQException et com.ibm.mq.jmqi.JmqiException encapsulent des informations spécifiques à IBM WebSphere MQ sur un problème.

Une exception MQException encapsule les informations suivantes :

- Un code achèvement, obtenu par une application en appelant la méthode getCompCode()
- Un code anomalie, obtenu par une application en appelant la méthode getReason()

Une exception JmqiException encapsule également un code achèvement et un code anomalie. En outre, une exception JmqiException encapsule les informations dans un message AMQ*nnnn* ou CSQ*nnnn* , s'il est associé à l'exception. En appelant les méthodes appropriées de l'exception, une application peut obtenir les divers composants de ce message, tels que la gravité, l'explication et l'action utilisateur.

Pour consulter des exemples d'utilisation des méthodes mentionnées dans cette section, reportez-vous à l'exemple de code dans «Exceptions associées», à la page 922.

Mise à niveau à partir de versions précédentes d' IBM WebSphere MQ classes for JMS

Par rapport aux versions précédentes de IBM WebSphere MQ classes for JMS, la plupart des codes d'erreur et des messages d'exception ont été modifiés dans la version 7. La raison de ces modifications est que IBM WebSphere MQ classes for JMS possède désormais une architecture en couches et que des exceptions sont émises à partir de différentes couches du code.

Par exemple, si une application tente de se connecter à un gestionnaire de files d'attente inexistant, une version précédente d'IBM WebSphere MQ classes for JMS émet une exception JMSException avec les informations suivantes :

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

Cette exception contenait une exception MQException associée comportant les informations suivantes :

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

Par comparaison dans les mêmes circonstances, la version 7 de IBM WebSphere MQ classes for JMS émet une exception JMSException avec les informations suivantes:

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

Cette exception contient une exception MQException associée comportant les informations suivantes :

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

Si votre application analyse ou teste les messages d'exception renvoyés par la méthode Throwable.getMessage() ou les codes d'erreur renvoyés par la méthode JMSException.getErrorCode() et que vous effectuez une mise à niveau à partir d'une édition antérieure à la version 7, votre application doit probablement être modifiée pour utiliser la version 7 de IBM WebSphere MQ classes for JMS.

Programmes d'écoute des exceptions

Une application peut connecter un programme d'écoute des exceptions à un objet Connexion. Par la suite, si un problème rendant la connexion inutilisable se produit, IBM WebSphere MQ classes for JMS envoie une exception au programme d'écoute des exceptions en appelant sa méthode onException(). L'application a ensuite l'opportunité de ré-établir la connexion.

V 7.5.0.8 Le correctif APAR IT14820, inclus depuis IBM WebSphere MQ Version 7.5.0, groupe de correctifs 8, a corrigé l'incident suivant : le programme d'écoute des exceptions JMS d'une application n'était pas appelé pour les exceptions non liées à une connexion interrompue (par exemple, MQRC_GET_INHIBITED), bien que la propriété ASYNC_EXCEPTIONS de la fabrique de connexions JMS utilisée par l'application avait pour valeur ASYNC_EXCEPTIONS_ALL. Il s'agissait de la valeur par défaut avant la Version 7.5.0, Fix Pack 8.

V 7.5.0.8 Pour conserver le comportement des applications JMS en cours qui configurent un MessageListener JMS et un ExceptionListenerJMS, et pour s'assurer que les IBM WebSphere MQ classes for JMS sont cohérents avec la spécification JMS, la valeur par défaut de la propriété ConnectionFactory JMS ASYNC_EXCEPTIONS a été remplacée par ASYNC_EXCEPTIONS_CONNECTIONBROKEN pour IBM WebSphere MQ classes for JMS. Par conséquent, par défaut, seules les exceptions correspondant à des

codes d'erreur de connexion interrompue sont distribuées au programme d'écoute des exceptions JMS d'une application.

V 7.5.0.8 Depuis la Version 7.5.0, Fix Pack 8, les IBM WebSphere MQ classes for JMS ont également été mises à jour de sorte que les exceptions JMS relatives aux erreurs non liées à une connexion interrompue survenant au cours de la distribution des messages aux consommateurs de messages asynchrones continuent d'être distribuées à un programme d'écoute des exceptions enregistré, lorsque la propriété ASYNC_EXCEPTIONS a pour valeur ASYNC_EXCEPTIONS_ALL dans la fabrique de connexions JMS utilisée par l'application.

V 7.5.0.8 Pour plus d'informations sur les modifications apportées aux programmes d'écoute des exceptions pour Version 7.5.0, Fix Pack 8 et sur les raisons pour lesquelles les modifications ont été apportées à partir de versions antérieures, voir JMS: Modifications apportées aux programmes d'écoute des exceptions dans la version 7.5.

Pour tout autre type de problème, une exception JMSException est émise par l'appel d'API JMS en cours.

Si une application n'enregistre pas de programme d'écoute des exceptions avec un objet Connection, toutes les exceptions qui auraient été distribuées au programme d'écoute des exceptions sont consignées dans le journal IBM WebSphere MQ classes for JMS .

Référence associée

ASYNCEXCEPTION

Erreurs de journalisation dans WebSphere MQ classes for JMS

Les informations sur les problèmes d'exécution qui peuvent nécessiter une action corrective de la part de l'utilisateur sont consignées dans le journal WebSphere MQ classes for JMS.

Par exemple, si une application tente de définir une propriété d'une fabrique de connexions, mais que le nom de la propriété n'est pas reconnu, WebSphere MQ classes for JMS écrit des informations sur le problème dans son journal.

Par défaut, le fichier contenant le journal est appelé mqjms.log et se trouve dans le répertoire de travail en cours. Toutefois, vous pouvez modifier le nom et l'emplacement du fichier journal en définissant la propriété com.ibm.msg.client.commonservices.log.outputName dans le fichier de configuration WebSphere MQ classes for JMS. Pour plus d'informations sur le fichier de configuration WebSphere MQ classes for JMS, voir «Le fichier de configuration IBM WebSphere MQ classes for JMS», à la page 753et pour plus de détails sur les valeurs valides de la propriété com.ibm.msg.client.commonservices.log.outputName , voir «Consignation etIBM WebSphere MQ classes for JMS», à la page 824.

Technologie de prise en charge de la première défaillance (FFST) dans WebSphere MQ classes for JMS

Si une erreur interne grave se produit dans les classes WebSphere MQ pour JMS, des informations sur la technologie de prise en charge de la première défaillance (FFST) sont générées.

Les informations FFST sont écrites dans un fichier appelé JMSC*nnnn*.FDC, où *nnnn* est un nombre à quatre chiffres. Ce fichier se trouve dans un répertoire appelé FFDC, qui est un sous-répertoire du répertoire dans lequel la sortie de trace est écrite. Par défaut, la sortie de trace est écrite dans le répertoire de travail en cours, mais vous pouvez rediriger la sortie de trace vers un autre répertoire en définissant la propriété **com.ibm.msg.client.commonservices.trace.outputName** dans le fichier de configuration WebSphere MQ classes for JMS. Pour plus d'informations sur le fichier de configuration WebSphere MQ classes for JMS, voir «Le fichier de configuration IBM WebSphere MQ classes for JMS», à la page 753.

Si la fonction de trace est activée lorsque les informations FFST sont générées, les informations FFST sont également écrites dans le fichier de trace. Pour plus d'informations sur le traçage des programmes JMS, voir Traçage des applications IBM WebSphere MQ classes for JMS.

Pour supprimer la production des fichiers FFDC, définissez la propriété **com.ibm.msg.client.commonservices.ffst.suppress**comme suit:

0
Sortie de tous les fichiers FFDC (par défaut).

-1
Sortie uniquement des premiers fichiers FFDC d'un type particulier.

entier
Supprimez tous les fichiers FFDC à l'exception de ceux qui sont un multiple de ce nombre.

Accès aux fonctions WebSphere MQ à partir d'une application WebSphere MQ classes for JMS

WebSphere MQ classes for JMS fournit des fonctions permettant d'exploiter un certain nombre de fonctions de WebSphere 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 WebSphere 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 WebSphere MQ requièrent que des valeurs spécifiques soient définies dans le MQMD des messages qui leur sont envoyés. WebSphere 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" WebSphere 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 929.

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 124. 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 125. 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	• **Non** 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.	• **False** • True	setMQMDWriteactivé

Tableau 125. 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_READ ACTIVÉ	• **Non** 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.	• **False** • True	setMQMDReadactivé
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 d'abord effacées, en fonction de 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 126. Noms, descriptions et types de propriété

Propriété	Description	Type Java	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
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	Identificateur 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

Tableau 126. Noms, descriptions et types de propriété (suite)

Propriété	Description	Type Java	Lien vers la description complète
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. Il n'est donc 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 des propriétés de tableau d'octets sur un message enfreint la spécification JMS.

Accès aux données de message IBM WebSphere MQ à partir d'une application à l'aide de WebSphere MQ classes for JMS

Vous pouvez accéder aux données de message WebSphere MQ complètes dans une application à l'aide de IBM WebSphere 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 WebSphere MQ et les données de message suivantes.

Définissez la propriété `WMQ_MESSAGE_BODY` de la destination sur `WMQ_MESSAGE_BODY_MQ`pour 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 WebSphere 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, WebSphere MQ classes for JMS génère automatiquement un en-tête MQRFH2 en fonction des paramètres des propriétés et des zones d'en-tête JMSMessage .

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, WebSphere 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 WebSphere MQ est généré. Utilisez les propriétés suivantes :

- JMS_IBM_Format ou JMS_IBM_MQMD_Format: cette propriété indique le format de l'en-tête ou du contenu de l'application WebSphere MQ 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 ou du contenu de l'application WebSphere MQ qui démarre le corps du message JMS s'il n'existe pas d'en-tête Websphere MQ précédent.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: cette propriété indique le codage de l'en-tête WebSphere 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 JMS IBM WebSphere MQ .
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 JMS MQRFH2 . 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 163, à la page 933 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 "ëÈÊ'>...??>?"
```

Figure 163. Données de message et MQMD codées de manière incohérente

L'un des fragments de code dans Figure 164, à la page 933 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 164. 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 WebSphere 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 ou du contenu de l'application WebSphere MQ 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 ou du contenu de l'application WebSphere MQ qui démarre le corps du message JMS s'il n'existe pas d'en-tête Websphere MQ précédent.
- JMS_IBM_Encoding ou JMS_IBM_MQMD_Encoding: cette propriété indique le codage de l'en-tête WebSphere 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' MQRFH2 d'un message WebSphere MQ dans le cadre de la charge du message (c'est-à-dire dans le cadre du corps du message JMS).

Tableau 127. Noms et descriptions des propriétés

Propriété	Forme abrégée	Description
WMQ_MESSAGE_BODY	MBODY	Indique si une application JMS traite l' MQRFH2 d'un message WebSphere MQ dans le cadre de la charge du message (c'est-à-dire dans le cadre du corps du message JMS).

Tableau 128. 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, WebSphere MQ classes for JMS génère et inclut ou non 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, WebSphere MQ classes for JMS génère automatiquement un en-tête MQRFH2 et l'inclut dans le message WebSphere MQ . Lors de la réception, WebSphere MQ classes for JMS définit les propriétés de message JMS en fonction des valeurs dans MQRFH2 (le cas échéant) ; il ne présente pas MQRFH2 comme partie intégrante du corps du message JMS. • MQ Lors de l'envoi, WebSphere MQ classes for JMS ne génère pas de MQRFH2. Lors de la réception, WebSphere 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

Messages persistants JMS

Les applications WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 JMS WebSphere MQ pour définir cette valeur ou une application peut appeler la méthode Destination.setPersistence() en transmettant la valeur WMQConstants.WMQ_PER_NPHIGH comme 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 WebSphere MQ sous-jacente est définie sur NPMCLASS (HIGH), le message est placé dans la file d'attente en tant que message non persistant WebSphere 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 WebSphere MQ et un message non persistant JMS est inséré dans la file d'attente en tant que message non persistant WebSphere MQ .

Si un message persistant JMS est inséré dans une file d'attente en tant que message non persistant WebSphere 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 via 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, WebSphere MQ classes for JMS émet une exception InvalidDestinationsi une application tente de créer un consommateur de message pour une destination où la propriété PERSISTENCE a la valeur HIGH et la file d'attente WebSphere MQ sous-jacente est définie sur 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, WebSphere MQ classes for JMS émet un appel MQINQ pour déterminer si NPMCLASS (HIGH) est défini sur la file d'attente WebSphere MQ sous-jacente. L'application doit donc avoir le droit d'interroger la file d'attente. En outre, WebSphere MQ classes for JMS conserve le résultat de l'appel MQINQ jusqu'à la suppression de la destination 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, WebSphere MQ classes for JMS ne remarque pas le nouveau paramètre.

En autorisant les messages persistants JMS à être placés dans des files d'attente WebSphere MQ en tant que messages non persistants WebSphere MQ , vous gagnez des 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. Si vous souhaitez utiliser des files d'attente temporaires pour accepter des messages persistants, vous devez utiliser SYSTEM.JMS.TEMPQ.MODEL, ou remplacez la file d'attente modèle par une autre file d'attente de votre choix.

Utilisation de SSL (Secure Sockets Layer) avec WebSphere MQ classes for JMS

WebSphere Les classes MQ pour les applications JMS peuvent utiliser le chiffrement SSL. Pour ce faire, ils ont besoin d'un fournisseur JSSE.

WebSphere MQ pour les connexions JMS utilisant TRANSPORT (CLIENT) prennent en charge le chiffrement SSL (Secure Sockets Layer). SSL 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.

WebSphere MQ classes for JMS utilise JSSE (Java Secure Socket Extension) pour gérer le chiffrement SSL 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, 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 WebSphere 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 WebSphere MQ classes for JMS», à la page 946.

propriété d'objet SSLCIPHERSUITE
Définissez SSLCIPHERSUITE pour activer le chiffrement SSL sur un objet ConnectionFactory .

Pour activer le chiffrement SSL 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. «SSL CipherSpecs et CipherSuites dans JMS», à la page 940 contient une table mappant les CipherSpecs pris en charge par WebSphere MQ à leurs CipherSuites équivalentes connues de JSSE. Pour plus d'informations sur les CipherSpecs et les CipherSuites avec WebSphere MQ, voir Sécurité.

Par exemple, pour configurer un objet ConnectionFactory qui peut être utilisé pour créer une connexion via un canal MQI activé pour SSL avec un CipherSpec de RC4_MD5_EXPORT, exécutez la commande suivante pour JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

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 WebSphere MQ classes for JMS», à la page 946.

Propriété d'objet SSLFIPSREQUIRED
Si vous avez besoin d'une connexion pour utiliser une CipherSuite prise en charge par le fournisseur Java JSSE FIPS IBM (IBMJSSEFIPS), définissez la propriété SSLFIPSREQUIRED de la fabrique de connexions sur YES.

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 WebSphere 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 UNIX, Linux et 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 CRL sous Java 2 v1.4 ou ultérieure. L'exemple JMSAdmin suivant indique 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 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 WebSphere MQ classes for JMS.

Par exemple, pour configurer un objet ConnectionFactory qui peut être utilisé pour créer une connexion via un canal MQI activé par SSL avec une clé secrète qui est renégociée après 4 Mo de données transmises, 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 SSL pour une application, créez une fabrique SSLSocketFactory et configurez JMS pour l'utiliser.

Vous pouvez personnaliser d'autres aspects de la connexion SSL 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 agresseur peut imiter le serveur LDAP, ce qui induit 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 l'emplacement du magasin de clés ou du magasin de clés de confiance, les classes WebSphere MQ pour les applications 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.

SSL CipherSpecs et CipherSuites dans JMS
CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuiteséquivalentes.

Le Tableau 129, à la page 941 répertorie les CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuiteséquivalentes. Si la propriété ConnectionFactory SSLFIPSREQUIRED est définie sur NO, une application WebSphere MQ pour JMS peut se connecter à un gestionnaire de files d'attente si un CipherSpec pris en charge est spécifié à l'extrémité serveur du canal MQI et que le CipherSuite équivalent est spécifié à l'extrémité client. Si SSLFIPSREQUIRED est défini sur YES, la combinaison de CipherSpec et CipherSuite détermine si l'application peut se connecter au gestionnaire de files d'attente.

A l'extrémité serveur d'un canal MQI, le nom d'un CipherSpec peut être spécifié comme valeur du paramètre SSLCIPH dans une commande DEFINE CHANNEL CHLTYPE (SVRCONN). A l'extrémité client d'un canal MQI, le nom d'une CipherSuite peut être spécifié de l'une des manières suivantes:

- Une application peut appeler la méthode setSSLCipherSuite () d'un objet ConnectionFactory .
- A l'aide de l'outil d'administration JMS WebSphere MQ , vous pouvez définir la propriété SSLCIPHERSUITE d'un objet ConnectionFactory .

Tableau 129. CipherSpecs pris en charge par WebSphere MQ et leurs CipherSuites équivalentes		
CipherSpec	**Equivalent CipherSuite**	**Connexion possible si SFIPS [1] est défini sur YES?**
NULL_MD5	SSL_RSA_WITH_NULL_MD5	Non
NULL_SHA	SSL_RSA_WITH_NULL_SHA	Non
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	Non
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	Non
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	Non
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	Non
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	Non
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	Non
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	Non
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Non
TLS_RSA_WITH_NULL_SHA256	SSL_RSA_WITH_NULL_SHA256	Non [7]
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	Oui [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	Oui [5] [7]
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	Oui [5] [7]
AES_SHA_US[2]		
TLS_RSA_WITH_DES_CBC_SHA[8] [9]	SSL_RSA_WITH_DES_CBC_SHA	Non[3]
TLS_RSA_WITH_3DES_EDE_CBC_SHA[8]	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Oui
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	Non[4]
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	Non[6]

Remarques :

1. Lors de l'utilisation de l'outil d'administration JMS WebSphere MQ , SFIPS est le nom abrégé de la propriété ConnectionFactory SSLFIPSREQUIRED.
2. Ce CipherSpec n'a pas de CipherSuiteéquivalente.
3. Ce CipherSpec a été certifié FIPS 140-2 avant le 19th mai 2007.
4. Ce CipherSpec a été certifié FIPS 140-2 avant le 19th mai 2007. Le nom FIPS_WITH_DES_CBC_SHA est historique et reflète le fait que ce CipherSpec était auparavant conforme à FIPS (mais ne l'est plus). Ce CipherSpec est déprécié et son utilisation est déconseillée.
5. Ces CipherSpecs (TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) ne peuvent pas être utilisés pour sécuriser une connexion de WebSphere MQ Explorer à un gestionnaire de files d'attente sauf si les fichiers de règles sans restriction appropriés sont appliqués à l'environnement d'exécution Java utilisé par l'explorateur.

   Pour plus d'informations sur les fichiers de règles, voir Informations sur la sécurité .
6. Le nom FIPS_WITH_3DES_EDE_CBC_SHA est historique et reflète le fait que ce CipherSpec était auparavant conforme à FIPS (mais ne l'est plus). Ce CipherSpec est déprécié et son utilisation est déconseillée.

7. Ces CipherSpecs (TLS_RSA_WITH_NULL_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA256) requièrent IBM JREs 6.0 SR13 FP2 , 7.0 SR4 FP2 ou version ultérieure.
8. Ces CipherSpecs (TLS_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_DES_CBC_SHA, TLS_RSA_WITH_RC4_128_SHA256) peuvent utiliser SSLv3 ou TLS. Par défaut, lorsque FIPS n'est pas activé, SSLv3 est utilisé. Pour utiliser TLS, définissez la propriété système Java **com.ibm.mq.cfg.preferTLS** sur `true`.
9. 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.

Information associée

Comment indiquer que seuls les CipherSpecs certifiés FIPS sont utilisés lors de l'exécution sur MQI Client FIPS (Federal Information Processing Standards) pour UNIX, Linux et Windows

Ecriture d'exits de canal dans Java pour WebSphere MQ classes for JMS

Vous créez des exits de canal en définissant des classes Java qui implémentent des interfaces spécifiées.

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 les classes WebSphere MQ pour 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 SSL 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 SSL est effectué *avant* l'appel des exits de canal.

Dans les versions de WebSphere MQ classes for JMS antérieures à la version 7.0, les exits de canal ont été 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 WebSphere MQ classes for JMS pour l'utilisation des exits de canal

Une application IBM WebSphere 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 WebSphere MQ pour les communications entrantes,

- La propriété **sendexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM WebSphere 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 WebSphere MQ pour les communications entrantes,
- La propriété **receiveexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM WebSphere 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 WebSphere MQ pour les communications entrantes,
- La propriété **securityexit** sur un objet ConnectionFactory utilisé par l'adaptateur de ressources IBM WebSphere 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 WebSphere MQ JMS ou de IBM WebSphere 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 WebSphere MQ .
2. 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 WebSphere MQ , comme illustré dans la Tableau 130, à la page 944. 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 130. Répertoire des exits IBM WebSphere MQ

Plateforme	**Répertoire**
UNIX and Linux	`/var/mqm/exits` (exits de canal 32 bits) `/var/mqm/exits64` (exits de canal 64 bits)
Windows	*rép_données_installation*\exits où *rép_données_installation* est le répertoire que vous avez choisi pour les fichiers de données IBM WebSphere MQ lors de l'installation. Le répertoire par défaut est `C:\Program Files\IBM\WebSphere MQ`.

Remarque : Si un exit de canal existe dans plusieurs emplacements, IBM WebSphere 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 WebSphere 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 WebSphere 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 le gestionnaire de sécurité Java 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 le gestionnaire de sécurité Java.

Les interfaces `MQSendExit`, `MQReceiveExit`et `MQSecurityExit` fournies avec des versions de IBM WebSphere MQ antérieures à Version 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 413. Vous devez stocker les programmes d'exit de canal écrits en C ou C++ dans le répertoire indiqué dans Tableau 130, à la page 944.

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 WebSphere MQ classes for JMS», à la page 946.

Spécification des données utilisateur à transmettre aux exits de canal lors de l'utilisation de WebSphere 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 JMS WebSphere MQ ou de l'explorateur WebSphere MQ . 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 JMS WebSphere MQ ou de l'explorateur WebSphere MQ . 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 WebSphere MQ classes for JMS», à la page 946.

Utilisation d'une table de définition de canal du client avec IBM WebSphere MQ classes for JMS

Une application IBM WebSphere MQ classes for JMS peut utiliser des définitions de canal de connexion client qui sont 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 WebSphere MQ classes for JMS peut utiliser des définitions de canal de connexion client stockées dans une table de définitions de canal client. Ces définitions sont créées par des commandes IBM WebSphere MQ Script (MQSC) ou des commandes IBM WebSphere MQ Programmable Command Format (PCF). Lorsque l'application crée un objet Connection, IBM WebSphere 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. 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 JMS IBM WebSphere MQ 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
- Un astérisque (*)
- 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 MQCONN. Vous pouvez définir la propriété QMANAGER à l'aide de l'outil d'administration JMS WebSphere MQ ou de l'explorateur IBM WebSphere MQ . Une application peut également définir la propriété en appelant la méthode setQueueManager () de l'objet ConnectionFactory .

Si une application crée ensuite un objet Connection à partir de l'objet ConnectionFactory , IBM WebSphere 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 WebSphere 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 SSL (Secure Sockets Layer):

- Un canal MQI utilise SSL 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 IBM WebSphere 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 WebSphere MQ classes for JMS utilise uniquement ces informations pour accéder aux serveurs LDAP qui contiennent des CRL.
- Une table de définition de canal du client peut également contenir l'emplacement d'un répondeur OCSP (Online Certificate Status Protocol). IBM WebSphere 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 indiqué à la section Using Online Certificate Protocol.

Pour plus d'informations sur l'utilisation de SSL avec une table de définition de canal du client, voir Utilisation du client transactionnel étendu avec des canaux SSL.

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 définition de canal 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 en Java, voir «Ecriture d'exits de canal dans Java pour WebSphere MQ classes for JMS», à la page 942.

  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.

Utilisez les propriétés `CONNECTIONNAMELIST` et `CLIENTRECONNECTOPTIONS` de la classe `MQConnectionFactory` pour configurer une connexion client afin qu'elle se reconnecte automatiquement à la suite d'un échec de connexion ou d'une demande d'administration pour reconnecter les applications client après l'arrêt d'un gestionnaire de files d'attente.

La liste complète des noms de connexion dans une liste connectionNameest uniquement accessible aux méthodes set /getconnectionNamequi peuvent gérer une liste de noms de connexion. Les méthodes telles que get /setHostname qui ne gèrent pas les listes de noms, accèdent au premier nom de la liste.

Les connexions client reconnectables automatiquement ne deviennent reconnectables qu'une fois la connexion établie.

Le fait qu'une application continue à fonctionner correctement après avoir été reconnectée automatiquement dépend de sa conception. Lisez les rubriques connexes pour savoir comment concevoir des clients reconnectables. Certains clients existants peuvent fonctionner correctement sans modification à la suite d'une reconnexion automatique.

La reconnexion client automatique n'estpas prise en charge par WebSphere MQ classes for Java.

Afin d'éviter que tous les clients connectés à un gestionnaire de files d'attente défaillant ne se reconnectent simultanément, les tentatives de reconnexion sont retardées par des intervalles en partie fixes et en partie aléatoires.

Par défaut, les tentatives de reconnexion se produisent aux intervalles suivants:

1. La première tentative est effectuée après un délai initial d'une seconde, plus un élément aléatoire jusqu'à 250 millisecondes.
2. La seconde tentative est effectuée deux secondes, plus un intervalle aléatoire pouvant atteindre 500 millisecondes, après l'échec de la première tentative.
3. La troisième tentative est effectuée quatre secondes, plus un intervalle aléatoire d'une seconde maximum, après l'échec de la deuxième tentative.
4. La quatrième tentative est effectuée huit secondes, plus un intervalle aléatoire pouvant aller jusqu'à deux secondes, après l'échec de la troisième tentative.
5. La cinquième tentative est effectuée 16 secondes, plus un intervalle aléatoire pouvant aller jusqu'à quatre secondes, après l'échec de la quatrième tentative.
6. La sixième tentative, et toutes les tentatives suivantes sont effectuées 25 secondes, plus un intervalle aléatoire de jusqu'à six secondes et 250 millisecondes, après l'échec de la tentative précédente.

Ce processus de reconnexion se poursuit jusqu'à ce que le client soit reconnecté au gestionnaire de files d'attente ou jusqu'à ce que l'intervalle maximal de reconnexion soit écoulé.

Si vous devez augmenter les valeurs par défaut, afin de refléter plus précisément le temps nécessaire à la reprise du gestionnaire de files d'attente ou à l'activation du gestionnaire de files d'attente de secours, modifiez les valeurs de délai dans MQCLIENT.INI à l'aide de l'attribut **ReconDelay** .

Concepts associés

Reconnexion client automatisée

Tâches associées

Configuration d'un client à l'aide d'un fichier de configuration

Partage d'une connexion TCP/IP dans IBM WebSphere 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 WebSphere MQ classes for JMS ou l'adaptateur de ressources IBM WebSphere MQ pour se connecter à un gestionnaire de files d'attente à l'aide du transport CLIENT peuvent être utilisées pour partager la même instance de canal.

Il existe une relation un à un entre les instances de canal et les connexions TCP/IP. Une connexion TCP/IP est créée pour chaque 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 chaque 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 WebSphere MQ classes for JMS 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 du partage de connexion. Par conséquent, lorsqu'une valeur de SHARECNV supérieure à 0 est utilisée avec IBM WebSphere MQ classes for JMS, il n'est pas garanti qu'une nouvelle demande de connexion partagera toujours une connexion déjà établie.

Calcul du nombre d'instances de canal

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**> + 1) / <**SHARECNV**>

Où <**maxPoolDepth**> est la valeur de la propriété **maxPoolDepth** et <**SHARECNV**> est la valeur de la propriété **SHARECNV** sur le canal utilisé par la spécification d'activation.

Autres applications JMS

Nombre d'instances de canal = (< connexions JMS > + < sessions JMS >) / <**SHARECNV**>

Où < connexions JMS > est le nombre de connexions créées par l'application, où < sessions JMS > est le nombre de sessions JMS créées par l'application et <**SHARECNV**> est la valeur de la propriété **SHARECNV** sur le canal utilisé par la spécification d'activation.

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 WebSphere MQ classes for JMS ou IBM WebSphere MQ classes for JMS .

Exemple d'application JMS

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.

Spécification d'une plage de ports pour les connexions client dans WebSphere MQ classes for JMS

Utilisez la propriété LOCALADDRESS pour spécifier une plage de ports auxquels votre application peut être liée.

Lorsqu'une application WebSphere MQ classes for JMS tente de se connecter à un gestionnaire de files d'attente WebSphere 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 JMS WebSphere MQ 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS

Une application WebSphere MQ classes for JMS peut utiliser les fonctions WebSphere MQ pour compresser un en-tête de message ou des données.

La compression des données qui circulent sur un canal WebSphere MQ peut améliorer les performances du canal et réduire le trafic réseau. A l'aide de la fonction fournie avec WebSphere 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 WebSphere 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 WebSphere 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:

- Propriété de destination JMS PUTSYNCALLOWED (nom abrégé-PAALD).

  PUTASYNCALLOWED contrôle si les applications JMS peuvent insérer des messages de manière asynchrone, si la file d'attente ou la rubrique sous-jacente représentée par la destination JMS autorise cette option.
- La propriété de file d'attente ou de rubrique IBM WebSphere 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 valeurs requises pour l'activation de la fonctionnalité d'insertion asynchrone:

Tableau 131. Les propriétés PUTASYNCALLOWED et DEFPRESP déterminent si les messages sont insérés de manière asynchrone.			
Propriété de file d'attente WebSphere MQ	**PUTASYNCALLOWED = NON**	**PUTASYNCALLOWED = OUI**	**PUTASYNCALLOWED = AS_DEST, AS_Q_DEF ou AS_T_DEF**
DEFPRESP=SYNC	La fonctionnalité d'insertion asynchrone n'est pas activée	Fonctionnalité d'insertion asynchrone activée	La fonctionnalité d'insertion asynchrone n'est pas activée
DEFPRESP=ASYNC	La fonctionnalité d'insertion asynchrone n'est pas activée	Fonctionnalité d'insertion asynchrone activée	Fonctionnalité d'insertion asynchrone activée

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 WebSphere 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 WebSphere 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.

Utilisation de la lecture anticipée avec WebSphere MQ classes for JMS

La fonctionnalité de lecture anticipée fournie par WebSphere MQ permet aux messages non persistants reçus en dehors d'une transaction d'être envoyés à IBM WebSphere MQ classes for JMS avant qu'une application ne les demande. Le IBM WebSphere 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 WebSphere 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 WebSphere 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 valeurs requises pour l'activation de la fonctionnalité de lecture anticipée:

Tableau 132. Les propriétés READAHEADALLOWED et DEFREADA déterminent 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 destination WebSphere MQ	**READAHEADALLOWED = OUI**	**READAHEADALLOWED = NON**	**AS_DEST ou AS_Q_DEF ou AS_T_DEF**
Propriété de file d'attente WebSphere MQ			
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 WebSphere 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 WebSphere 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 WebSphere 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 WebSphere MQ classes for JMS dans la mémoire tampon interne créée pour la destination. Lorsque l'application demande le traitement du message suivant, IBM WebSphere MQ classes for JMS renvoie le message suivant dans la mémoire tampon interne.

IBM WebSphere 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 WebSphere 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 tous les messages restants dans la mémoire tampon interne sont supprimés.

Publications conservées dans WebSphere MQ classes for JMS

Un client WebSphere 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. Cette opération est effectuée dans WebSphere MQ classes for JMS en définissant la propriété entière 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 les classes WebSphere MQ pour JMS

JMS prend en charge les transactions compatibles XA dans les liaisons et les modes client avec un gestionnaire de transactions pris en charge.

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.

Utilisation d'une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker

Une application WebSphere MQ classes for JMS peut utiliser une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker pour la messagerie de publication / abonnement. Le courtier et WebSphere MQ classes for JMS doivent être configurés pour activer une connexion en temps réel.

Lorsqu'une application utilise une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker, l'application et le courtier échangent des messages à l'aide de WebSphere MQ Real-Time Transport. En fonction de la configuration, les messages peuvent également être distribués à l'application à l'aide de WebSphere MQ Multicast Transport.

Pour plus d'informations sur la façon dont une application peut se connecter à un gestionnaire de files d'attente WebSphere MQ et utiliser WebSphere MQ Enterprise Transport pour échanger des messages avec un courtier WebSphere Event Broker ou WebSphere Message Broker, voir la documentation des éditions précédentes de WebSphere MQ classes for JMS. Notez que pour utiliser WebSphere MQ Enterprise Transport, une application doit se connecter à un gestionnaire de files d'attente à l'aide d'une fabrique de connexions s'exécutant en mode de migration du fournisseur de messagerie WebSphere MQ .

Configuration d'un courtier WebSphere Event Broker ou WebSphere Message Broker pour une connexion en temps réel

Pour qu'une application WebSphere MQ classes for JMS utilise une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker, vous devez configurer le courtier en créant et en déployant un flux de messages pour lire les messages à partir du port TCP/IP sur lequel le courtier écoute et publie les messages. En fonction de vos besoins, vous devrez peut-être configurer le courtier de manière supplémentaire.

Pour configurer le courtier, vous devez créer et déployer l'un des flux de messages suivants:

- Flux de messages contenant un noeud de traitement de message de flux réeltimeOptimized
- Flux de messages contenant un noeud de traitement de message Real-timeInput et un noeud de traitement de message de publication

Vous devez configurer le noeud Real-timeOptimizedFlow ou Real-timeInput pour qu'il soit à l'écoute sur le port TCP/IP utilisé pour les connexions en temps réel. Par défaut, le numéro de port pour les connexions en temps réel est 1506.

Vous devez également configurer le courtier si vous avez l'une des exigences suivantes:

- Si vous souhaitez que l'application se connecte au courtier à l'aide de l'authentification SSL (Secure Sockets Layer)
- Si vous souhaitez que l'application se connecte au courtier à l'aide de la tunnellisation HTTP
- Si vous souhaitez que les messages soient distribués à un consommateur de messages à l'aide de la multidiffusion

Pour plus d'informations sur la configuration d'un courtier, voir la *documentation du produit WebSphere Event Broker* ou la *documentation du produit WebSphere Message Broker*.

Configuration de WebSphere MQ classes for JMS pour une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker

Pour qu'une application WebSphere MQ classes for JMS utilise une connexion en temps réel à un courtier WebSphere Event Broker ou WebSphere Message Broker, WebSphere MQ classes for JMS doit être configuré en définissant certaines propriétés de la fabrique de connexions. En fonction de vos besoins, WebSphere MQ classes for JMS peut avoir besoin d'être configuré de manière supplémentaire.

Pour configurer les classes WebSphere MQ pour JMS, les propriétés suivantes de la fabrique de connexions doivent être définies:

- La propriété TRANSPORT doit être définie sur DIRECT.

  Toutefois, pour qu'une application puisse se connecter à l'aide d'une tunnellisation HTTP, la propriété TRANSPORT doit être définie sur DIRECTIVE THTTP à la place. Voir «Utilisation de la tunnellisation HTTP», à la page 956.
- La propriété HOSTNAME doit être définie sur le nom d'hôte ou l'adresse IP du système sur lequel le courtier s'exécute.
- La propriété PORT doit être définie sur le numéro du port sur lequel le courtier écoute les connexions en temps réel.

Une application peut définir ces propriétés de manière dynamique lors de l'exécution à l'aide des extensions JMS IBM ou des extensions JMS WebSphere MQ . Sinon, si la fabrique de connexions est un objet géré, un administrateur peut définir ces propriétés à l'aide de l'outil d'administration JMS WebSphere MQ ou de WebSphere MQ Explorer.

Pour plus d'informations sur les propriétés et les méthodes utilisées par les applications pour définir leurs valeurs, voir Propriétés des objets IBM WebSphere MQ classes for JMS. Pour plus d'informations sur l'utilisation de l'outil d'administration JMS WebSphere MQ , voir «Utilisation de l'outil d'administration JMS WebSphere MQ», à la page 965. Pour plus d'informations sur l'utilisation de WebSphere MQ Explorer, voir l'aide fournie avec WebSphere MQ Explorer.

Si vous disposez de l'une des exigences suivantes, WebSphere MQ classes for JMS requiert une configuration supplémentaire:

- Si vous souhaitez qu'une application se connecte au courtier à l'aide de l'authentification SSL (Secure Sockets Layer)
- Si vous souhaitez qu'une application se connecte au courtier à l'aide de la tunnellisation HTTP
- Si vous souhaitez qu'une application se connecte au courtier via un serveur proxy
- Si vous souhaitez que les messages soient distribués à un consommateur de messages à l'aide de la multidiffusion

Les sections suivantes expliquent comment configurer WebSphere MQ classes for JMS pour chacune de ces exigences.

Utilisation de l'authentification SSL (Secure Sockets Layer)

L'authentification SSL peut être utilisée sur une connexion en temps réel à un courtier. Seule l'authentification est prise en charge pour ce type de connexion. Vous ne pouvez pas utiliser SSL pour chiffrer et déchiffrer les données de message qui transitent entre l'application et le courtier ou pour détecter la contrefaçon des données.

Notez la différence entre cette situation et celle qui se produit lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client. Dans ce dernier cas, vous pouvez utiliser la prise en charge SSL de WebSphere MQ pour chiffrer et déchiffrer les données de message qui circulent entre l'application et le gestionnaire de files d'attente et pour détecter la falsification des données, ainsi que pour fournir l'authentification.

Si vous souhaitez protéger les données de message sur une connexion en temps réel à un courtier, vous pouvez utiliser la fonction fournie par le courtier à la place. Vous pouvez affecter une valeur de qualité de protection (QoP) à chaque rubrique avec les messages que vous souhaitez protéger. Vous pouvez donc sélectionner un niveau de protection des messages différent pour chaque rubrique. Pour plus d'informations sur la protection des messages fournie par un courtier, voir la *documentation du produit WebSphere Event Broker* ou la *documentation du produit WebSphere Message Broker*.

Pour utiliser l'authentification SSL sur une connexion en temps réel à un courtier, la propriété DIRECTAUTH de la fabrique de connexions doit être définie sur CERTIFICATE.

Si vous souhaitez utiliser SSL pour l'authentification mutuelle, la propriété Type de protocole d'authentification du courtier doit spécifier l'option R pour SSL symétrique. Si vous souhaitez utiliser SSL uniquement pour l'authentification du courtier, la propriété Type de protocole d'authentification du courtier doit spécifier l'option S pour SSL asymétrique. Toutefois, dans ce cas, l'application doit se connecter au courtier en appelant createConnection() avec un ID utilisateur et un mot de passe comme paramètres, comme dans l'exemple suivant:

```
factory.createConnection("user1", "user1pw");
```

Le courtier utilise ensuite l'ID utilisateur et le mot de passe, au lieu de SSL, pour authentifier l'application. Pour plus d'informations sur la configuration du courtier pour l'authentification SSL, voir la *documentation du produit WebSphere Event Broker* ou la *documentation du produit WebSphere Message Broker*.

Remarques :

1. La valeur de la propriété DIRECTAUTH détermine si l'authentification SSL est utilisée sur une connexion en temps réel à un courtier, et non la valeur de la propriété SSLCIPHERSUITE.
2. Lorsque l'authentification SSL est utilisée sur une connexion en temps réel à un courtier, les propriétés SSLPEERNAME et SSLCRL sont utilisées pour effectuer les mêmes vérifications que celles effectuées lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client.
3. WebSphere Les classes MQ pour JMS peuvent utiliser la même configuration de magasin de clés et de magasin de clés de confiance JSSE (Java Secure Socket Extension) pour fournir la prise en charge SSL dans l'une des situations suivantes:
   - Lorsqu'une application utilise une connexion en temps réel à un courtier
   - Lorsqu'une application se connecte à un gestionnaire de files d'attente en mode client

Utilisation de la tunnellisation HTTP

Une application WebSphere MQ classes for JMS peut se connecter à un courtier à l'aide de la tunnellisation HTTP, ce qui signifie que l'application se connecte au courtier à l'aide du protocole HTTP comme si elle se connectant à un site Web.

Pour utiliser la tunnellisation HTTP sur une connexion en temps réel à un courtier, la propriété TRANSPORT de la fabrique de connexions doit être définie sur DIRECTIVE THTTP.

La tunnellisation HTTP ne peut pas être utilisée avec l'authentification SSL, la connexion via un serveur proxy ou la distribution de messages à l'aide de la multidiffusion. La version prise en charge du protocole HTTP est 1.0. HTTP version 1.1 n'est pas pris en charge.

Connexion via un serveur proxy

Une application WebSphere MQ classes for JMS peut utiliser une connexion en temps réel à un courtier en se connectant via un serveur proxy. WebSphere MQ classes for JMS se connecte directement au serveur proxy et utilise le protocole Internet défini dans RFC 2817 pour demander au serveur proxy de transmettre la demande de connexion au courtier.

Pour se connecter à un courtier via un serveur proxy, les propriétés suivantes de la fabrique de connexions doivent être définies:

- La propriété PROXYHOSTNAME doit être définie sur le nom d'hôte ou l'adresse IP du système sur lequel le serveur proxy s'exécute.
- La propriété PROXYPORT doit être définie sur le numéro du port sur lequel le serveur proxy est à l'écoute.

Si la propriété PROXYHOSTNAME n'est pas définie ou est définie sur la chaîne vide, WebSphere MQ classes for JMS tente de se connecter directement au courtier en utilisant uniquement les propriétés HOSTNAME et PORT et ne tente pas de se connecter via un serveur proxy.

Distribution de messages à l'aide de la multidiffusion

A l'aide d'une connexion en temps réel à un courtier, les messages peuvent être distribués à un consommateur de message à l'aide de la multidiffusion.

Pour activer la multidiffusion, la propriété MULTICAST de l'objet Topic doit être définie sur l'option de multidiffusion requise. Sinon, si la propriété MULTICAST de l'objet Topic est définie sur ASCF, la propriété MULTICAST de la fabrique de connexions doit être définie sur l'option de multidiffusion requise.

WebSphere MQ classes for JMS prend en charge les protocoles de multidiffusion PTL (Packet Transfer Layer) et PGM (Pragmatique General Multicast), et inclut la prise en charge des deux implémentations du protocole PGM, PGM/IP et PGM UDP encapsulé. Cependant, la prise en charge de PGM/IP est disponible uniquement sur les plateformes suivantes:

- AIX (32 bits uniquement)
- Linux (plateformex86)
- Linux (plateforme zSeries , 32 bits uniquement)
- Solaris SPARC (32 bits uniquement)
- Windows (32 bits uniquement)
- z/OS

WebSphere MQ classes for JMS Application Server Facilities

Cette rubrique décrit comment WebSphere 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.

WebSphere Les classes MQ pour JMS prennent en charge les fonctions ASF (Application Server Facilities) qui sont spécifiées dans *Java Message Service Specification, Version 1.1* (voir le site Web Java de Sun à l'adresse https://java.sun.com). Cette spécification identifie trois rôles dans ce modèle de programmation:

- **Le fournisseur JMS** 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.

ConnectionConsumer JMS

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 958
- «Principes généraux de la messagerie de publication / abonnement à l'aide d'ASF», à la page 959
- «Suppression de messages de la file d'attente dans ASF», à la page 960
- Traitement des messages incohérents dans ASF. Voir «Traitement des messages incohérents dans IBM WebSphere MQ classes for JMS», à la page 919.

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.

Dans les termes WebSphere 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 variable WebSphere MQ QLOCAL 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 960). 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 WebSphere 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 WebSphere MQ pour 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 Référence 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 WebSphere MQ classes for JMS est en mode de migration du fournisseur de messagerie WebSphere 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 WebSphere MQ classes for JMS est en mode de migration du fournisseur de messagerie WebSphere 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 WebSphere 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 WebSphere 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 indique que JMS ne peut pas effectuer d'analyse syntaxique.

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 WebSphere MQ de rebut (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 962. Les autres zones sont décrites dans MQDLH-Dead-letter header.

Gestion des messages incohérents dans ASF

Dans les fonctions du serveur d'applications, la gestion des messages incohérents est traitée de manière légèrement différente par rapport à d'autres classes de WebSphere MQ pour JMS.

Pour plus d'informations sur la gestion des messages incohérents dans les classes WebSphere MQ pour JMS, voir «Traitement des messages incohérents dans IBM WebSphere MQ classes for JMS», à la page 919.

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.

Le fournisseur JMS imbriqué dans WebSphere Application Server, version 5.0 et version 5.1 gère les messages incohérents d'une manière différente de celle décrite pour WebSphere MQ classes for JMS. Pour plus d'informations sur la façon dont le fournisseur JMS intégré gère les messages incohérents, voir la documentation du produit WebSphere Application Server appropriée.

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 962 et «Codes raison et commentaires en retour dans ASF», à la page 962.

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)

Raison
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 JMS de 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)

Raison
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)

Raison
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 Codes anomalie d'API.

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 165, à la page 964 récapitule les principes de la fonctionnalité ServerSessionPool et ServerSession .

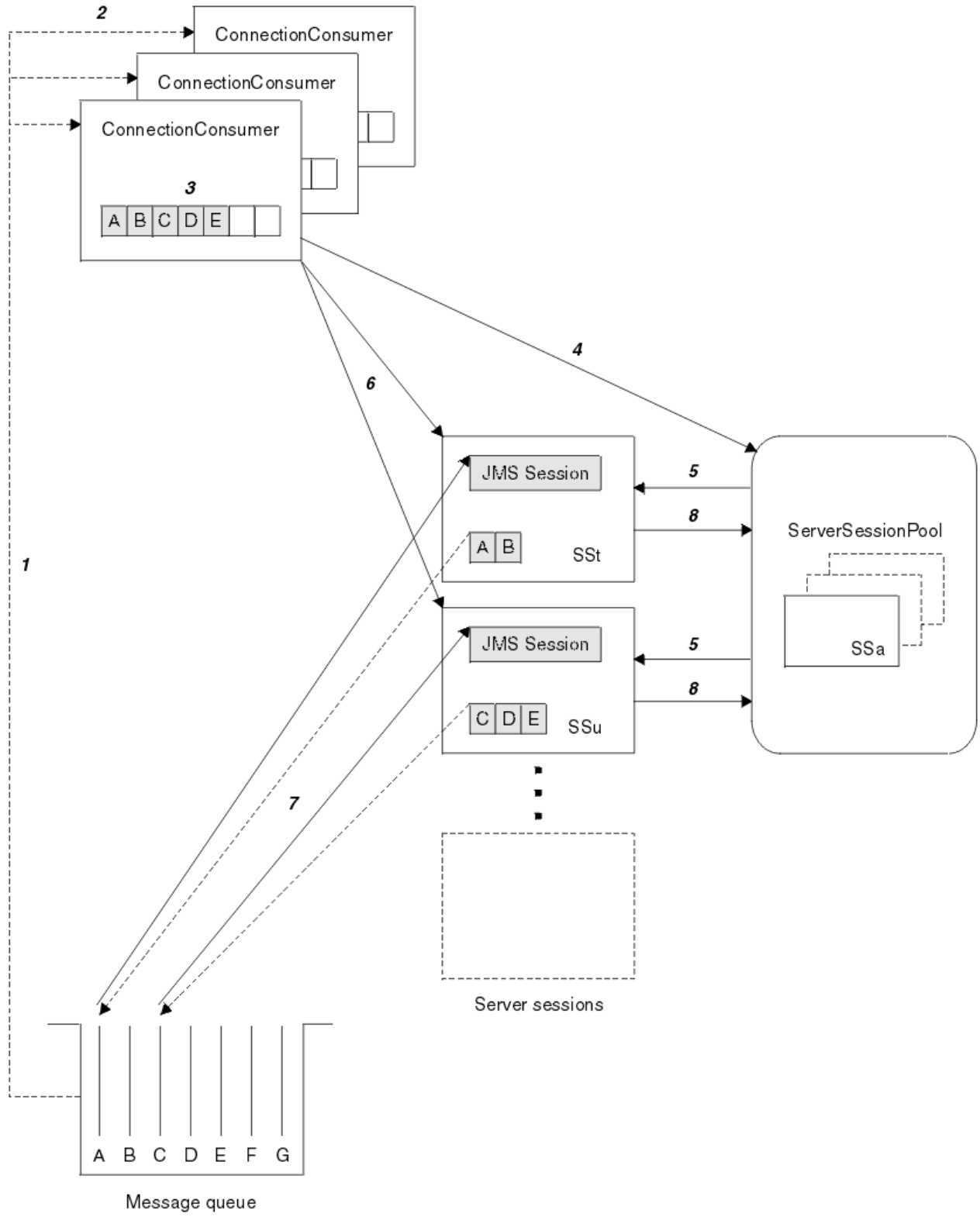


Figure 165. Fonctionnalité ServerSessionPool et ServerSession

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 .

Utilisation de l'outil d'administration JMS WebSphere MQ

L'outil d'administration permet de définir les propriétés de huit types d'objet WebSphere MQ classes for JMS et de les stocker dans un espace de nom JNDI. Les applications peuvent ensuite utiliser JNDI pour extraire ces objets gérés de l'espace de nom.

Les objets JMS des classes WebSphere MQ que vous pouvez administrer à l'aide de l'outil sont les suivants:

- MQConnectionFactory
- Fabrique MQQueueConnection
- Fabrique MQTopicConnection
- MQQUEUE
- Sujet MQ
- MQXAConnectionFactory
- Fabrique MQXAQueueConnection
- Fabrique MQXATopicConnection

Pour plus de détails sur ces objets, voir «Administration des objets JMS», à la page 970 .

Les types de propriété et les valeurs dont vous avez besoin pour utiliser cet outil sont répertoriés dans la section Propriétés des objets IBM WebSphere MQ classes for JMS.

L'outil permet également aux administrateurs de manipuler les sous-contextes d'espace de nom de répertoire dans JNDI. Voir «Manipulation de sous-contextes à l'aide de l'outil d'administration JMS WebSphere MQ», à la page 969.

Vous pouvez également créer et configurer des objets gérés JMS avec WebSphere MQ Explorer.

Appel de l'outil d'administration IBM WebSphere MQ classes for JMS

L'outil d'administration dispose d'une interface de ligne de commande. Vous pouvez l'utiliser de manière interactive ou l'utiliser pour démarrer un traitement par lots.

Le mode interactif fournit une invite de commande dans laquelle vous pouvez entrer des commandes d'administration. En mode de traitement par lots, la commande de démarrage de l'outil inclut le nom d'un fichier contenant un script de commande d'administration.

Mode interactif

Pour démarrer l'outil en mode interactif, entrez la commande suivante:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

où :

-t

Active la trace (la valeur par défaut est trace off)

Le fichier de trace est généré dans "%MQ_JAVA_DATA_PATH%"\errors (Windows) ou /var/mqm/trace (UNIX). Le nom du fichier de trace est au format suivant:

mqjms_*PID*.trc

où *PID* est l'ID de processus de la machine virtuelle Java.

-v
Produit une sortie prolixe (par défaut, il s'agit d'une sortie terse)

-cfg nom_fichier_config
Nomme un autre fichier de configuration. Si ce paramètre est omis, le fichier de configuration par défaut, JMSAdmin.config, est utilisé. (Voir «Configuration de l'outil d'administration JMS», à la page 966)

Une invite de commande s'affiche, indiquant que l'outil est prêt à accepter les commandes d'administration. Cette invite apparaît initialement comme suit:

```
InitCtx>
```

indiquant que le contexte en cours (c'est-à-dire le contexte JNDI auquel toutes les opérations de nommage et d'annuaire font actuellement référence) est le contexte initial défini dans le paramètre de configuration PROVIDER_URL (voir «Configuration de l'outil d'administration JMS», à la page 966).

Au fur et à mesure que vous parcourez l'espace de nom du répertoire, l'invite change pour refléter cela, de sorte que l'invite affiche toujours le contexte en cours.

Mode de traitement par lots

Pour démarrer l'outil en mode de traitement par lots, entrez la commande suivante:

```
JMSAdmin <test.scp
```

où *test.scp* est un fichier script qui contient des commandes d'administration (voir «Commandes d'administration dans l'outil d'administration JMS WebSphere MQ», à la page 968). La dernière commande du fichier doit être la commande END .

Configuration de l'outil d'administration JMS

L'outil d'administration JMS WebSphere MQ utilise un fichier de configuration pour définir les valeurs de certaines propriétés. Un exemple de fichier est fourni, que vous pouvez personnaliser en fonction de votre système.

Le fichier de configuration est un fichier en texte clair qui se compose d'un ensemble de paires clé-valeur, séparées par le signe égal (=). Ceci est illustré dans l'exemple suivant:

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(Un # dans la première colonne de la ligne indique un commentaire ou une ligne qui n'est pas utilisée.)

Un exemple de fichier de configuration est fourni avec WebSphere MQ. Le fichier est appelé `JMSAdmin.config`et se trouve dans le répertoire `<MQ_JAVA_INSTALL_PATH>/bin` . Editez ce fichier pour l'adapter à la configuration de votre système.

Configurez l'outil d'administration avec des valeurs pour les propriétés suivantes:

INITIAL_CONTEXT_FACTORY
Fournisseur de services utilisé par l'outil. Les valeurs prises en charge pour cette propriété sont les suivantes:

- com.sun.jndi.ldap.LdapCtxFactory (pour LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (pour le contexte du système de fichiers)

Vous pouvez également utiliser une fabrique InitialContextqui ne figure pas dans la liste précédente. Pour plus d'informations, voir «Utilisation d'une fabrique InitialContextnon répertoriée avec l'outil d'administration JMS WebSphere MQ», à la page 967.

PROVIDER_URL
URL du contexte initial de la session ; racine de toutes les opérations JNDI effectuées par l'outil. Deux formes de cette propriété sont prises en charge:

- ldap://nom_hôte/nom_contexte
- file: [unité:] /pathname

Le format de l'URL LDAP peut varier en fonction de votre fournisseur LDAP. Pour plus d'informations, consultez la documentation LDAP.

AUTHENTIFICATION_SÉCURITÉ_AUTHENTIFICATION
Indique si JNDI transmet les données d'identification de sécurité à votre fournisseur de services. Cette propriété est utilisée uniquement lorsqu'un fournisseur de services LDAP est utilisé. Cette propriété peut prendre l'une des trois valeurs suivantes:

- none (authentification anonyme)
- simple (authentification simple)
- CRAM-MD5 (mécanisme d'authentificationCRAM-MD5)

Si aucune valeur valide n'est fournie, la valeur par défaut de la propriété est none. Voir «Configuration de la sécurité pour l'outil d'administration JMS», à la page 967 pour plus de détails sur la sécurité à l'aide de l'outil d'administration.

Ces propriétés sont définies dans un fichier de configuration. Lorsque vous appelez l'outil, vous pouvez spécifier cette configuration à l'aide du paramètre de ligne de commande `-cfg` , comme décrit dans «Appel de l'outil d'administration IBM WebSphere MQ classes for JMS», à la page 965. Si vous n'indiquez pas de nom de fichier de configuration, l'outil tente de charger le fichier de configuration par défaut (`JMSAdmin.config`). Il recherche d'abord ce fichier dans le répertoire en cours, puis dans le répertoire `<MQ_JAVA_INSTALL_PATH>/bin` , où `<MQ_JAVA_INSTALL_PATH>` est le chemin d'accès à votre installation WebSphere MQ classes for JMS.

Utilisation d'une fabrique InitialContextnon répertoriée avec l'outil d'administration JMS WebSphere MQ

Deux valeurs de fabrique InitialContextsont prises en charge. Vous pouvez utiliser d'autres contextes JNDI en définissant des paramètres dans le fichier de configuration d'administration JMS.

Vous pouvez utiliser l'outil d'administration pour vous connecter à des contextes JNDI autres que ceux répertoriés dans «Configuration de l'outil d'administration JMS», à la page 966 à l'aide de trois paramètres définis dans le fichier de configuration JMSAdmin.

Pour utiliser une autre fabrique InitialContext:

1. Définissez la propriété INITIAL_CONTEXT_FACTORY sur le nom de classe requis.
2. Définissez le comportement de la fabrique InitialContextà l'aide des propriétés USE_INITIAL_DIR_CONTEXT, NAME_PREFIX et NAME_READABILITY_MARKER.

Les paramètres de ces propriétés sont décrits dans les exemples de commentaires de fichier de configuration.

Vous n'avez pas besoin de définir les trois propriétés répertoriées ici si vous utilisez l'une des valeurs INITIAL_CONTEXT_FACTORY prises en charge. Toutefois, vous pouvez leur attribuer des valeurs pour remplacer les valeurs par défaut du système. Si vous omettez une ou plusieurs des trois propriétés de la fabrique InitialContext, l'outil d'administration fournit des valeurs par défaut appropriées en fonction des valeurs des autres propriétés.

Configuration de la sécurité pour l'outil d'administration JMS

Utilisez la propriété SECURITY_AUTHENTICATION pour déterminer si les données d'identification de sécurité sont transmises au fournisseur de services.

La propriété SECURITY_AUTHENTICATION est décrite dans «Configuration de l'outil d'administration JMS», à la page 966. Son effet est le suivant:

- Si vous définissez ce paramètre sur `none`, JNDI ne transmet aucune donnée d'identification de sécurité au fournisseur de services et l' *authentification anonyme* est effectuée.
- Si vous définissez le paramètre sur `simple` ou `CRAM-MD5`, les données d'identification de sécurité sont transmises via JNDI au fournisseur de services sous-jacent. Ces données d'identification de sécurité se présentent sous la forme d'un nom distinctif d'utilisateur (nom distinctif d'utilisateur) et d'un mot de passe.

Si des données d'identification de sécurité sont requises, vous êtes invité à les indiquer lors de l'initialisation de l'outil. Pour éviter cela, définissez les propriétés PROVIDER_USERDN et PROVIDER_PASSWORD dans le fichier de configuration JMSAdmin.

Remarque : Si vous n'utilisez pas ces propriétés, le texte saisi, *incluant le mot de passe*, est renvoyé à l'écran. Cela peut avoir des implications sur la sécurité.

L'outil ne s'authentifie pas lui-même ; la tâche est déléguée au serveur LDAP. L'administrateur du serveur LDAP doit configurer et gérer les droits d'accès à différentes parties de l'annuaire. Pour plus d'informations, consultez la documentation LDAP. Si l'authentification échoue, l'outil affiche un message d'erreur approprié et s'arrête.

Des informations plus détaillées sur la sécurité et JNDI sont disponibles sur le site Web Java de Sun (https://java.sun.com).

Commandes d'administration dans l'outil d'administration JMS WebSphere MQ

L'outil d'administration accepte les commandes consistant en un verbe d'administration et ses paramètres appropriés.

Lorsque l'invite de commande s'affiche, l'outil est prêt à accepter des commandes. Les commandes d'administration se présentent généralement sous la forme suivante:

```
verb [param]*
```

où **verb** est l'un des verbes d'administration répertoriés dans Tableau 133, à la page 968. Toutes les commandes valides contiennent une instruction, qui apparaît au début de la commande sous sa forme standard ou abrégée.

Les paramètres qu'un verbe peut prendre dépendent du verbe. Par exemple, le verbe `END` ne peut pas prendre de paramètres, mais le verbe `DEFINE` peut prendre n'importe quel nombre de paramètres. Les détails des instructions qui prennent au moins un paramètre sont traités dans les rubriques connexes.

Tableau 133. Instructions d'administration

Verbe	Forme abrégée	Description
ALTER	ALT	Modifier au moins une des propriétés d'un objet géré
DEFINIR	DEF	Créer et stocker un objet géré ou créer un sous-contexte
affichage	DIS	Afficher les propriétés d'un ou de plusieurs objets gérés stockés ou le contenu du contexte en cours
SUPPRIMER	caractère d"effacement	Supprimer un ou plusieurs objets gérés de l'espace de nom ou supprimer un sous-contexte vide
MODIFIER	CHG	Modifier le contexte en cours, ce qui permet à l'utilisateur de traverser l'espace de nom du répertoire n'importe où en dessous du contexte initial (en attente d'une autorisation de sécurité)

Tableau 133. Instructions d'administration (suite)		
Verbe	**Forme abrégée**	**Description**
COPIER	programme de contrôle	Faire une copie d'un objet géré stocké, en le stockant sous un autre nom
DEPLACER	mégavolt	Modifier le nom sous lequel un objet géré est stocké
Fin		Fermer l'outil d'administration

Les noms d'instruction ne sont pas sensibles à la casse.

Généralement, pour arrêter les commandes, appuyez sur la touche de retour chariot. Toutefois, vous pouvez le remplacer en entrant le signe plus (+) directement avant le retour chariot. Cela vous permet d'entrer des commandes multilignes, comme illustré dans l'exemple suivant:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Les lignes commençant par l'un des caractères suivants sont traitées comme des commentaires et sont ignorées: * # /.

Manipulation de sous-contextes à l'aide de l'outil d'administration JMS WebSphere MQ

Utilisez les instructions **CHANGE**, **DEFINE**, **DISPLAY** et **DELETE** pour manipuler les sous-contextes d'espace de nom de répertoire.

L'utilisation de ces instructions est décrite dans Tableau 134, à la page 969.

Tableau 134. Syntaxe et description des commandes utilisées pour manipuler les sous-contextes	
Syntaxe de la commande	**Description**
DEFINE CTX (ctxName)	Tente de créer un sous-contexte enfant du contexte en cours, nommé ctxName. Échoue en cas de violation de sécurité, si le sous-contexte existe déjà ou si le nom fourni n'est pas valide.
DISPLAY CTX	Affiche le contenu du contexte en cours. Les objets gérés sont annotés avec a, les sous-contextes avec `[D]`. Le type Java de chaque objet est également affiché.
DELETE CTX (ctxName)	Tente de supprimer le contexte enfant du contexte en cours portant le nom ctxName. Échoue si le contexte est introuvable, s'il n'est pas vide ou s'il existe une violation de sécurité.
CHANGE CTX (ctxName)	Modifie le contexte en cours, de sorte qu'il fait désormais référence au contexte enfant portant le nom ctxName. L'une des deux valeurs spéciales de ctxName peut être fournie: **= ACTIF** se déplace vers le parent du contexte en cours **= INIT** se déplace directement dans le contexte initial Échoue si le contexte spécifié n'existe pas ou s'il existe une violation de sécurité.

Administration des objets JMS

Cette section décrit les huit types d'objet que l'outil d'administration peut gérer. Il inclut des détails sur chacune de leurs propriétés configurables et les verbes qui peuvent les manipuler.

Vous pouvez également créer et configurer des objets gérés JMS avec WebSphere MQ Explorer.

Types d'objet JMS

Le tableau présente les huit types d'objets gérés.

La colonne Mot clé affiche les chaînes que vous pouvez remplacer par *TYPE* dans les commandes présentées dans Tableau 136, à la page 971.

Tableau 135. Types d'objet JMS gérés par l'outil d'administration

Type d'objet	Mot clé	Description
MQConnectionFactory	CF	L'implémentation WebSphere MQ de l'interface JMS ConnectionFactory . Il s'agit d'un objet de fabrique permettant de créer des connexions dans les domaines point à point et de publication / abonnement.
Fabrique MQQueueConnection	QCF	Implémentation WebSphere MQ de l'interface JMS QueueConnectionFactory. Il s'agit d'un objet de fabrique permettant de créer des connexions dans le domaine point-à-point.
Fabrique MQTopicConnection	TCF	Implémentation WebSphere MQ de l'interface JMS TopicConnectionFactory. Il s'agit d'un objet de fabrique permettant de créer des connexions dans le domaine de publication / abonnement.
MQQUEUE	Q	Implémentation WebSphere MQ de l'interface de file d'attente JMS. Il s'agit d'une destination pour les messages du domaine point à point.
Sujet MQ	T	Implémentation WebSphere MQ de l'interface de rubrique JMS. Il s'agit d'une destination pour les messages du domaine de publication / abonnement.
MQXAConnectionFactory«1», à la page 971	XACF	L'implémentation WebSphere MQ de l'interface JMS XAConnectionFactory . Il s'agit d'un objet de fabrique permettant de créer des connexions dans les domaines de point à point et de publication / abonnement, et dans lequel les connexions utilisent les versions XA des classes JMS.
MQXAQueueConnectionFabrique«1», à la page 971	XAQCF	L'implémentation WebSphere MQ de l'interface JMS XAQueueConnectionFactory. Il s'agit d'un objet de fabrique permettant de créer des connexions dans le domaine point à point qui utilisent les versions XA des classes JMS.

Tableau 135. Types d'objet JMS gérés par l'outil d'administration (suite)		
Type d'objet	**Mot clé**	**Description**
MQXATopicConnectionMQXATopicConnection«1», à la page 971	Fonction XATCF	Implémentation WebSphere MQ de l'interface JMS XATopicConnectionFactory. Il s'agit d'un objet de fabrique permettant de créer des connexions dans le domaine de publication / abonnement qui utilisent les versions XA des classes JMS.

Remarque :

1. Ces classes sont fournies pour être utilisées par les fournisseurs de serveurs d'applications. Il est peu probable qu'ils soient directement utiles aux programmeurs d'application.

Instructions utilisées avec les objets JMS

Vous pouvez utiliser les instructions `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY`et `MOVE` pour manipuler les objets gérés dans l'espace de nom du répertoire.

Le Tableau 136, à la page 971 récapitule l'utilisation de ces verbes. Remplacez *TYPE* par le mot clé qui représente l'objet géré requis, comme indiqué dans Tableau 135, à la page 970.

Tableau 136. Syntaxe et description des commandes utilisées pour manipuler les objets gérés	
Syntaxe de la commande	**Description**
ALTER *TYPE*(nom) [propriété] *	Tente de mettre à jour les propriétés de l'objet géré avec celles fournies. Échoue en cas de violation de sécurité, si l'objet spécifié est introuvable ou si les nouvelles propriétés fournies ne sont pas valides.
DEFINE *TYPE*(nom) [propriété] *	Tente de créer un objet géré de type *TYPE* avec les propriétés fournies et le stocke sous le nom `name` dans le contexte en cours. Echec en cas de violation de sécurité, si le nom fourni n'est pas valide ou qu'un objet de ce nom existe, ou si les propriétés fournies ne sont pas valides.
DISPLAY *TYPE*(nom)	Affiche les propriétés de l'objet géré de type *TYPE*, lié sous le nom `name` dans le contexte en cours. Échoue si l'objet n'existe pas ou s'il existe une violation de sécurité.
DELETE *TYPE*(nom)	Tente de supprimer l'objet géré de type *TYPE*, portant le nom `name`, du contexte en cours. Échoue si l'objet n'existe pas ou s'il existe une violation de sécurité.
COPY *TYPE*(nameA) *TYPE*(nameB)	Effectue une copie de l'objet géré de type *TYPE*, portant le nom `nameA`, en nommant la copie `nameB`. Tout cela se produit dans la portée du contexte en cours. Echec si l'objet à copier n'existe pas, s'il existe un objet nommé nameB ou s'il existe une violation de sécurité.
MOVE *TYPE*(nameA) *TYPE*(nameB)	Déplace (renomme) l'objet géré de type *TYPE*, portant le nom `nameA`, vers `nameB`. Tout cela se produit dans la portée du contexte en cours. Échoue si l'objet à déplacer n'existe pas, s'il existe un objet nommé nameB ou s'il existe une violation de sécurité.

Création d'objets à l'aide de l'outil d'administration JMS WebSphere MQ

Créez des objets et stockez-les dans un espace de nom JNDI à l'aide de la commande DEFINE,

Utilisez la syntaxe de commande suivante:

```
DEFINE TYPE(name) [property]*
```

Il s'agit du verbe DEFINE , suivi d'une *TYPE*(name) référence d'objet géré, suivi de zéro ou plusieurs *propriétés* (voir Propriétés des objets IBM WebSphere MQ classes for JMS).

Remarques sur la dénomination LDAP pour les objets JMS
Pour stocker vos objets dans un environnement LDAP, vous devez leur attribuer des noms conformes à certaines conventions. L'outil d'administration peut vous aider à respecter les conventions de dénomination en ajoutant un préfixe par défaut.

Une convention de dénomination est que les noms d'objet et de sous-contexte doivent inclure un préfixe, tel que cn= (nom usuel) ou ou= (unité organisationnelle).

L'outil d'administration simplifie l'utilisation des fournisseurs de services LDAP en vous permettant de faire référence à des noms d'objet et de contexte sans préfixe. Si vous n'indiquez pas de préfixe, l'outil ajoute automatiquement un préfixe par défaut au nom que vous indiquez. Pour LDAP, il s'agit de cn=.

Vous pouvez modifier le préfixe par défaut en définissant la propriété NAME_PREFIX dans le fichier de configuration JMSAdmin, comme décrit dans «Utilisation d'une fabrique InitialContextnon répertoriée avec l'outil d'administration JMS WebSphere MQ», à la page 967.

Ceci est illustré dans l'exemple suivant.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

Bien que le nom d'objet fourni (testQueue) ne comporte pas de préfixe, l'outil en ajoute automatiquement un pour garantir la conformité avec la convention de dénomination LDAP. De même, la soumission de la commande DISPLAY Q(testQueue) entraîne également l'ajout de ce préfixe.

Vous devrez peut-être configurer votre serveur LDAP pour stocker des objets Java. Pour plus d'informations sur cette configuration, voir la documentation de votre serveur LDAP.

Exemple de conditions d'erreur lors de la création d'un objet JMS

Un certain nombre de conditions d'erreur communes peuvent se produire lorsque vous créez un objet.

Voici des exemples de ces conditions d'erreur:

CipherSpec mappé à CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

Propriété non valide pour l'objet

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

Type non valide pour la valeur de propriété

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

Conflit de propriétés-client/bin

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

Conflit de propriétés-Initialisation de l'exit

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

Valeur de propriété en dehors de la plage valide

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

propriété inconnue

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

Voici des exemples de conditions d'erreur pouvant survenir sous Windows lors de la recherche d'objets administrés JNDI à partir d'une application JMS.

1. Si vous utilisez le fournisseur JNDI WebSphere , com.ibm.websphere.naming.WsnInitialContextFactory, vous devez utiliser une barre oblique (/) pour accéder aux objets gérés définis dans les sous-contextes ; par exemple, jms /MyQueueName. Si vous utilisez une barre oblique inversée (\), une exception InvalidNameest émise.
2. Si vous utilisez le fournisseur JNDI Sun, com.sun.jndi.fscontext.RefFSContextFactory, vous devez utiliser une barre oblique inversée (\) pour accéder aux objets gérés définis dans les sous-contextes ; par exemple, ctx1\\fred. Si vous utilisez une barre oblique (/), une exception NameNotFoundException est émise.

Utilisation de la configuration WebSphere MQ Explorer for JMS

Utilisez l'interface graphique de IBM WebSphere MQ Explorer pour créer des objets JMS à partir d'objets WebSphere MQ et d'objets WebSphere MQ à partir d'objets JMS, ainsi que pour administrer et surveiller d'autres objets WebSphere MQ .

Avant de commencer

Avant de créer et de configurer des objets gérés JMS avec WebSphere MQ Explorer, ajoutez un contexte initial pour définir la racine de l'espace de nom JNDI dans lequel les objets JMS sont stockés dans le service d'annuaire. Pour plus d'informations, consultez l'aide aux utilisateurs de IBM WebSphere MQ Explorer pour les objets gérés par JMS.

Pourquoi et quand exécuter cette tâche

Vous pouvez effectuer les tâches suivantes à l'aide de l'explorateur IBM WebSphere MQ , soit en contexte à partir d'un objet existant dans l'explorateur IBM WebSphere MQ , soit à partir d'un assistant de création d'objet. Reportez-vous à l'aide de WebSphere MQ Explorer pour obtenir des exemples de l'assistance utilisateur WebSphere MQ Explorer pour certaines tâches standard.

Procédure

- Créez une fabrique de connexions JMS à partir de l'un des objets WebSphere MQ suivants:
  a) Un gestionnaire de files d'attente WebSphere MQ , que ce soit sur votre ordinateur local ou sur un système distant.
  b) Un canal WebSphere MQ

c) Un programme d'écoute WebSphere MQ

- Ajout d'un gestionnaire de files d'attente WebSphere MQ à WebSphere MQ Explorer à l'aide d'une fabrique de connexions JMS
- Création d'une file d'attente JMS à partir d'une file d'attente WebSphere MQ
- Création d'une file d'attente WebSphere MQ à partir d'une file d'attente JMS
- Création d'une rubrique JMS à partir d'une rubrique WebSphere MQ , qui peut être un objet WebSphere MQ ou une rubrique dynamique
- Création d'une rubrique WebSphere MQ à partir d'une rubrique JMS

Utilisation du package WebSphere MQ Headers

Le package WebSphere MQ Headers fournit un ensemble d'interfaces et de classes auxiliaires que vous pouvez utiliser pour manipuler les en-têtes WebSphere MQ d'un message. En règle générale, vous utilisez le package WebSphere 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 WebSphere MQ Headers se trouve dans les packages `com.ibm.mq.headers` et `com.ibm.mq.pcf` . Vous pouvez utiliser cette fonction pour les deux autres API fournies par WebSphere MQ dans les applications Java:

- WebSphere MQ classes for Java (également appelées WebSphere MQ Headers Base Java).
- WebSphere MQ classes for Java Message Service (WebSphere MQ classes for JMS, également appelé WebSphere MQ JMS).

Les applications Java de base WebSphere MQ manipulent généralement les objets MQMessage et les classes de support Headers peuvent interagir directement avec ces objets, car elles comprennent en mode natif les interfaces Java de base WebSphere MQ .

Dans WebSphere MQ JMS, le contenu d'un message est généralement un objet de type chaîne ou tableau d'octets, qui peut être manipulé avec des flux DataInput et DataOutput . Le package WebSphere 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 JMS WebSphere MQ .

Par conséquent, bien que le package WebSphere MQ Headers contienne des références au package WebSphere MQ Base Java, il est également destiné à être utilisé dans les applications WebSphere MQ JMS et peut être utilisé dans les environnements Java Platform, Enterprise Edition (Java EE).

Une façon typique d'utiliser le package d'en-têtes WebSphere MQ consiste à manipuler les messages d'administration au format PCF (Programmable Command Format), par exemple pour l'une des raisons suivantes:

- Pour accéder aux détails d'une ressource WebSphere MQ .
- Permet de surveiller la longueur d'une file d'attente.
- Pour empêcher l'accès à une file d'attente.

En utilisant des messages PCF avec l'API JMS WebSphere MQ , ce type d'administration des ressources centrées sur les applications peut être effectué à partir d'applications Java EE sans qu'il soit nécessaire d'utiliser l'API Java de base WebSphere MQ .

Procédure

- Pour utiliser le package WebSphere MQ Headers afin de manipuler les en-têtes de message pour WebSphere MQ classes for Java, voir «Utilisation avec WebSphere MQ classes for Java», à la page 975.
- Pour utiliser le package WebSphere MQ Headers afin de manipuler les en-têtes de message pour JMS, voir «Utilisation avec WebSphere MQ classes for JMS», à la page 975.

Utilisation avec WebSphere MQ classes for Java

Les classes WebSphere MQ pour les applications 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 classes WebSphere MQ pour les interfaces Java.

Pourquoi et quand exécuter cette tâche

WebSphere MQ fournit des exemples d'application qui expliquent comment utiliser le package WebSphere MQ Headers avec l'API Java de base WebSphere MQ (WebSphere 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 des classes WebSphere MQ pour Java.

Selon la plateforme que vous utilisez, ces exemples sont installés dans le répertoire `pcf` du répertoire `samples` ou `tools` de votre installation WebSphere MQ (voir «Répertoires d'installation pour WebSphere MQ classes for Java», à la page 681).

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 des classes WebSphere MQ pour Java.

Concepts associés

«Gestion des en-têtes de message WebSphere MQ avec WebSphere MQ classes for Java», à la page 703
Des classes Java sont fournies pour représenter différents types d'en-tête de message. Deux classes auxiliaires sont également fournies.

«Gestion des messages PCF avec WebSphere MQ classes for Java», à la page 708
Des 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 avec WebSphere MQ classes for JMS

Pour utiliser les en-têtes WebSphere MQ avec les classes WebSphere MQ pour JMS, vous devez effectuer les mêmes étapes essentielles que pour les classes WebSphere MQ pour Java. Le message PCF peut être créé et la réponse analysée de la même manière à l'aide du package WebSphere MQ Headers et du même exemple de code que pour WebSphere MQ classes for Java.

Pourquoi et quand exécuter cette tâche

Pour envoyer un message PCF à l'aide de l'API WebSphere MQ , la charge de message doit être écrite dans un message JMS Bytes 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 les informations extraites du message de réponse sont les mêmes que pour WebSphere MQ classes for Java (voir «Utilisation avec WebSphere MQ classes for Java», à la page 975).

Procédure

1. Créez une destination de file d'attente JMS qui représente SYSTEM.ADMIN.COMMAND.QUEUE.

WebSphere MQ Les applications 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 d'octets JMS 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 d'octets JMS et y écrire un objet com.ibm.mq.headers,pcf.PCFMessage . L'objet PCFMessage (pcfCmd) a déjà été généré à l'aide du package WebSphere 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 836
Les messages JMS sont composés d'un en-tête, de propriétés et d'un corps. JMS définit cinq types de corps de message.

Utilisation des services Web dans WebSphere MQ

Vous pouvez développer des applications IBM WebSphere MQ pour les services Web à l'aide du transport IBM WebSphere MQ pour SOAP ou du pont IBM WebSphere MQ pour HTTP.

Le transport IBM WebSphere MQ pour SOAP fournit un transport JMS pour SOAP. Le transport IBM WebSphere MQ pour SOAP est également intégré à d'autres environnements tels que Microsoft Windows Communication Foundation, WebSphere Application Server et CICS Transaction Server.

Pour plus d'informations sur le transport IBM WebSphere MQ pour SOAP, voir «Transport WebSphere MQ pour SOAP», à la page 978.

Avec le pont IBM WebSphere MQ pour HTTP, les applications client peuvent échanger des messages avec IBM WebSphere MQ sans qu'il soit nécessaire d'installer un client WebSphere MQ MQI. Vous pouvez appeler WebSphere MQ à partir de n'importe quelle plateforme ou langue avec des fonctions HTTP.

Pour plus d'informations sur le pont IBM WebSphere MQ pour HTTP, voir «WebSphere MQ Bridge for HTTP», à la page 1055.

Concepts associés

«Concepts de développement d'applications», à la page 8
Vous pouvez utiliser un choix de langages procéduraux ou orientés objet pour écrire des applications IBM WebSphere MQ . Utilisez les liens de cette rubrique pour obtenir des informations sur les concepts IBM WebSphere MQ utiles aux développeurs d'applications.

«Choix du langage de programmation à utiliser», à la page 81
Utilisez ces informations pour en savoir plus sur les langages de programmation et les infrastructures pris en charge par IBM WebSphere MQ, ainsi que sur les remarques relatives à leur utilisation.

«Conception d'applications IBM WebSphere MQ», à la page 93
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 WebSphere MQ.

«Exemples de programmes WebSphere MQ», à la page 100
Utilisez cette collection de rubriques pour en savoir plus sur les exemples de programmes WebSphere MQ sur différentes plateformes.

«Ecriture d'une application de mise en file d'attente», à la page 202
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 client», à la page 366
Informations à connaître pour écrire des applications client sur WebSphere MQ.

«Ecriture d'applications de publication / abonnement», à la page 290
Commencez à écrire des applications de publication / abonnement WebSphere MQ .

«Génération d'une application IBM WebSphere MQ», à la page 446
Utilisez ces informations pour en savoir plus sur la génération d'une application IBM WebSphere MQ sur différentes plateformes.

«Traitement des erreurs de programme», à la page 569
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.

Transport WebSphere MQ pour SOAP

Le transport WebSphere MQ pour SOAP fournit un transport JMS pour SOAP. Le transport WebSphere MQ pour SOAP est également intégré à d'autres environnements tels que Microsoft Windows Communication Foundation, WebSphere Application Server et CICS Transaction Server.

Introduction au transport IBM WebSphere MQ pour SOAP

Le transport IBM WebSphere MQ pour SOAP fournit un transport JMS pour SOAP. L'émetteur et le programme d'écoute SOAP WebSphere MQ permettent d'appeler des services Web.

Le programme d'écoute SOAP WebSphere MQ prend en charge les services hébergés par .NET Framework 1, .NET Framework 2 et Axis 1.4. L'émetteur SOAP WebSphere MQ prend en charge les clients de services Web exécutés sur .NET Framework 1, .NET Framework 2, Axis 1.4 et Axis2. Les clients peuvent être un serveur WebSphere MQ ou une application client. Le transport IBM WebSphere MQ pour SOAP est également intégré à d'autres environnements tels que Microsoft Windows Communication Foundation, WebSphere Application Server et CICS Transaction Server.

L'intégration à Microsoft Windows Communication Foundation fait partie de la prise en charge de IBM WebSphere MQ pour .NET Framework 3.

Le transport IBM WebSphere MQ pour SOAP est un ensemble de protocoles et d'outils permettant de transporter des messages SOAP à l'aide de JMS sur IBM WebSphere MQ. Il est conditionné de différentes manières pour différents environnements d'application, comme illustré dans la Tableau 137, à la page 978.

Tableau 137. Transport IBM WebSphere MQ pour les environnements d'application SOAP

	Intégré avec des composants WebSphere MQ supplémentaires	**Intégré dans un cadre**
Fourni dans le cadre de l'installation de WebSphere MQ	.NET Framework 1 .NET Framework 2 Axis 1.4	Windows Communication Foundation (.NET Framework 3) Axis2 (Client uniquement)
Fourni dans un autre progiciel		WebSphere Application Server CICS Transaction Server 4.1 WebSphere ESB WebSphere Process Server for Multiplatforms

L'intégration du transport IBM WebSphere MQ pour SOAP dans une infrastructure d'application simplifie le développement et le déploiement de services Web dans IBM WebSphere MQ.

Avec des composants SOAP IBM WebSphere MQ supplémentaires, vous pouvez interagir directement avec les composants SOAP WebSphere MQ pour développer et déployer des services. Utilisez les outils SOAP IBM WebSphere MQ pour configurer et déployer les services Web et les clients de service Web dans IBM WebSphere MQ.

Dans les environnements intégrés, le développement et le déploiement sont plus simples. Vous utilisez les mêmes outils pour le développement et le déploiement que pour le développement et le déploiement d'un service Web SOAP HTTP. Vous devez toujours configurer les files d'attente, les canaux et les gestionnaires de files d'attente IBM WebSphere MQ dont vous avez besoin à l'aide des outils WebSphere MQ .

Vous pouvez combiner des clients et des serveurs IBM WebSphere MQ SOAP à partir de n'importe lequel de ces environnements.

Avantages

Le transport WebSphere MQ pour SOAP offre aux utilisateurs IBM WebSphere MQ existants les principaux avantages suivants:

Utilisation de votre réseau IBM WebSphere MQ pour connecter des services Web existants.

Il peut s'agir de services que vous avez écrits ou de services qui sont fournis en tant qu'interfaces à d'autres applications logicielles packagées que vous avez déployées.

L'avantage vient de l'utilisation de votre réseau WebSphere MQ existant pour la connexion de services Web. Le transport IBM WebSphere MQ présente l'avantage d'être un service de messagerie géré et fiable en file d'attente.

Ecriture de nouvelles applications ou conversion d'applications existantes pour utiliser des interfaces SOAP plutôt que IBM WebSphere MQ .

En règle générale, les applications nécessitent un adaptateur WebSphere MQ spécifique à développer pour s'intégrer à une autre application. Les adaptateurs comportent deux parties: l'élément de connecteur, qui insère et extrait les messages vers et depuis le transport, et l'élément d'adaptateur qui convertit les données vers et depuis des formats spécifiques à l'application. L'intégration de chaque paire d'applications est un nouveau défi.

L'avantage de SOAP vient de la standardisation sur SOAP pour la définition des interfaces d'application, puis d'un choix de transports. Vous n'avez pas besoin d'écrire des adaptateurs spécifiques à l'application et vous pouvez choisir d'utiliser IBM WebSphere MQ ou HTTP comme connecteur. Le transport que vous choisissez dépend des qualités de service et de la connectivité dont vous avez besoin.

Pour les utilisateurs SOAP sur HTTP existants, l'avantage du transport WebSphere MQ pour SOAP est l'utilisation d'un transport asynchrone géré et fiable. Les avantages sont doubles:

Un modèle de programmation véritablement asynchrone pour la disponibilité et les performances.
A l'aide d'une interface client asynchrone, il n'est pas nécessaire que les applications client et de service soient disponibles en même temps. Les demandes envoyées par le client seront stockées jusqu'à ce que le service soit disponible pour les traiter.

Réseau géré prêt à l'emploi, conçu pour être fiable et disponible.

En choisissant IBM WebSphere MQ comme transport, vous bénéficiez de l'utilisation d'un réseau géré qui fournit une messagerie fiable.

En revanche, les transports tels que HTTP et FTP sur TCP/IP ne sont pas gérés. Un réseau non géré est idéal pour les connexions imprévisibles: il y a moins de tâches de gestion.

Récapitulatif

Le transport IBM WebSphere MQ pour SOAP fournit les composants suivants:

- La liaison de transport SOAP/JMS est utilisée dans les documents WSDL pour lier un service SOAP à un transport JMS. L'implémentation WebSphere MQ de la liaison SOAP/JMS utilise un URI qui prend l'une des deux formes suivantes:

  Transport WebSphere MQ pour SOAP

  ```
  jms:/queue?&Name=Value&Name=Value...
  ```

  WebSphere MQ pour la recommandation de candidat W3C

  ```
  jms:queue:qName?connectionFactory=connectQueueManager(qMgrName)&Name=Value&Name=Value...
  ```

- Mappage d'un message SOAP sur un message WebSphere MQ .
- Deux programmes d'écoute SOAP IBM WebSphere MQ pour la réception de demandes SOAP, l'un pour Java et l'autre pour .NET Framework 1 ou .NET Framework 2. Les programmes d'écoute utilisent .NET ou Axis 1.4 pour traiter la demande SOAP.
- Deux émetteurs SOAP IBM WebSphere MQ pour créer des demandes SOAP IBM WebSphere MQ . Les clients de services Web s'enregistrent auprès d'un expéditeur pour traiter les demandes SOAP `jms:` .
- Intégration à Windows Communication Foundation (WCF), parfois appelé .NET 3, pour envoyer et recevoir des messages WebSphere MQ Transport for SOAP.

- Intégration du client à Axis2, parfois appelé JAX-WS, pour envoyer des messages JMS SOAP WebSphere MQ Transport for SOAP ou W3C .
- La commande **amqwdeployWMQService**, qui crée des composants et des scripts de développement et d'exécution pour déployer un service Web à l'aide du transport IBM WebSphere MQ pour SOAP.
- Exemple de client et de code de service Java et .NET.
- Un script pour définir le chemin d'accès aux classes et d'autres scripts d'utilitaire.

Dans les environnements intégrés, l'expéditeur et le programme d'écoute sont intégrés dans chaque environnement, tout comme les extensions des outils de développement et de déploiement.

Intégration de SOAP et WebSphere MQ

Le transport WebSphere MQ pour SOAP étend SOAP, ainsi que les outils et la phase d'exécution des services Web, avec WebSphere MQ comme moyen de transport alternatif à HTTP pour SOAP. Il n'est pas nécessaire de modifier les services Web existants pour utiliser le transport WebSphere MQ pour SOAP en tant que transport. Le transport utilise un format d'URI personnalisé pour SOAP/JMS. Le format d'URI W3C pour SOAP/JMS est pris en charge de manière limitée par les clients Axis2 .

Une ligne de code supplémentaire doit être ajoutée aux clients dans les environnements .NET Framework 1, .NET Framework 2 et Axis 1.4 . Aucun code supplémentaire n'est requis dans les clients Axis 2 et Windows Communication Foundation (WCF). Le programme d'écoute SOAP WebSphere MQ exécute des services dans les environnements .NET Framework 1, .NET Framework 2 et Axis 1.4 . Le transport WebSphere MQ pour SOAP est intégré à d'autres environnements de serveur d'applications, notamment WCF, CICS et WebSphere Application Server.

Qu'est-ce que SOAP?

SOAP[9]décrit le format normalisé des messages et des protocoles d'interaction utilisés par les applications pour échanger des demandes, des réponses et des datagrammes. SOAP est indépendant du transport utilisé pour transférer les messages et de l'environnement d'application qui envoie et reçoit les messages. Le W3C définit SOAP Version 1.2 de manière succincte:

SOAP Version 1.2 fournit la définition des informations XML qui peuvent être utilisées pour échanger des informations structurées et typées entre des homologues dans un environnement décentralisé et distribué.[10].

Pour utiliser SOAP, il doit être lié à un transport, tel que HTTP, e-mail ou WebSphere MQ.

Une structure de liaison de protocole SOAP est un ensemble de règles permettant de transporter un message SOAP sur un autre protocole, tel que HTTP. SOAP Version 1.2 Partie 2: Adjuncts (Second Edition) décrit la liaison HTTP SOAP.

La W3C du 4 juin 2009, SOAP over Java Message Service 1.0, décrit la recommandation pour la liaison JMS SOAP. Comme JMS est une spécification d'API et non un protocole de transport, la recommandation JMS SOAP ne décrit pas le format WF des messages SOAP JMS. Il décrit les protocoles d'interaction SOAP et la liaison d'API JMS. Par conséquent, lorsque vous utilisez la recommandation SOAP JMS, vous devez toujours utiliser la même implémentation JMS pour le client SOAP et le serveur SOAP. Il permet l'exécution d'une application JMS SOAP sur n'importe quelle implémentation de JMS. Une implémentation JMS peut être intégrée à un serveur d'applications J2EE , si le serveur et l'implémentation JMS sont conformes à la spécification JCA. WebSphere MQ JMS est conforme à la spécification JCA et peut être connecté à un serveur d'applications compatible.

WebSphere MQ transport for SOAP binding est similaire à la norme W3C proposée, mais elle n'est pas la même. Son utilisation est décrite dans la rubrique MQRFH2 Paramètres SOAP. Contrairement à la recommandation de candidat W3C , la liaison SOAP n'est pas formellement spécifiée. En fait, il s'agit de la liaison HTTP et l'adresse de service prend la forme `jms:/queue?name=value&name=value...` plutôt que `http://authority/path?query#fragment`. `jms:` n'est pas un schéma d'URI IANA officiellement enregistré.

[9] Historiquement, l'acronyme signifie Simple Object Access Protocol.
[10] W3C: SOAP Version 1.2 Partie 0

Qu'est-ce qu'un service Web ?

SOAP permet aux programmes écrits dans des langages différents, s'exécutant sur des plateformes différentes, de communiquer à l'aide de différents protocoles de transport. SOAP est la spécification de protocole. Un service Web est une application qui fournit un service via une interface SOAP accessible à l'aide de protocoles Internet.

Un objectif important de SOAP est de fournir des services que les clients peuvent utiliser facilement. Une fois que vous avez conçu un client pour utiliser un service, vous pouvez programmer l'appel pour appeler le service sans référence à la documentation externe. Les interfaces de service sont décrites en XML, dans un document WSDL. La requête, `http://authority/path?wsdl`, renvoie la description WSDL d'un service SOAP.

Conseil : Lorsque vous déployez un service Web pour utiliser WebSphere MQ, déployez également le service sur HTTP afin que la requête WSDL standard fonctionne.

Développement de services Web

Les services Web comportent un client et un composant de service. Le service est écrit en premier, soit à partir de la description de l'interface dans WSDL, soit en suivant les règles d'écriture de la classe de service. Les kits d'outils de service Web comportent des utilitaires permettant de générer un WSDL à partir de la définition d'interface d'une classe ; par exemple, **`java2wsdl`** ou **`disco`**. Ils disposent également d'outils permettant de générer ou de classer des squelettes à partir de descriptions d'interface WSDL ; par exemple, **`wsdl2java`**, **`wsimport`**ou **`wsdl`**. Le premier est connu sous le nom de développement ascendant, et le second de haut en bas.

La commande **`amqwdeployWMQService`** dans WebSphere MQ transport for SOAP utilise ces outils pour générer des fichiers WSDL, des modules de remplacement client et des proxys client.

Les services Web sont généralement écrits à l'aide d'un environnement de développement intégré destiné à un environnement de serveur d'applications particulier:

Eclipse IDE for Java EE Developers
Crée des services Web pour Axis 2. Prend en charge JAX-RPC et JAX-WS

Rational Application Developer V7.5
Crée des services Web pour WebSphere Application Server V7 et les versions précédentes, ainsi que pour Axis. Prend en charge JAX-RPC et JAX-WS.

WebSphere Integration Developer V6.2
Crée des services Web pour WebSphere Process Server et WebSphere ESB. Prend en charge JAX-RPC et JAX-WS.

Visual Studio 2008 (version 9)
Crée des services Web pour .NET Framework 3.5 et versions antérieures (Windows Communication Foundation)

Visual Studio 2005 (version 8)
Crée des services Web pour .NET Framework 2 et les versions antérieures

Vous pouvez utiliser l'un de ces outils en combinaison avec WebSphere MQ transport for SOAP. Une fois que vous avez développé un service à utiliser avec HTTP, utilisez l'outil **`amqwdeployWMQService`** pour déployer les services afin d'utiliser WebSphere MQ en tant que transport. Vous pouvez écrire un nouveau client à l'aide de la sortie de l'outil ou modifier vos clients existants pour utiliser le transport WebSphere MQ pour SOAP.

Si WebSphere MQ transport for SOAP est intégré à l'environnement d'application, vous n'avez pas besoin d'utiliser l'outil **`amqwdeployWMQService`** ni de modifier le code client. La couche SOAP du client dirige les demandes du client dont l'URI est précédé du préfixe `jms:` vers le transport WebSphere MQ pour SOAP. La couche SOAP du serveur appelle WebSphere MQ transport for SOAP to wait for `jms:` SOAP requests et renvoie des réponses à WebSphere MQ transport for SOAP.

Généralement, les services .NET ont été développés de bas en haut à l'aide d'annotations de service Web dans le code, et les services Java de haut en bas à l'aide de définitions d'interface WSDL. La différence dans les approches est limitée, car Java Standard Edition version 6 prend en charge JAX-WS 2.0et utilise

des annotations pour qualifier la définition des interfaces de service. Il est désormais aussi facile de développer des services Java de bas en haut que de haut en bas. L'approche que vous choisissez est une question de méthode de développement.

Le client de services Web est écrit après le service, à l'aide de la définition de service WSDL et des modules de remplacement et des proxys de client générés. Dans certaines applications, la définition de service n'est pas connue lorsque le client est écrit. Le client extrait le WSDL du service et crée des demandes de service de manière dynamique. Plus généralement, la définition de service est connue, mais l'adresse à laquelle le service est déployé ne l'est pas. Le kit d'outils de services Web génère des interfaces que le client peut utiliser pour effectuer des demandes de service. Le client fournit l'adresse de service lorsqu'elle est requise. Dans le troisième cas, le WSDL contient toutes les informations dont un client a besoin. Le WSDL contient à la fois l'interface et l'adresse du service. Le code généré par le kit d'outils de service Web contient toutes les informations requises par le client pour effectuer des demandes de service.

Vous pouvez utiliser l'un de ces trois styles avec WebSphere MQ transport for SOAP.

Environnements d'application de service Web

Les kits d'outils de service Web nécessitent un mappage de la définition WSDL d'un service vers les flux d'octets qui sont transférés dans les demandes et les réponses SOAP. Le flux d'octets est défini par la spécification SOAP et est contenu dans l'enveloppe SOAP. L'enveloppe SOAP est présentée dans la Figure 166, à la page 982.

```
<?xml version='1.0'?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header> <!-- optional -->
<!-- headers... -->
</soap:Header>
<soap:Body>
<!-- payload or fault message -->
</soap:Body>
</soap:Envelope>
```

Figure 166. enveloppe SOAP

Le mappage de l'enveloppe SOAP à la liaison de langage et au retour est une partie standardisée et propriétaire de la partie. Le mappage est fondamental pour l'architecture .NET et est fourni dans le cadre de Common Language Runtime (CLR). Le mappage est normalisé dans Java par les spécifications JAX. Etant donné que les mappages Java sont normalisés, les clients et les services de service Web Java sont portables entre différents environnements d'application Java. JAX-RPC (parfois appelé JAX-WS 1.0) est le mappage le plus utilisé aujourd'hui. Il est pris en charge par l'axe 1.4. JAX-WS (parfois appelé JAX-WS 2.0) est une norme considérablement améliorée et est susceptible de remplacer JAX-RPC rapidement. JAX-WS est pris en charge par Axis 2.0. WebSphere MQ 7.0.1 ne prend pas en charge JAX-WS et Axis 2.

WebSphere MQ transport for SOAP ne modifie pas le contenu de l'enveloppe SOAP et le contenu n'affecte pas le transport. Les liaisons de langage affectent le transport WebSphere MQ pour SOAP. WebSphere MQ 7.0.1 prend en charge .NET Framework 1, .NET Framework 2 et Axis 1.4 à l'aide du code et des utilitaires fournis avec WebSphere MQ transport for SOAP. La prise en charge du transport WebSphere pour SOAP dans .NET Framework 3 et 3.5 est implémentée à l'aide du canal personnalisé WebSphere MQ pour Windows Communication Foundation.

D'autres environnements de développement et d'exécution SOAP peuvent fournir la prise en charge du transport WebSphere MQ pour SOAP et prendre en charge différents langages. Par exemple, les services Web exécutés sous CICS prennent en charge des langages tels que COBOL et PL/1.

Remarque : Le mappage utilisé ne change rien à l'interopérabilité des services Web. Vous pouvez combiner des clients et des services écrits à l'aide de mappages .NET, JAX-RPC et JAX-WS.

Qu'est-ce que le transport WebSphere MQ pour SOAP?

WebSphere MQ transport for SOAP est une liaison SOAP et un kit d'outils de services Web. Ensemble, ils permettent aux applications d'échanger des messages SOAP à l'aide de WebSphere MQ plutôt que HTTP. Figure 167, à la page 983 Affiche WebSphere MQ comme alternative à HTTP en tant que transport SOAP.

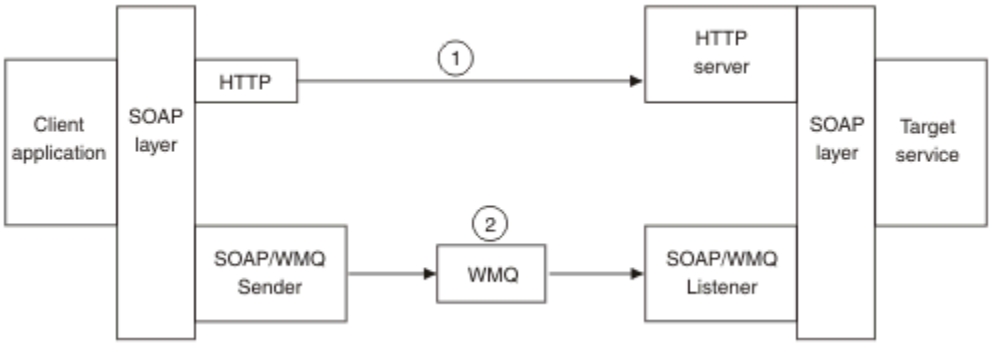


Figure 167. Présentation du transport WebSphere MQ pour SOAP

SOAP sur HTTP est affiché sous la forme (1) dans le diagramme. La couche SOAP du client convertit une demande en message SOAP et le composant HTTP est envoyé via TCP/IP. Le composant serveur HTTP écoute les demandes HTTP, généralement sur le port TCP/IP 80. Si la demande concerne un service SOAP, le composant serveur HTTP appelle la couche SOAP pour convertir la demande SOAP en appel de méthode. Il renvoie ensuite la réponse.

SOAP sur WebSphere MQ s'affiche sous la forme (2). L'application client enregistre le composant émetteur SOAP WebSphere MQ en tant que gestionnaire pour le protocole `jms:` avec la couche SOAP. La couche SOAP transmet les messages SOAP adressés à `jms:` à l'expéditeur SOAP WebSphere MQ . L'expéditeur utilise l'URI du message pour placer le message dans la file d'attente des demandes avec les qualités de service requises. Le programme d'écoute SOAP WebSphere MQ correspondant attend les messages dans sa file d'attente des demandes et appelle la couche SOAP pour traiter les demandes et renvoyer les réponses.

L'émetteur et le programme d'écoute SOAP sont des programmes WebSphere MQ normaux. Ils peuvent être connectés au même gestionnaire de files d'attente, comme dans Figure 168, à la page 984, ou à des gestionnaires de files d'attente différents ; voir Figure 169, à la page 985. Le client peut être connecté par une connexion client.

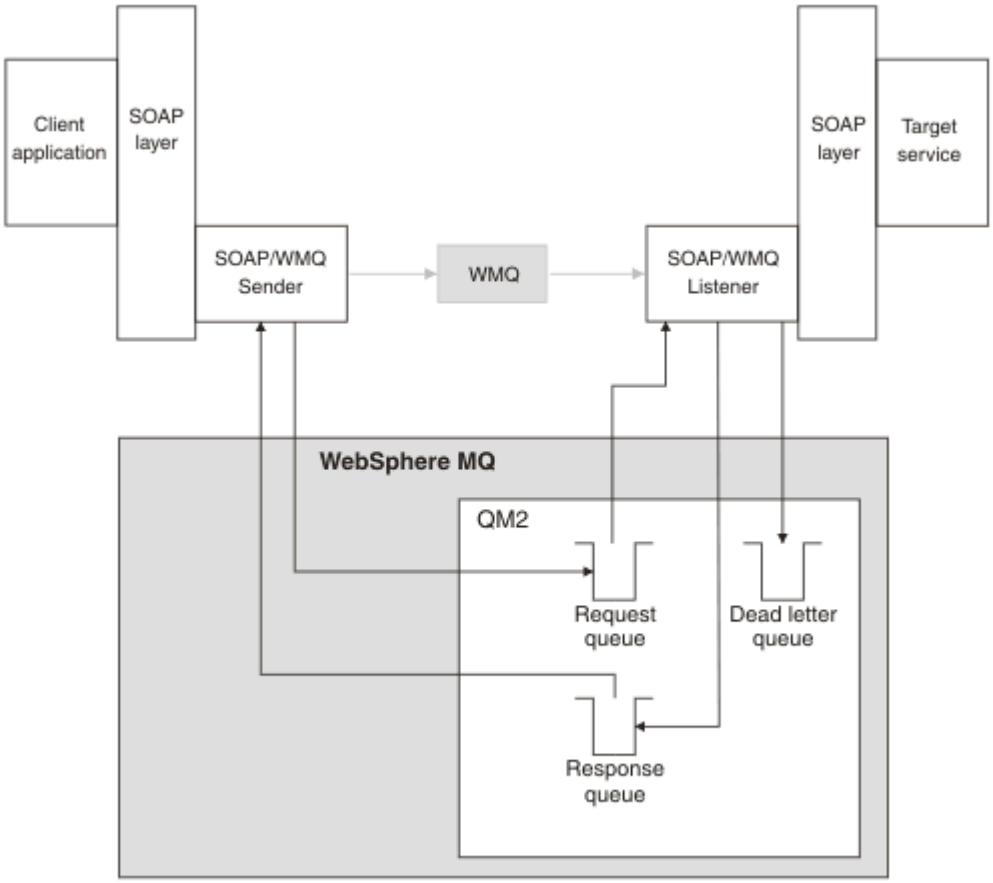


Figure 168. Files d'attente utilisées par SOAP/WebSphere MQ (gestionnaire de files d'attente unique)

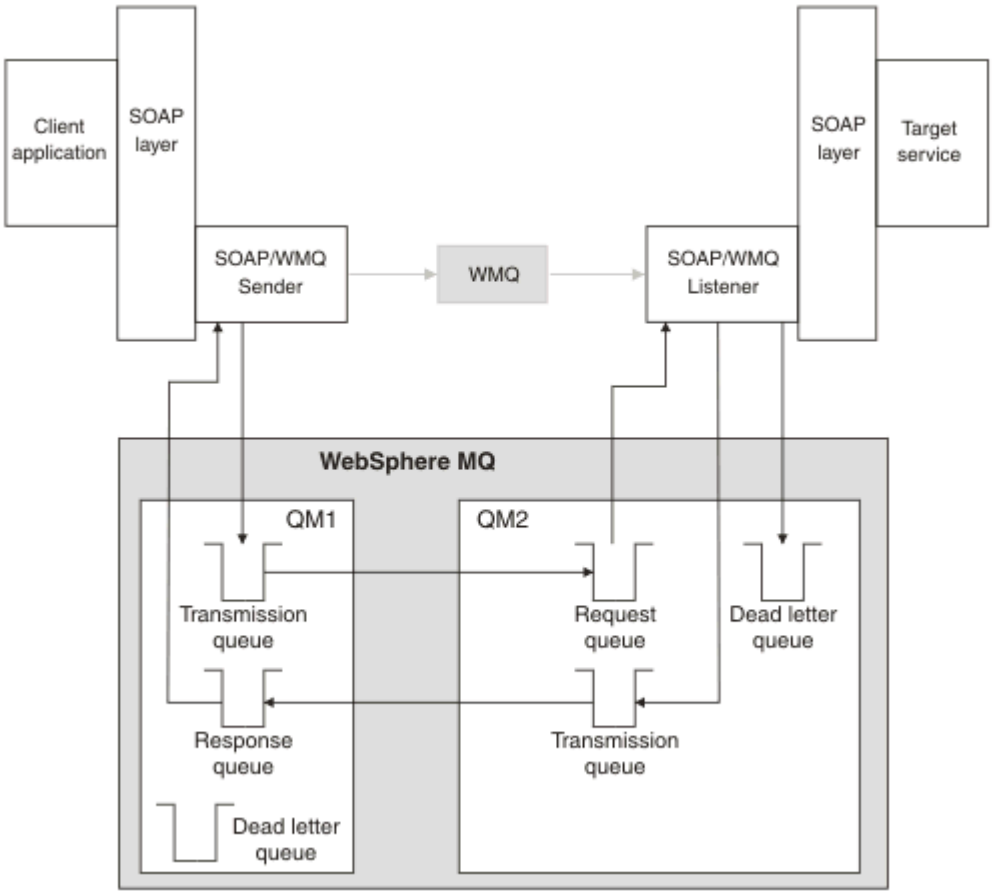


Figure 169. Files d'attente utilisées par SOAP/WebSphere MQ (gestionnaires de files d'attente distincts)

Recommandation de candidat W3C pour la liaison de SOAP à JMS.

La recommandation candidate W3C définit la liaison SOAP sur JMS, SOAP over Java Message Service 1.0. Il est également utile pour ses exemples : Schéma d'URI pour Java (tm) Message Service 1.0[11].

Certaines infrastructures d'application, telles que WebSphere Application Server v7, prennent en charge la recommandation de candidat W3C . Envoyez des demandes SOAP formatées avec un URI compatible avec la recommandation candidate W3C à l'aide du client Axis2 ; voir W3C SOAP sur JMS URI pour le WebSphere MQ Axis 2 . Le client Axis2 envoie une demande SOAP formatée avec un W3C ou un transport WebSphere MQ pour SOAP en fonction de l'URI de la demande SOAP.

La prise en charge du client Axis2 pour la recommandation W3C est introduite dans le groupe de correctifs 7.0.1.3 . La prise en charge des autres clients et des programmes d'écoute SOAP fournis par WebSphere MQ n'est pas fournie.

Concepts associés

Implémentation du transport WebSphere pour SOAP sur .NET Framework 1, .NET 2 et Axis 1.4
Vous pouvez écrire votre propre émetteur et programme d'écoute SOAP WebSphere MQ . Utilisez l'implémentation de WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2 et Axis 1.4 comme guide.

WebSphere Transport MQ pour la messagerie fiable des services SOAP et Web
La messagerie fiable des services Web est un protocole permettant d'échanger de manière fiable des demandes et des réponses de services Web via une connexion non fiable. Il est le mieux adapté pour résoudre les problèmes d'interruption de connexion de courte durée.

11 Recherchez *Schéma d'URI pour JMS* dans les références de spécification W3C pour le dernier brouillon.

Implémentation du transport WebSphere pour SOAP sur .NET Framework 1, .NET 2 et Axis 1.4

Vous pouvez écrire votre propre émetteur et programme d'écoute SOAP WebSphere MQ . Utilisez l'implémentation de WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2 et Axis 1.4 comme guide.

1. Un programme client utilise la structure de services Web appropriée de la même manière que pour le transport HTTP. Il doit également enregistrer le préfixe `jms:` . Le préfixe est enregistré à l'aide de la méthode Java `com.ibm.mq.soap.Register.extension()` ou de la méthode CLR `IBM.WMQSOAP.Register.Extension()` .
2. Axis 1.4 ou .NET Framework 1 ou 2 permet de convertir l'appel en message de demande SOAP exactement comme pour SOAP/HTTP.
3. Un service WebSphere MQ est identifié par un URI préfixé avec `jms:`. Lorsque l'infrastructure identifie l'URI `jms:` , elle appelle le code émetteur du transport WebSphere MQ ; `com.ibm.mq.soap.transport.jms.WMQSender` (pour Axis 1.4) ou `IBM.WMQSOAP.MQWebRequest` (pour .NET1 et 2). Si l'infrastructure rencontre un URI avec un préfixe `http:` , elle appelle l'expéditeur SOAP sur HTTP standard.
4. Le message SOAP est transporté par l'émetteur SOAP WebSphere MQ à l'aide de la file d'attente des demandes. **`SimpleJavaListener`** (pour Java) ou **`amqwSOAPNETListener`** (pour .NET) reçoit le message de demande.

   Les programmes d'écoute SOAP WebSphere MQ sont des processus autonomes à unités d'exécution multiples avec un nombre d'unités d'exécution personnalisable.
5. Le programme d'écoute SOAP WebSphere MQ lit la demande SOAP entrante et la transmet à l'infrastructure de service Web appropriée.
6. L'infrastructure de service Web analyse le message de demande SOAP et appelle le service. La procédure est la même que pour un message arrivé sur un transport HTTP.
7. L'infrastructure formate la réponse dans un message de réponse SOAP et la renvoie au programme d'écoute SOAP WebSphere MQ .
8. Le programme d'écoute place le message dans la file d'attente de réponses et le message est transféré à l'émetteur SOAP WebSphere MQ . L'expéditeur le transmet à l'infrastructure de service Web du client.
9. L'infrastructure client analyse le message SOAP de la réponse et transmet le résultat à l'application client.

Chaque contexte d'application est servi par une file d'attente de demandes WebSphere MQ distincte.

Le contexte d'application est contrôlé dans l'axe 1.4 en s'assurant que le programme d'écoute SOAP WebSphere MQ et le service s'exécutent dans le répertoire approprié. Axis 1.4 définit la variable CLASSPATH correcte pour le répertoire.

Le contexte d'application est contrôlé dans .NET par le programme d'écoute SOAP WebSphere MQ qui exécute le service dans un contexte créé par un appel à `ApplicationHost.CreateApplicationHost`. L'appel spécifie le répertoire d'exécution cible. Chaque service fonctionne ensuite dans le répertoire dans lequel il a été déployé.

`amqwdeployWMQService` génère les files d'attente de demandes et de réponses. Il génère également l'infrastructure nécessaire à la gestion des files d'attente et au déploiement des services sur Axis 1.4.

Concepts associés

Intégration de SOAP et WebSphere MQ

WebSphere Transport MQ pour la messagerie fiable des services SOAP et Web

La messagerie fiable des services Web est un protocole permettant d'échanger de manière fiable des demandes et des réponses de services Web via une connexion non fiable. Il est le mieux adapté pour résoudre les problèmes d'interruption de connexion de courte durée.

WebSphere Transport MQ pour la messagerie fiable des services SOAP et Web

La messagerie fiable des services Web est un protocole permettant d'échanger de manière fiable des demandes et des réponses de services Web via une connexion non fiable. Il est le mieux adapté pour résoudre les problèmes d'interruption de connexion de courte durée.

WebSphere MQ for SOAP tire parti de l'utilisation d'un réseau géré et fiable WebSphere MQ pour la transmission de messages SOAP. Les transports tels que HTTP et FTP ne sont pas gérés. Les réseaux non gérés sont idéaux pour les connexions imprévisibles, où les difficultés et les coûts de la gestion des connexions l'emportent sur les avantages de ne pas perdre les demandes et les réponses.

Pour surmonter le problème de perte de fichiers lorsque les connexions se rompent dans des réseaux non gérés, des services tels que le FTP géré créent une couche de gestion sur le FTP. La couche de gestion prend la charge de vérifier que les fichiers ont été transférés avec succès des utilisateurs, et retransmet les fichiers manquants si nécessaire. Pour utiliser le FTP géré, le logiciel de gestion doit être installé aux deux extrémités de la connexion.

La messagerie fiable des services Web (WSRM) adopte une approche différente pour résoudre le problème des connexions non fiables. Son objectif est de transférer des demandes et des réponses de service Web de manière fiable, sans que les deux extrémités de la connexion aient à utiliser le même logiciel. Tout logiciel, en implémentant le protocole de messagerie fiable des services Web, peut échanger des messages de manière fiable avec d'autres logiciels.

En cas d'échec d'une connexion, un expéditeur et un récepteur doivent conserver le contexte du transfert de message WSRM, en utilisant un URI généré comme clé. L'expéditeur et le destinataire continuent de tenter d'établir une nouvelle connexion. Si une nouvelle connexion est établie avec succès, le transfert se termine. La spécification WSRM ne spécifie pas comment le contexte est conservé, ni quand une nouvelle connexion est tentée.

Vous pouvez décider que seules les indisponibilités de courte durée présentent un intérêt. Pour les indisponibilités plus longues, vous pouvez être prêt à supprimer les transferts qui n'ont pas pu être redémarrés après un certain temps. De même, vous pouvez être prêt à supprimer les transferts en cas d'échec du client ou du service. Laisser à l'utilisateur la responsabilité d'assurer les transferts, impose moins de contraintes à la gestion de la coordination du client et du service.

Si les indisponibilités du réseau sont de longue durée, de plus de 30 minutes environ, ou si le client ou le serveur est défaillant, il est plus probable que certaines connexions ne soient jamais rétablies. Vous ne pouvez plus compter sur la restauration automatique du transfert de message par WSRM de manière non gérée. Vous devez envisager de gérer les connexions WSRM ayant échoué, ce qui implique le développement de logiciels pour gérer le réseau de clients et de services.

L'utilisation de WSRM pour surmonter des indisponibilités de courte durée peut réduire considérablement le traitement des messages perdus sur un réseau mobile. Si vous n'avez pas besoin d'assurer la distribution des messages, les avantages de la réduction de la perte de messages peuvent justifier le coût supplémentaire du développement d'une implémentation WSRM.

SOAP sur JMS assure la distribution des messages et gère les indisponibilités de longue durée du client, du serveur et du réseau. Si vous recherchez une qualité de service plus fiable pour SOAP que HTTP, quelle solution choisissez-vous: WebSphere MQ transport pour SOAP ou WSRM? La réponse dépend de nombreux facteurs. Voici quelques-uns des facteurs à prendre en considération:

1. Indique si la non-fiabilité est due à un échec de connexion.
2. Combien de temps les échecs de connexion aimés sont.
3. Si vous pouvez gérer à la fois le côté client et le côté serveur de la connexion.
4. Si l'utilisateur ou un administrateur est en dernier ressort responsable de la distribution des messages.

Concepts associés

Intégration de SOAP et WebSphere MQ

Implémentation du transport WebSphere pour SOAP sur .NET Framework 1, .NET 2 et Axis 1.4

Vous pouvez écrire votre propre émetteur et programme d'écoute SOAP WebSphere MQ . Utilisez l'implémentation de WebSphere MQ transport for SOAP on .NET Framework 1, .NET Framework 2 et Axis 1.4 comme guide.

Installation et vérification des services Web WebSphere MQ

Utilisez les instructions de ces rubriques pour installer et vérifier le transport WebSphere MQ pour SOAP.

Installation de WebSphere MQ Web transport for SOAP

Utilisez ces instructions pour installer le transport Web WebSphere MQ pour SOAP. L'installation crée des outils pour exécuter des clients de service Web ou des services en utilisant WebSphere MQ comme transport SOAP. Les outils sont utilisés dans les environnements SOAP .NET Framework 1, .NET 2, Axis 1.4ou Axis2 .

Avant de commencer

Vérifiez les produits prérequis à l'adresse Configuration système requise pour IBM WebSphere MQ. Le processus d'installation ne vérifie pas la présence et la disponibilité des logiciels prérequis. Vous devez vérifier que les prérequis sont installés.

WebSphere MQ fournit une copie de l'environnement d'exécution Axis 1.4 . Utilisez cette version avec WebSphere MQ plutôt qu'avec toute autre version que vous avez peut-être installée. IBM ne fournit pas de support technique pour Apache Axis. Contactez Apache Software Foundation si vous rencontrez des problèmes techniques.

Pour exécuter des services Web dans l'environnement SOAP .NET Framework 3, WebSphere MQ utilise Windows Communication Foundation. Le canal personnalisé WebSphere MQ pour Windows Communication Foundation exécute des clients de service Web et des services en utilisant WebSphere MQ comme transport pour les messages SOAP.

Pourquoi et quand exécuter cette tâche

Vous pouvez installer WebSphere MQ Web transport for SOAP en tant que client WebSphere MQ MQI ou application serveur. Si vous avez déjà installé WebSphere MQ en tant que client ou serveur sur votre ordinateur, vérifiez que vous avez installé les composants répertoriés.

MQ_INSTALLATION_PATH représente le répertoire de haut niveau dans lequel WebSphere MQ est installé.

Effectuez les étapes d'installation suivantes.

Procédure

1. Sélectionnez le composant "Java and. Net Messaging and Web services" pour l'installation.
2. Sous Solaris et HP-UX, sélectionnez le composant "Environnement d'exécution Java" pour l'installation.
3. Sélectionnez le kit d'outils de développement pour l'installation.
4. Installez et vérifiez WebSphere MQ comme décrit dans le guide de démarrage rapide de votre plateforme.
5. Copiez l'environnement d'exécution Apache Axis 1.4 , `axis.jar` depuis le répertoire `prereqs/axis` sur le support d'installation WebSphere MQ . Copiez-le dans le répertoire d'installation décrit dans Tableau 138, à la page 989, Tableau 139, à la page 989 ou Tableau 140, à la page 989.

   Fenêtres

   ```
   Copy D:\PreReqs\axis\axis.jar MQ_INSTALLATION_PATH\java\lib\soap
   ```

   AIX

   ```
   cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
   ```

Répertoires d'installation de HP-UX, Solaris et Linux (toutes les plateformes)

```
cp -f PreReqs/axis/axis.jar MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chown mqm:mqm MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
chmod 444 MQ_INSTALLATION_PATH/java/lib/soap/axis.jar
```

6. Sous Windows 2003, exécutez **Aspnet_regiis.exe** pour mettre à jour les mappes de script afin de pointer vers la version de l'environnement d'exécution du langage commun que vous utilisez.

   Recherchez l'utilitaire **Aspnet_regiis.exe** dans %SystemRoot%Microsoft.NET/Framework/*version-number* .
7. Définissez la variable d'environnement WMQSOAP_HOME pour qu'elle pointe vers le répertoire d'installation WebSphere MQ .

Résultats

Tableau 138. Répertoires d'installation de Windows

Emplacement	Contenu
MQ_INSTALLATION_PATH\programs\bin	Fichiers binaires, de commande, DLL et exécutables
MQ_INSTALLATION_PATH\programs\java\lib	.jar files
MQ_INSTALLATION_PATH\programs\java\lib\soap	SOAP .jar files
MQ_INSTALLATION_PATH\programs\soap\samples	Exemples et IVT

Tableau 139. AIX -Répertoires d'installation

Emplacement	Contenu
MQ_INSTALLATION_PATH/bin	Scripts shell
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar et autres fichiers .jar JAX-RPC
MQ_INSTALLATION_PATH/samp/soap	Exemples et IVT

Tableau 140. Répertoires d'installation de HP-UX, Solaris et Linux (toutes les plateformes)

Emplacement	Contenu
MQ_INSTALLATION_PATH/bin	Scripts shell
MQ_INSTALLATION_PATH/java/lib	.jar files
MQ_INSTALLATION_PATH/java/lib/soap	axis.jar et autres fichiers .jar JAX-RPC
MQ_INSTALLATION_PATH/samp/soap	Exemples et IVT

Que faire ensuite

1. Pour .NET uniquement, vous devez enregistrer le transport WebSphere MQ pour les fichiers SOAP avec le cache d'assemblage global. Si .NET est déjà installé lorsque vous installez WebSphere MQ, l'enregistrement est effectué automatiquement lors de l'installation. Si vous installez .NET après WebSphere MQ, l'enregistrement est effectué automatiquement lors de la première exécution de l'IVT.

   Vous pouvez exécuter **amqiregisterdotnet.cmd** pour effectuer l'enregistrement des assemblages .NET. Vous pouvez également exécuter **amqiregisterdotnet.cmd** pour forcer le

réenregistrement à n'importe quelle étape. Une fois effectué, cet enregistrement survit aux redémarrages du système et le réenregistrement ultérieur n'est normalement pas nécessaire.

2. Exécutez le test de vérification de l'installation, comme décrit dans «Vérification du transport IBM WebSphere MQ pour SOAP», à la page 990.
3. Si vous avez l'intention de développer le client Axis2 , vous devez télécharger Axis2 1.4.1 à partir d' Apache; voir «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013.

Vérification du transport IBM WebSphere MQ pour SOAP

Vérifiez le transport IBM WebSphere MQ pour SOAP à l'aide de la commande **runivt** . La commande exécute un certain nombre d'applications de démonstration et garantit que l'environnement est correctement configuré après l'installation.

Avant de commencer

Avant d'exécuter la commande **runivt** , vérifiez que vous disposez des environnements d'exécution suivants:

- Pour une exécution sur Axis uniquement: vous devez disposer d'un SDK Java (dans SOE) disponible sur votre système. Vous devez également inclure l'emplacement des commandes `java.exe` et `javac.exe` dans la variable d'environnement **PATH** des systèmes.
- Pour exécuter un test sur .NET uniquement (pris en charge uniquement sur Windows): vous devez disposer d'un SDK Java et des compilateurs et outils .NET sur votre système. Pour ce faire, accédez à une invite de commande Visual Studio ou à l'invite de commande Microsoft Windows SDK, puis ajoutez l'emplacement des fichiers `java.exe` et `javac.exe` à la variable d'environnement **PATH** .
- Pour exécuter tous les tests disponibles: pour les plateformes Windows , l'environnement doit être configuré comme décrit dans l'exécution de test .NET. Sur les plateformes UNIX and Linux , l'environnement doit être configuré comme décrit dans l'exécution de test Axis uniquement.

Pourquoi et quand exécuter cette tâche

Au lieu d'exécuter le test de vérification sur .NET et Axis, vous pouvez exécuter le test uniquement sur Axis ou uniquement sur .NET.

Si vous rencontrez des problèmes avec les tests et que vous souhaitez recommencer:

1. Arrêtez le gestionnaire de files d'attente `WMQSOAP.DEMO.QM` à l'aide de l'option `immediate` .
2. Arrêtez le programme d'écoute qui a été démarré dans une autre fenêtre.
3. Supprimez le gestionnaire de files d'attente.
4. Supprimez le répertoire des exemples temporaires que vous avez créé et recommencez.

Sur les plateformes UNIX and Linux , vous devez exécuter la commande à l'aide d'une session système X Windows .

La commande **runivt** modifie le contenu du répertoire `soap/samples` . Pour conserver l'image d'installation inchangée, copiez le répertoire des exemples dans un emplacement temporaire et exécutez le test de vérification à partir de l'emplacement temporaire.

Vous pouvez exécuter la vérification de l'installation autant de fois que vous le souhaitez.

Procédez comme suit pour vérifier l'installation du transport IBM WebSphere MQ pour SOAP sur .NET Framework 1, .NET Framework 2 et Axis 1.4:

Procédure

1. Copiez l'arborescence de répertoires `./tools/soap/samples` dans un emplacement temporaire.
2. Ouvrez une fenêtre de commande avec le répertoire temporaire comme répertoire de travail.
3. Utilisez la commande **runivt** pour démarrer le test d'installation. Le script `runivt` compile une classe de test, l'exemple de client et les services avant de les déployer et de les exécuter. Pour

la classe de test, l'exemple de client et les services à exécuter, effectuez les étapes d'installation décrites dans Installation de WebSphere(r) MQ Web transport for SOAP et vérifiez que l'invite de commande utilisée pour exécuter la commande `runivt` possède l'environnement d'exécution requis défini. Utilisez l'une des méthodes suivantes pour exécuter la commande **runivt** :

- Exécutez un test sur l'axe uniquement: `runivt Axis`.
- Exécutez un test sur .NET uniquement (pris en charge uniquement sous Windows): `runivt DotNet`.
- Exécutez tous les tests disponibles:`runivt`.

Pour plus d'informations sur la syntaxe et les paramètres de la commande `runivt` , voir **runivt**: IBM WebSphere MQ transport for SOAP installation verification test. Les tests que vous pouvez exécuter sont répertoriés dans le fichier `ivttests.txt` sous Windows et `ivttests_unix.txt` sur les plateformes UNIX and Linux .

Référence associée

runivt: WebSphere MQ pour le test de vérification de l'installation SOAP

Développement de services Web pour WebSphere MQ transport for SOAP

Utilisez votre environnement de développement de services Web normal pour développer des services à utiliser avec le transport WebSphere MQ pour SOAP.

Avant de commencer

1. Si vous prévoyez d'utiliser les outils de ligne de commande fournis avec WebSphere MQ transport for SOAP:
   a. Créez un répertoire de déploiement pour le service.
   b. Ouvrez une fenêtre de commande dans le répertoire.
   c. Pour .NET, csc.exe et wsdl.exe doivent se trouver dans le chemin et provenir de la même version de .NET Framework.
   d. Pour Java,
      i) Exécutez la commande **amqwsetcp** pour configurer le chemin d'accès aux classes.
      ii) Un environnement d'exécution Java (JRE) IBM et un kit JDK au même niveau de version doivent se trouver dans le chemin en cours. Le niveau de version doit être au moins 5.0.
      iii) Personnalisez le chemin d'accès aux classes pour inclure les emplacements des bibliothèques `.jar` supplémentaires et des répertoires contenant les packages .java, y compris pour le service que vous développez. Placez le répertoire en cours "`.`" dans le chemin d'accès aux classes.
      iv) Créez un répertoire, relatif au répertoire en cours de la fenêtre de commande, correspondant au nom de package du service que vous développez.
2. Vous pouvez également utiliser les outils du plan de travail qui prennent en charge le développement de services Web. Les exemples de tâches de développement utilisent Microsoft Visual Studio 2008, Eclipse IDE for Java EE Developers et WebSphere Application Server Community Edition.

Pourquoi et quand exécuter cette tâche

Les services Web existants n'ont pas besoin d'être modifiés pour fonctionner avec le transport WebSphere pour SOAP. Les outils fournis avec WebSphere MQ transport for SOAP permettent de déployer un service Web et de l'exécuter à l'aide d'un programme d'écoute SOAP WebSphere MQ . Les outils génèrent également des fichiers WSDL, des modules de remplacement de client .NET et des classes de proxy `.java` pour développer le transport WebSphere MQ pour les clients SOAP.

Procédez comme suit pour créer un service et le préparer pour le déploiement et la génération de clients. Suivez les étapes des tâches connexes pour créer un service à l'aide d' Eclipse ou de Microsoft Visual Studio 2008.

Procédure

1. Développez le service à l'aide de votre environnement de développement normal.
2. Tester le service à l'aide d'un client de services Web HTTP
3. Pour préparer le répertoire de déploiement, procédez comme suit:
   - Pour Java
     a. Copiez le fichier `.java` définissant l'interface de service dans le répertoire de déploiement.
     b. Copiez les fichiers `.class` du service dans le répertoire correspondant au nom du package.
     c. Vérifiez que le chemin d'accès aux classes peut localiser toutes les classes requises: compilez le fichier `.java` du service à l'aide de **javac**.
   - pour .NET
     a. Copiez le fichier `.asmx` définissant le service dans le répertoire de déploiement, et
     b. Si vous utilisez le modèle code-behind, copiez tous les fichiers `.dll` dans un répertoire *deployment directory*`\bin` .

Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse

Développez un service Web Axis 1.4 à exécuter en utilisant WebSphere MQ comme fournisseur de services. Utilisez votre environnement de développement de service Web normal pour créer un service à déployer sur Axis 1.4.

Avant de commencer

Prenez en compte les conditions requises pour le déploiement d'un serveur Web sur le WebSphere MQ programme d'écoute SOAP pour Axis 1.4.

- Le programme d'écoute SOAP WebSphere MQ pour Axis 1.4 requiert un environnement d'exécution Java IBM version 5.0 ou ultérieure. Le JRE et le JDK utilisés pour le développement doivent être au même niveau de version.
- Le programme d'écoute SOAP WebSphere MQ pour Axis 1.4 requiert que `axis.jar` soit installé avec WebSphere MQ. Modifiez le chemin de génération dans votre environnement de développement pour référencer le fichier `axis.jar` installé avec WebSphere MQ, plutôt que les fichiers `axis.jar` installés avec l'environnement de développement.
- Jusqu'à WebSphere MQ V7.0.1inclus, le WSDL généré pour le service déployé est RPC/encodé. A partir de V7.1, vous pouvez également demander un WSDL de style RPC/littéral. Le WSDL généré est utilisé uniquement pour le déploiement. Vous pouvez définir le service à l'aide d'un WSDL compatible WS-I.

Pourquoi et quand exécuter cette tâche

Créez le service à l'aide de votre environnement de développement de services Web normal.

Dans cette tâche, nous utilisons l'environnement de développement intégré libre Eclipse Java EE for Web Developers, connu sous le nom de Galileo. Pour le serveur d'applications, nous utilisons WebSphere Application Server Community Edition v2.1 (Community Edition), basé sur Geronimo. Pour plus d'informations sur l'obtention, l'installation et la configuration de l'IDE et du serveur, voir les tâches associées.

Testez le service en utilisant HTTP comme transport à l'aide de l'explorateur de services Web fourni dans l'IDE précédemment. Vous pouvez également générer un proxy client HTTP et tester le service à l'aide de votre propre code client.

Vous pouvez suivre ces étapes pour développer un service Web ascendant. Utilisez l'exemple de programme `StockQuoteAxis.java` comme exemple.

Procédure

1. Démarrez Eclipse IDE for Java EE Developers avec un nouvel espace de travail.
2. Configurez l'espace de travail pour utiliser Java50,
   WebSphere Application Server Community Edition 2.1.4 ne fonctionne pas avec Java60.
   a) **Fenêtre** > **Préférences** > **Java** > **JRE installés** > **Ajouter ...** > **VM standard** > **Suivant** > **Répertoire ...**
   b) Accédez au répertoire d'installation de **Java50** > **OK** > **Terminer**
   c) Vérifiez l'environnement d'exécution Java **Java50** > OK
3. Ajoutez l'environnement d'exécution Community Edition et démarrez Community Edition.
   a) **Fenêtre** > **Préférences** > **Serveur** > **Environnements d'exécution** > **Ajouter ...**
   b) Sélectionnez **IBM WASCE v2.1** dans la liste **New Server Runtime Environment** > Vérifier **Créer un serveur local** > **Suivant**
      Si **IBM WASCE 2.1** ne figure pas dans la liste, vous devez effectuer deux autres tâches:
      i) Installez WebSphere Application Server Community Edition.
      ii) Installez la mise à jour d' Eclipse pour Community Edition.
      Recherchez les détails à l'adresse WebSphere Application Server Community Edition
   c) Accédez au répertoire d'installation du serveur d'applications > **OK** > **Terminer** > **OK**.
   d) Cliquez avec le bouton droit de la souris sur **IBM WASCE v2.1** dans la vue Serveurs > **Démarrer**

      Conseil : Vous pouvez gérer WASCE dans Eclipse: cliquez avec le bouton droit de la souris sur **IBM WASCE v2.1** > **Lancer la console WASCE** . Les valeurs par défaut **Username** et **Password** sont `system` et `manager` .
4. Configurez le serveur et l'environnement d'exécution pour les services Web.
   a) **Fenêtre** > **Préférences** > **Services Web** > **Serveur et exécution**
   b) Sélectionnez **IBM WASCE v2.1 Server** comme serveur.
   c) Laissez **Apache Axis** comme environnement d'exécution du service Web.
5. Créez un projet Web dynamique.
   a) **Fichier** > **Nouveau** > **Projet Web dynamique**.
      Nommez le projet `StockQuoteAxis`.
   b) Cochez la case **Ajouter un projet à un fichier EAR** > **Nouveau ...**
   c) Dans la page **Projet d'application EAR** , entrez **Project name** `StockQuoteAxisEAR` > **Terminer**
      Répondez **OK** en réponse à la boîte de dialogue qui vous suggère de passer à la perspective Java EE ou de rester dans la perspective Java pour suivre exactement ces instructions.
   d) **IBM WASCE 2.1 server** est sélectionné comme environnement d'exécution cible. Acceptez-la et les autres valeurs par défaut > **Terminer**.
      Répondez **OK** en réponse à la boîte de dialogue qui vous suggère de passer à la perspective Java EE ou de rester dans la perspective Java pour suivre exactement ces instructions.
6. Importation de l'exemple de programme `StockQuoteAxis.java`
   a) Ouvrez le projet Web **StockQuoteAxis** > Cliquez avec le bouton droit de la souris sur le dossier **src** > **Importer ...**
   b) Sélectionnez **Général** > **Système de fichiers** > **Suivant**
   c) Accédez à *MQ_INSTALLATION_PATH*`\tools\soap\samples\java\server` > cochez **StockQuoteAxis.java** > **Terminer**
      MQ_INSTALLATION_PATH représente le répertoire de niveau supérieur dans lequel WebSphere MQ est installé. Vous devez mettre en évidence le répertoire du serveur pour voir les fichiers qu'il contient.

7. Corrigez l'erreur de compilation en déplaçant `StockQuoteAxis.java` dans le package approprié.
   a) Ouvrez `StockQuoteAxis.java` et cliquez avec le bouton droit de la souris sur le problème > **Correctif rapide**
   b) Cliquez deux fois sur **Move'StockQuoteAxis.java'to package'soap.server'** > **Save**.
8. Créez un service Web à partir de `StockQuoteAxis.java` .
   a) Cliquez avec le bouton droit de la souris sur `StockQuoteAxis.java` > **Services Web** > **Créer un service Web** > **Suivant**.

      Acceptez la configuration par défaut pour le service:

      Type de service Web
      Service Java beanWeb ascendant

      Implémentation de service
      soap.server.StockQuoteAxis

      serveur
      `IBM WASCE v2.1 server`

      Environnement d'exécution de service Web
      Apache Axis

      projet de service
      Axe StockQuote

      Projet EAR de service
      StockQuoteAxisEAR

      Configuration
      Aucune génération de client
9. Sélectionnez les méthodes d'accès et le style du service Web > **Suivant**.

   Démarrez le serveur si vous y êtes invité.
   a) Laissez toutes les méthodes sélectionnées.
   b) Sélectionnez le style document/littéral (encapsulé).
10. **Terminer**

    Une fois le service déployé, recherchez le fichier `StockQuoteAxis.wsdl` généré dans le dossier `WebContent\wsdl` du projet `StockQuoteAxis` Web.
11. Testez le service à l'aide de HTTP avec l'explorateur de services Web.
    a) Cliquez avec le bouton droit de la souris sur `StockQuoteAxis.wsdl` > **Tester avec l'explorateur de services Web**.
    b) Cliquez sur l'opération **getQuote** dans les actions **StockQuoteAxisSoapBinding** de la fenêtre **Explorateur de services Web** .
    c) Entrez `ibm` dans la zone d'entrée **symbol** > **Go**
12. Testez le service à l'aide du transport WebSphere MQ pour SOAP.

    Pour déployer le service, utilisez **`SimpleJavaListener`**, qui se trouve dans `com.ibm.mq.soap.jar`. Vous devez ajouter les bibliothèques Java et SOAP WebSphere MQ au chemin de génération.
    a) Cliquez avec le bouton droit de la souris sur le projet Web **StockQuoteAxis** > **Chemin de génération** > **Configurer le chemin de génération ...**
    b) Cliquez sur l'onglet **Bibliothèques** > **Ajouter des fichiers JAR externes ...**. Accédez à *MQ_INSTALLATION_PATH*`\java\lib` . et sélectionnez tous les fichiers `.jar` > **Ouvrir** > **Ajouter des fichiers JAR externes ...**. Accédez à *`WMQ Install directory`*`\java\lib\soap` et sélectionnez tous les fichiers `.jar` > **Ouvrir** > **OK**.

       MQ_INSTALLATION_PATH représente le répertoire de niveau supérieur dans lequel WebSphere MQ est installé.
    c) Dans l'explorateur de projets, cliquez avec le bouton droit de la souris sur `StockQuoteAxis\Java`

Resources\Libraries\com.ibm.mq.soap.jar\com.ibm.mq.soap.transport.jms\SimpleJavaListener.class\ SimpleJavaListener > **Exécuter en tant que ...**> **Configurations d'exécution ...**

Conseil :

S'il n'existe aucune configuration pour SimpleJavaListener , cliquez sur l'icône **Nouvelle configuration** dans la page **Créer, gérer et exécuter des configurations** de l'assistant **Configurations d'exécution** .
SimpleJavaListener n'a pas de commande pour l'arrêter. Pour surveiller ou arrêter **SimpleJavaListener**, ouvrez la **perspective Débogage** dans Eclipse.

d) Ouvrez l'onglet **(x) = Arguments** . Dans la zone d'entrée **Arguments du programme** , entrez les paramètres dans **SimpleJavaListener**.

Pour cet exemple, entrez

```
-u "jms:/queue?destination=REQUESTAXIS@QM1&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" -n 10
```

Remarque : Le service cible est StockQuoteAxis pour correspondre au nom de service cible créé dans le descripteur de déploiement de service, StockQuoteAxis\WebContent\WEB-INF\server-config.wsdd . amqwdeployWMQService crée un service cible appelé soap.server.StockQuoteAxis. Dans cet exemple, vous utilisez les mêmes StockQuoteAxis.class et service-config.wsdd que le serveur HTTP.

e) Dans le même onglet, configurez **Répertoire de travail** pour référencer le fichier server-config.wsdd :
${workspace_loc:StockQuoteAxis/WebContent/WEB-INF}

a) **Exécutez**

Les erreurs sont écrites sur la console. Si la console reste vide, cela signifie que **SimpleJavaListener** a démarré correctement.

b) Pour tester le déploiement, exécutez un client Axis StockQuote, développé dans la tâche «Développement d'un client JAX-RPC pour le transport WebSphere pour SOAP à l'aide d' Eclipse», à la page 1005.

Exemple: exemple de programme de l'axe StockQuote

L'exemple de service Web Java, StockQuoteAxis.java, est installé dans *WMQ install directory*\tools\soap\samples\java\server . StockQuoteAxis.java, Figure 170, à la page 996, comporte quatre méthodes:

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteTran(String symbol)

```
package soap.server;
import java.lang.Thread;
import java.io.FileWriter;
public class StockQuoteAxis {
    public float getQuote(String symbol) throws Exception {
        return ((float) 55.25);
    }
    public void getQuoteOneWay(String symbol) throws Exception {
        try {
            // Write the results for this service to a file
            FileWriter f = new FileWriter("getQuoteOneWay.txt", true);
            f.write("One way service result via proxy is: 44.44\n");
            f.close();
        } catch (Exception ee) {
            System.out.println("Error writing result file in getQuoteOneWay");
            ee.printStackTrace();
        }
    }
    public int asyncQuote(int delay) {
        try {
            Thread.sleep(delay);
        } catch (Exception e) {
            System.out.println("Exception in asyncQuote during sleep");
        }
        return delay;
    }

    public float getQuoteTran(String symbol) throws Exception {
        if (symbol.equalsIgnoreCase("ROLLBACK")) {
            System.out.println("Rollback was requested,
                exiting from service by calling System.exit().");
            System.exit(0);
        }
        return ((float) 55.25);
    }
}
```

Figure 170. Axe StockQuote

Que faire ensuite

Déployez le service à l'aide de WebSphere MQ Transport for SOAP, au lieu de HTTP à l'aide de la commande **amqwdeployWMQService**.

La commande comporte une option, `axisDeploy`, qui déploie le service en créant un descripteur de déploiement Apache Axis 1.4 . Le programme d'écoute SOAP WebSphere MQ exécute le service. Le programme d'écoute SOAP est appelé SimpleJavaListener et est fourni avec le transport WebSphere MQ pour SOAP.

Tâches associées

Développement d'un service .NET 1 ou 2 pour WebSphere MQ transport for SOAP à l'aide de Microsoft Visual Studio 2008
Développez le service Web SampleStockQuote pour .NET 1 ou .NET 2 à l'aide de Microsoft Visual Studio 2008

Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications JEE. Cette tâche est l'étape 2 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur d'applications WebSphere à l'aide du protocole W3C SOAP sur JMS.

Développement d'un service .NET 1 ou 2 pour WebSphere MQ transport for SOAP à l'aide de Microsoft Visual Studio 2008

Développez le service Web SampleStockQuote pour .NET 1 ou .NET 2 à l'aide de Microsoft Visual Studio 2008

Pourquoi et quand exécuter cette tâche

Créez le service StockQuote avec une implémentation de code à l'aide de Visual Studio 2008.

Procédure

1. Créez un modèle pour le service et vérifiez qu'il s'exécute sur HTTP.
   a) Démarrez Visual Studio 2008 > **Fichier** > **Nouveau** > **Projet ...**. Sélectionnez **C#** Project Type, **, NET Framework 2**et **ASP.NET Application de service Web**. Entrez le **Nom:** et **Nom de la solution:** `StockQuoteDotNet` > **OK**
   b) Cliquez avec le bouton droit de la souris sur **Service1.asmx** dans l' **Explorateur de solutions** > **Renommer** > `StockQuote.asmx`.
   c) Remplacez le fragment de code `public class Service1` par `public class StockQuote`.
   d) Cliquez avec le bouton droit de la souris sur **StockQuote.asmx** dans l' **Explorateur de solutions** > **Ouvrir avec ...** > **Editeur XML**. Remplacez `Class="StockQuoteDotNet.Service1"` par `Class="StockQuoteDotNet.StockQuote"`
   e) Remplacez le fragment de code `[WebService(Namespace = "http://tempuri.org/")]` par `[WebService(Namespace = "http://stock.samples/")]`.
   f) Supprimez la ligne de code `[ToolboxItem(false)]`.
   g) Vérifiez que tout est correct jusqu'à présent: **Déboguer** > **Démarrer le débogage (F5)**. Vérifiez la sortie dans l'explorateur.
2. Ajoutez les méthodes de l'exemple `SQDNNonInline.asmx.cs`et testez le service sur HTTP.
   a) Ouvrez *MQ_INSTALLATION_PATH*`\tools\soap\samples\dotnet\SQDNNonInline.asmx.cs` et remplacez la méthode `HelloWorld` par les quatre méthodes `Quote` ; voir Figure 171, à la page 998. *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere MQ est installé.
   b) **Génération** > **Reconstruction** de la solution > Cliquez avec le bouton droit de la souris sur l'une des **unités d'exécution** en erreur > **Résolution** > Utilisation de **System.Threading**.
   c) Appuyez sur F5 pour démarrer le débogage.

      Le service n'est pas conforme au profil de base WS-I v1.1. Vous avez le choix entre modifier l'annotation WebMethod de `[SoapRpcMethod]` en `[SoapDocumentMethod]`ou supprimer l'annotation `[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]`.
   d) Appuyez sur F5 pour vérifier l'implémentation à l'aide de HTTP.
3. Générez WSDL, des clients et exécutez le service à l'aide du transport WebSphere MQ pour SOAP.
   a) Ouvrez une fenêtre de commande dans l'arborescence des répertoires de projet, dans laquelle le fichier `StockQuote.asmx` est stocké.
   b) (Facultatif) Utilisez amqwdeployWMQService pour générer des artefacts. Le gestionnaire de files d'attente doit être démarré:

      ```
      amqswdeployWMQService -f StockQuote.asmx
      -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
      &connectionFactory=()
      &destination=REQUESTDOTNET@QM1
      &targetService=StockQuote.asmx"
      StockQuote.asmx StockQuote.wsdl
      ```

      Tous les artefacts sont créés dans l'arborescence de répertoires `./generated` .
   c) (Facultatif) Générez uniquement le WSDL pour l'appel du service à l'aide du transport WebSphere MQ pour SOAP.

      ```
      amqswsdl -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
      &connectionFactory=()
      &destination=REQUESTDOTNET@QM1
      &targetService=StockQuote.asmx"
      StockQuote.asmx StockQuote.wsdl
      ```
   a) Exécutez le programme d'écoute .NET.
   Utilisez `.\generated\server\startWMQNListener.cmd` ou entrez la commande suivante:

```
amqSOAPNETListener -u "jms:/queue?initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=()
&destination=REQUESTDOTNET@QM1
&targetService=StockQuote.asmx"
```

4. Testez le service à l'aide d'un client généré à partir du WSDL ou à l'aide des clients générés par **amqwdeployWMQService**.

Exemple de code

L'exemple de service Web .NET, StockQuoteDotNet, est installé dans *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet. *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ est installé. La liaison de service Web des exemples publiés diffère légèrement de la liaison utilisée dans la tâche. La tâche utilise les valeurs par défaut utilisées dans Visual Studio 2008.

Il existe deux exemples de services Web .NET Framework 1 et .NET Framework 2. StockQuoteDotNet.asmx, est un service en ligne. SQDNNoninline.asmx, est un service Web code-behind implémenté par SQDNNoninline.asmx.cs.

StockQuoteDotNet comporte quatre méthodes:

1. float getQuote(String symbol)
2. void getQuoteOneWay(String symbol).
3. int asyncQuote(int delay)
4. float getQuoteDOC(String symbol)

```
<%@ WebService Language="C#" Class="StockQuoteDotNet" %>
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;
[WebService (Namespace="http://stock.samples")]
public class StockQuoteDotNet {
     [WebMethod] [ SoapRpcMethod(OneWay=true) ]
     public void getQuoteOneWay(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
          System.Console.WriteLine("getQuoteOneWay was invoked.");
     }
     [WebMethod] [SoapRpcMethod]
     public float getQuote(String symbol) {
          if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
          return 88.88F;
     }
     [WebMethod] [SoapRpcMethod]
     public int asyncQuote(int delay) {
          Thread.Sleep(delay);
          return delay;
     }
     [WebMethod]
     public float getQuoteDOC(String symbol) {
          return 77.77F;
     }
}
```

Figure 171. Service en ligne: StockQuoteDotNet.asmx

```
<%@ WebService Language="C#" Codebehind="SQDNNonInline.asmx.cs" Class="SQDNNonInline" %>
```

Figure 172. Code-behind: Conception SQDNNonInline.asmx

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Web.Services.Description;
using System.Threading;

[WebService(Namespace = "http://stock.samples")]
public class SQDNNonInline : System.Web.Services.Protocols.SoapHttpClientProtocol
{

    [WebMethod]
    [SoapRpcMethod(OneWay = true)]
    public void getNonInlineQuoteOneWay(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(5000);
        System.Console.WriteLine("getNonInlineQuoteOneWay was invoked.");
    }

    [WebMethod]
    [SoapRpcMethod]
    public float getNonInlineQuote(String symbol)
    {
        if (symbol.ToUpper().Equals("DELAY")) Thread.Sleep(10000);
        return 88.88F;
    }

    [WebMethod]
    [SoapRpcMethod]
    public int asyncNonInlineQuote(int delay)
    {
        Thread.Sleep(delay);
        return delay;
    }
    [WebMethod]
    public float getNonInlineQuoteDOC(String symbol)
    {
        return 77.77F;
    }
}
```

Figure 173. Code-behind: Implémentation: SQDNNonInline.asmx.cs

Tâches associées

Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse
Développez un service Web Axis 1.4 à exécuter en utilisant WebSphere MQ comme fournisseur de services. Utilisez votre environnement de développement de service Web normal pour créer un service à déployer sur Axis 1.4.

Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications JEE. Cette tâche est l'étape 2 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur d'applications WebSphere à l'aide du protocole W3C SOAP sur JMS.

Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS

Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications JEE. Cette tâche est l'étape 2 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur d'applications WebSphere à l'aide du protocole W3C SOAP sur JMS.

Avant de commencer

Utilisez Rational Application Developer pour créer le service Web EJB. L'assistant de service Web de Rational Application Developer dispose d'une option permettant de créer un service Web à l'aide de la recommandation de candidat W3C pour la liaison SOAP sur JMS. Rational Application Developer 7.54 est requis. L'exercice a utilisé Rational Application Developer inclus dans Rational Software Architect for WebSphere Software v7.5.5.1,

L'EJB est déployé sur WebSphere Application Server à partir de Rational Application Developer dans le cadre de cette tâche. Vous devez effectuer les «Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS», à la page 1038

Pour créer le WSDL réellement utilisé dans la tâche, vous devez d'abord exécuter la tâche «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992. Vous pouvez ensuite importer le WSDL à partir du projet Web dynamique dans l'espace de travail Galileo Eclipse ou à partir du service Web HTTP en cours d'exécution déployé sur WASCE.

Il se peut que WebSphere Application Server soit toujours en cours d'exécution suite à l'exécution de «Configuration des ressources WebSphere Application Server», à la page 1039. Si ce n'est pas le cas, vous pouvez le démarrer à partir de la vue Serveurs dans RAD.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous redéployez le service `StockQuoteAxis` qui s'exécute en tant que service JAX-RPC Axis exécuté par **`SimpleJavaListener`** à l'aide du transport WebSphere MQ pour SOAP, vers un service JAX-WS s'exécutant dans WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS.

La migration du service depuis **`SimpleJavaListener`** vers WebSphere Application Serverse fait en deux parties:

1. Générez l'interface de service Web à partir du WSDL du service à l'aide de l'assistant de service Web EJB descendant dans Rational Application Developer.
2. Implémentation du service en important l' WebSphere MQ exemple SOAP `StockQuoteAxis.java`.

Une autre approche aurait consisté à générer le service de bas en haut, à partir du `StockQuoteAxis.java`. Toutefois, pour vous assurer que l'interface du service migré est identique, l'approche descendante est meilleure, car elle utilise le même WSDL.

Le service Web est développé pour le conteneur d'EJB et non pour le conteneur Web car le support JMS fait partie du conteneur d'EJB.

Procédure

1. Démarrez Rational Application Developeret vérifiez que WebSphere Application Server est en cours d'exécution.
   a) Démarrez Rational Application Developer dans un nouvel espace de travail.
   b) Ouvrez la perspective Java EE .
   c) Ouvrez l'onglet **Serveurs** et vérifiez que WebSphere Application Server est en cours d'exécution.
      - S'il n'y a pas de WebSphere Application Server v7.0 dans la vue, cliquez avec le bouton droit de la souris dans la vue > **Nouveau** > **Serveur**. Suivez les options de l'assistant pour créer une instance WebSphere Application Server v7.0 .
      - Si le serveur est présent, mais qu'il n'est pas démarré, cliquez sur la flèche pour le démarrer.
      - Pour vérifier les propriétés et accéder rapidement aux journaux du serveur, cliquez avec le bouton droit de la souris sur **WebSphere Application Server v7.0 à l'adresse localhost** > **Propriétés** > **WebSphere Application Server**.
      - Pour administrer le serveur, utilisez un navigateur externe et ouvrez l'URL `http://localhost:9061/ibm/console/unsecureLogon.jspou` cliquez avec le bouton droit de la souris sur **WebSphere Application Server v7.0 sur localhost** > **Exécuter la console d'administration**.
      - Le paramètre par défaut consiste à publier automatiquement. De nombreuses personnes préfèrent déployer les mises à jour sur le serveur manuellement. Cliquez deux fois sur **WebSphere Application Server v7.0 sur localhost**et développez le triangle **Publication** dans la fenêtre**Présentation** . Cliquez sur **Ne jamais publier automatiquement**.
      - Une autre valeur par défaut que vous pouvez modifier consiste à désélectionner la case à cocher **Arrêter le serveur lors de l'arrêt du plan de travail** dans la fenêtre **Présentation** .

2. Créer les projets JEE

   Vous devez créer un projet d'application d'entreprise (EAR) et un projet EJB (Enterprise Java Bean).

   a) **Fichier** > **Nouveau** > **Projet d'application d'entreprise**. Nommez le projet `W3CJMSEAR` > **Terminer**.

      Les valeurs par défaut doivent identifier WebSphere Application Server v7.0 comme environnement d'exécution cible et EAR version 5.0. La configuration par défaut doit être sélectionnée.

   b) **Fichier** > **Nouveau** > **Projet EJB**. Nommez le projet `W3CJMSEJB`. Sélectionnez `W3CEARJMS` comme **Nom du projet EAR** > **Suivant**.

      La version du module EJB par défaut est 3.0 et la configuration par défaut est réutilisée.

   c) Décochez la case **Créer un module JAR de client EJB** > **Terminer**.

3. Générez et déployez le service Web EJB à partir du WSDL `StockQuoteAxis` .

   a) **Exécutez** > **Lancez l'explorateur de services Web**.

   b) Sélectionnez la page WSDL à l'aide des icônes de la fenêtre**Explorateur de services Web** > cliquez sur **WSDL principal** dans le Navigator.

   c) Dans la fenêtre **Actions** , entrez ou accédez à l'URL WSDL pour `StockQuoteAxis.wsdl`.

      Si WASCE s'exécute avec `StockQuoteAxis` déployé en tant que service HTTP, l'URL est la suivante:

      ```
      http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
      ```

      Si le WSDL se trouve dans le système de fichiers, l'URL peut être:

      ```
      File:\Dirpath\StockQuoteAxis\WebContent\wsdl\StockQuoteAxis.wsdl
      ```

   d) Cliquez sur la ligne contenant l'URL importée dans l'arborescence Navigator .

      Il s'agit de la ligne qui suit immédiatement **WSDL principal**, s'il s'agit du premier WSDL que vous avez importé dans l'explorateur de services Web.

   e) Dans la fenêtre **Actions** , cliquez sur **Lancer l'assistant de service Web** > **Web Service Skeleton** > **Aller**

   f) Dans l'assistant de service Web, sélectionnez **Service Web EJB descendant**

      Sélectionnez ou vérifiez la configuration à l'aide des informations de la page Tableau 141, à la page 1001 Vérifier **Remplacer les fichiers sans avertissement** > **Suivant**.

      Tableau 141. Configuration du service Web EJB descendant

Zone	**Valeur**
serveur	`WebSphere Application Server v7.0`
Exécution du service Web	`IBM WebSphere JAX-WS`
projet de service	`W3CJMSEJB`
Projet EAR de service	`W3CJMSEAR`
Configuration :	`No client generation`

   g) Dans la page sous-titrée, **Spécifiez les options de création d'un service Web WebSphere JAX-WS EJB Top Down**, cochez la case **Passer à la liaison JMS**. Cochez également **Activer le style d'encapsuleur**, **Copier WSDL dans le projet** et **Générer un descripteur de déploiement de service Web** > **Suivant**.

   h) Sur la page intitulée **WebSphere JAX-WS JMS Binding Configuration**, cochez la case **Utiliser le protocole d'interopérabilité SOAP/JMS** et fournissez des valeurs à partir de Tableau 142, à la page 1002, en laissant les autres zones vides > **Suivant**.

Tableau 142. WebSphere Configuration de liaison JMS JAX-WS	
Zone	**Valeur**
Destinations JMS	queue
Nom JNDI de la destination :	requestaxis
Fabrique de connexions JMS	qm1
Nom de la réponse	W3CJMSEAR
Configuration :	replyaxis

a) Sur la page intitulée **WebSphere JAX-WS Router Project Configuration**, entrez qm1as dans la zone **ActivationSpec JNDI name** > **Suivant**.

RAD prend environ 30 secondes à une minute pour générer et déployer le projet.

b) Ignorez les options de la page **Publication du service Web** > **Terminer**.

4. Vérifiez le fichier WSDL généré.

Vous avez demandé que le WSDL spécifique au service soit généré et sauvegardé dans le projet.

a) Dans le navigateur Enterprise Explorer, ouvrez le dossier **W3CJMSEJB** > **ejbmodule** > **META-INF** > **wsdl**. Cliquez deux fois sur StockQuoteAxis.wsdl pour l'ouvrir dans l'éditeur WSDL.

Inspectez la liaison ; vous voyez l'URL JMS:

```
jms:jndi:requestaxis?jndiConnectionFactoryName=qm1&targetService=StockQuoteAxis
```

5. Etape facultative: Liaison de l'EJB à SOAP sur HTTP à l'aide de JAX-WS.

Le fait de fournir deux liaisons à l'EJB permet aux clients de choisir des liaisons SOAP pour appeler le service Web. Il fournit également aux clients les moyens d'interroger le serveur Web pour obtenir son WSDL à l'aide de HTTP.

Les étapes de liaison d'un EJB à SOAP sur HTTP ne sont pas incluses dans la tâche.

6. Implémentez et redéployez StockQuoteAxis à l'aide de l'exemple StockQuoteAxis.java

a) Dans le navigateur de l'explorateur d'entreprise, ouvrez le dossier **W3CJMSEJB** > **Services** Cliquez deux fois sur StockQuoteAxisService pour ouvrir la classe d'implémentation dans un éditeur Java.

b) Ouvrez l'exemple de programme StockQuoteAxis.java dans le dossier *WebSphere MQ Installation directory*\tools\soap\samples\java\server > Sélectionnez toutes les méthodes, mais pas le nom de classe > **Copier**.

c) Dans StockQuoteAxisSoapBindingImpl.java , sélectionnez toutes les méthodes, mais pas le nom de classe, et collez les méthodes à partir de StockQuoteAxis.java.

d) Ajoutez une instruction d'impression à la sortie dans la console WebSphere Application Server lorsque le service est appelé.
Modifiez la méthode getQuote(String symbol):

```
public float getQuote(String symbol) {
    System.out.println("StockQuoteAxisSoapBindingImpl called with symbol: "
                       + symbol);
    return ((float) 55.25);
```

e) Corrigez les importations: **Source** > **Organiser les importations** > **Enregistrer**.

f) Corrigez les trois erreurs car l'implémentation ne correspond pas à l'interface.

Les erreurs sont dues à trois des méthodes de StockQuoteAxis.java qui génèrent des exceptions et au WSDL du service qui ne contient pas de messages d'erreur. Le problème est diagnostiqué comme étant une non-concordance entre les signatures de méthode et les annotations de service Web de méthode.

Annotez les méthodes avec @WebFault et régénérez le WSDL, ou conservez l'interface inchangée et supprimez les exceptions.

Pour conserver la même interface, supprimez les trois `throws exception` des signatures de méthode > **Sauvegarder**.

Que faire ensuite

«Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS», à la page 1049

Tâches associées

Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse
Développez un service Web Axis 1.4 à exécuter en utilisant WebSphere MQ comme fournisseur de services. Utilisez votre environnement de développement de service Web normal pour créer un service à déployer sur Axis 1.4.

Développement d'un service .NET 1 ou 2 pour WebSphere MQ transport for SOAP à l'aide de Microsoft Visual Studio 2008
Développez le service Web SampleStockQuote pour .NET 1 ou .NET 2 à l'aide de Microsoft Visual Studio 2008

Développement de clients de service Web WebSphere MQ pour WebSphere MQ transport for SOAP

Utilisez votre environnement de développement normal pour développer des clients de service Web à utiliser avec le transport WebSphere MQ pour SOAP.

Avant de commencer

Créez le service. Vous pouvez utiliser l'un des exemples de la rubrique «Développement de services Web pour WebSphere MQ transport for SOAP», à la page 991.

Faites des choix sur la façon dont vous allez développer, déployer et utiliser les clients, et où aller pour obtenir le WSDL pour la génération de client.

Choisissez votre approche pour le développement de clients et de services pour WebSphere MQ transport for SOAP.

Il y a deux approches.

1. Utilisez les outils de développement standard, développez un service HTTP et un client, puis utilisez l'URL du transport WebSphere MQ pour SOAP.
2. Utilisez les outils et les exemples fournis avec WebSphere MQ transport for SOAP.

Si vous empruntez la route HTTP, vous pouvez exécuter le service sur un serveur HTTP et également à l'aide du transport WebSphere MQ pour SOAP. Pour l'exécuter à l'aide de WebSphere MQ transport for SOAP, configurez le programme d'écoute WebSphere MQ approprié pour SOAP et configurez les chemins et les descripteurs de déploiement pour exécuter le service. Les outils fournis par WebSphere MQ transport for SOAP permettent d'effectuer la configuration pour vous. Vous pouvez également configurer l'environnement pour exécuter les programmes d'écoute.

Les outils fournis avec WebSphere MQ transport for SOAP sont utiles pour démarrer et apprendre à déployer le transport. Pour le travail de production, il est avantageux d'utiliser des outils standard et de déployer le même service accessible à différents transports SOAP.

Décider du type de client à développer

Vous devez décider du type de client de service Web à développer. Le choix varie selon que vous connaissez l'interface de service et l'adresse du service.

Si l'interface est connue, utilisez les outils Axis ou .NET pour générer des classes de client proxy à partir de l'interface de service. Les classes de client proxy facilitent l'écriture d'un client pour appeler le service. Si l'emplacement du service est connu lorsque vous développez le client, utilisez l'interface

de proxy statique. Si l'emplacement du service change, par exemple si le service est redéployé sur un serveur de production, utilisez l'interface de proxy dynamique.

Si l'interface de service n'est pas connue au moment où vous développez un client, sur Axis, vous pouvez créer un client DII (Dynamic Invocation Interface) pour Axis 1.4. Un client DII utilise une interface générique pour appeler n'importe quel service. Pour transmettre correctement des paramètres à un service particulier, vous devez générer l'interface de service spécifique à l'aide d'un programme. Générez l'interface à l'aide d'un programme dans le client ou en chargeant le WSDL du service dans le client. Sur Axis2, vous pouvez créer un client Dispatch. Le client Dispatch utilise un modèle de document pour décrire la demande du client, alors qu'un client DII utilise un modèle d'appel. Les deux travaillent à la génération dynamique de la demande.

Obtenir le WSDL du service

A l'exception du cas où l'interface de service est générée à l'aide d'un programme, vous devez d'abord obtenir le WSDL du service afin de créer un client de service Web. Le WSDL du service peut être obtenu à partir de trois sources différentes:

1. Directement à partir de l'implémentation du service Web à l'aide d'un outil tel que **java2wsdl** (Axis) ou **disco** (.NET).
2. En interrogeant le service Web à l'aide de l'URL: *Web service http url*?wsdl.
3. A partir d'un fichier, sur un système de fichiers ou à partir d'un registre tel qu'UDDI ou WebSphere Service Registry and Repository.

Remarque : Si le service n'est pas accessible via HTTP, la requête WSDL ne fonctionne pas. Le service lui-même peut être disponible uniquement à l'aide du transport WebSphere MQ pour SOAP.

Le WSDL généré par **amqwdeployWMQService** est différent du WSDL généré à l'aide de **java2wsdl** ou de **disco**. Le WSDL généré est également différent de tout WSDL avec lequel vous avez pu démarrer pour créer le service "Top Down". Sur Axis, le descripteur de déploiement server-config.wsdd mappe le message SOAP produit par un client vers une opération et un service. **amqwdeployWMQService** génère un descripteur de déploiement différent à partir d' Eclipse.

Le WSDL que vous utilisez pour générer des clients dépend de la manière dont le service est déployé:

Déployé à l'aide de amqwdeployWMQService
Utilisez le WSDL généré par **amqwdeployWMQService** . Spécifiez l'indicateur -w et sélectionnez le WSDL rpcLiteral . Pour des raisons de compatibilité, vous pouvez sélectionner le WSDL rpcEncoded . rpcEncoded WSDL fonctionne uniquement avec les clients .NET et Axis 1.4 .

Déploiement manuel à l'aide de SimpleJavaListener
Utilisez l'un des fichiers WSDL suivants:

1. WSDL utilisé pour définir le service ou stocké dans un référentiel.
2. WSDL généré à partir du service par **java2wsdl** .
3. WSDL interrogé à l'aide de l'URL *Web service http url* ?wsdl, si disponible à partir d'un serveur HTTP. Vous pouvez exécuter un outil tel que l'explorateur de services Web pour importer la définition de service directement dans Eclipse.

Vous devrez peut-être modifier l'URI du service. Remplacez l'adresse du service HTTP par l'URI du transport WebSphere MQ pour SOAP.

Déploiement manuel à l'aide de amqSOAPNETListener.
Utilisez l'un des fichiers WSDL suivants:

1. WSDL utilisé pour définir le service ou stocké dans un référentiel.
2. WSDL obtenu à partir de la classe de service .NET (.asmx). à l'aide de **disco**.
3. WSDL interrogé à l'aide de l'URL *Web service http url* ?wsdl, si disponible. Vous pouvez exécuter un outil tel que l'explorateur de services Web pour importer la définition de service directement dans Eclipse.
4. WSDL obtenu en exécutant **amqswsdl** sur la classe de service .NET (.asmx).

Vous devrez peut-être modifier l'URI du service. Remplacez l'adresse du service HTTP par l'URI du transport WebSphere MQ pour SOAP.

Déployé dans Windows Communication Foundation
Obtenez le WSDL du service à l'aide de l'URL *Web service http url*?wsdl. Le service doit être défini avec la configuration de comportement `serviceMetaData` dans le cadre de la définition du service.

Déploiement sur une plateforme de serveur différente.
Suivez les instructions fournies avec la plateforme pour obtenir le WSDL de service correct.

Pourquoi et quand exécuter cette tâche

Développez des clients à l'aide d'outils de développement standard. Les tâches suivantes expliquent comment générer des clients pour .NET 1 et 2, Axis 1.4 (JAX-RPC) et Axis2 (JAX-WS). Pour Windows Communication Foundation, consultez les liens des tâches connexes.

Développement d'un client JAX-RPC pour le transport WebSphere pour SOAP à l'aide d' Eclipse

Développez un client de service Web Axis 1.4 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Avant de commencer

Le service doit être disponible. Si vous suivez la tâche en tant qu'exercice pratique, utilisez l'espace de travail et le service que vous avez créés dans la tâche, «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992. Vous devez disposer d'un serveur d'applications en cours d'exécution dans eclipse qui prend en charge les services Web Axis 1.4 . Dans cette tâche, nous utilisons gratuitement WebSphere Application Server Community Edition Version 2.1.4. Il est configuré dans le cadre de la tâche «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992. Vous pouvez également utiliser Tomcat 6, qui est un serveur d'applications open source plus petit.

Pourquoi et quand exécuter cette tâche

La tâche montre le développement de trois types de client pour l'exemple de service Axis StockQuoteà l'aide d' Eclipse exécuté sous Windows. Les clients sont un client statique et dynamique développé à l'aide du proxy client et un client DII.

Deux approches alternatives pour générer les proxys client à partir de WSDL sont illustrées:

1. Génération de proxys client à l'aide de **amqwdeployWMQService** .
2. Importation de WSDL dans Eclipseet utilisation de l'assistant de service Web pour générer les proxys client.

Procédure

1. Démarrez les développeurs Eclipse IDE for Java EE .
2. Créez un projet Java appelé `StockQuoteAxisClient`:
   a) Passez à la perspective Java > **Fichier** > **Nouveau** > **Projet Java**. Dans la zone **Project name** de la **page Créer un projet Java** , entrez `StockQuoteAxisEclipseClient`. Assurez-vous que l'environnement d'exécution est **J2SE1-1.4** ou **J2SE-1.5** > **Suivant** .
   b) Sur la page **Paramètres Java** , sélectionnez l'onglet **Bibliothèques** > **Ajouter des fichiers JAR externes ...**
   c) Accédez à *MQ_INSTALLATION_PATH*/java/lib et sélectionnez tous les fichiers `.jar` > **Ouvrir**.
   MQ_INSTALLATION_PATH représente le répertoire dans lequel WebSphere MQ est installé.
   d) Accédez à *MQ_INSTALLATION_PATH*/java/lib/soap et sélectionnez tous les fichiers `.jar` > **Ouvrir**. Vous devez avoir installé `axis.jar` depuis le support d'installation WebSphere MQ dans ce répertoire.

MQ_INSTALLATION_PATH représente le répertoire dans lequel WebSphere MQ est installé.

e) L'onglet **Bibliothèque** fait désormais référence à tous les fichiers `.jar` nécessaires à la génération du client > **Terminer**.

3. Suivez l'une des deux approches suivantes pour créer des proxys dans Eclipse pour l'exemple de service Web Axis StockQuote:

   - Générez les proxys client à l'aide de **amqwdeployWMQService** .

     a. Créez un gestionnaire de files d'attente. Pour la tâche, créez `QM1` comme gestionnaire de files d'attente par défaut.

     b. Créez un répertoire de travail, `samples`. Copiez l'exemple de programme `StockQuoteAxis.java` dans `samples/soap/server`.

     c. Modifiez `amqwsetcp.cmd` dans *MQ_INSTALLATION_PATH*`/bin` pour inclure le répertoire en cours dans le chemin d'accès aux classes. *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ est installé.

     d. Ouvrez une fenêtre de commande dans `samples` et exécutez la commande **amqwsetcp** modifiée

     e. Créez un fichier WSDL pour le service Axis StockQuoteen exécutant la commande suivante:

     ```
     amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genAxisWsdl
     -u "jms:/queue?destination=REQUESTAXIS
     &initialContextFactory=com.ibm.mq.jms.Nojndi
     &connectionFactory=(connectQueueManager(QM1)binding(auto))"
     ```

     A faire : Utilisez "/"plutôt que "." ou "\" lorsque vous utilisez des commandes Java.

     Conseil : Au lieu d'importer les proxys générés dans Eclipse, vous pouvez importer le WSDL généré à partir de `.samples/generated`. Les proxys qui en résultent diffèrent de deux manières:

     i) Les noms de package sont différents-que vous pouvez restructurer.

     ii) Les proxys générés par Eclipse incluent une classe auxiliaire supplémentaire, `StockQuoteAxisProxy.java`

     f. Créez les proxys client pour le service Axis StockQuoteen exécutant la commande suivante:

     ```
     amqwdeployWMQService -f soap/server/StockQuoteAxis.java -c genProxiestoAxis
     -u "jms:/queue?destination=REQUESTAXIS
     &initialContextFactory=com.ibm.mq.jms.Nojndi
     &connectionFactory=(connectQueueManager(QM1)binding(auto))"
     ```

     g. Importez les proxys client dans StockQuoteAxisClient:

     i) Cliquez avec le bouton droit de la souris sur **StockQuoteAxisClient\src** > Sélectionnez **Système de fichiers** > **Suivant** > **Parcourir ...** > recherchez le dossier `.\samples\generated\client\remote\soap\server` > **OK**.

     ii) Cochez **server** dans la page**Importer** > **Terminer**.

     h. Restructurez le nom du package en `soap.server`.

     i) Cliquez avec le bouton droit de la souris sur le package contenant les proxys client > **Restructurer** > **Renommer**. Entrez **New name**: `soap.server` > conservez les valeurs par défaut sélectionnées pour les autres options > **OK** . Toutes les erreurs sont corrigées.

   - Générez les proxys client à l'aide d' Eclipse.

     Vous avez le choix entre plusieurs méthodes pour obtenir le WSDL du service. Dans cet exemple, le service a été déployé dans WebSphere Application Server Community Edition et vous obtenez le WSDL du serveur Web. Le déploiement est décrit dans la tâche «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992,

a. Dans Eclipse, passez à la perspective Web et vérifiez que le serveur WebSphere Application Server Community Edition v2.1 Server est en cours d'exécution et que l'axe StockQuoteest déployé et synchronisé.

b. Importez le fichier WSDL dans l'explorateur de services Web:

   i) Cliquez sur l'icône **Explorateur de services Web** dans la barre d'actions ou cliquez sur **Exécuter** > **Lancer l'explorateur de services Web**.

   ii) Cliquez sur l'icône de page WSDL dans l'explorateur de services Web pour passer à la page WSDL.

   iii) Cliquez sur **WSDL principal** dans la fenêtre Navigator de l'explorateur de services Web.

   iv) Entrez l'URL du service Web, suivie de `?WSDL` . L'URL de l'axe StockQuote, déployée dans la tâche «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992, est:

   ```
   http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl
   ```

c. Générez les proxys client:

   i) Dans le navigateur de l'explorateur de services Web, cliquez sur **http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl** .

   ii) Dans la fenêtre **Actions** , cliquez sur **Lancer l'assistant de service Web** > laisser **Client de service Web** sélectionné > **Aller**.

   iii) Sur la première page de l'assistant, cliquez sur le lien du projet **Client** dans la configuration > Sélectionnez le projet client **StockQuoteAxisClient** > **OK**.

   Conseil : La fenêtre de l'assistant risque de perdre la mise en évidence. Vous devez le remettre en évidence manuellement.

   iv) L'environnement d'exécution du service Web doit être Apache Axis pour générer un client JAX-RPC.

   v) Cliquez sur **Terminer**.

   vi) Modifiez l'URL statique du service pour qu'elle pointe vers le transport WebSphere MQ pour l'adresse SOAP du service StockQuoteAxis. Vous pouvez choisir d'ignorer cette étape jusqu'à ce que vous ayez testé le client avec un serveur HTTP.

      a) Ouvrez `StockQuoteAxisServiceLocator.java` et recherchez la déclaration pour `StockQuoteAxis_address`.

      b) Remplacez l'URL par

      ```
      "jms:/queue?destination=REQUESTAXIS
      &amp;initialContextFactory=com.ibm.mq.jms.Nojndi
      &amp;connectionFactory=(connectQueueManager(QM1)binding(auto))"
      ```

      Conseil : Eclipse transforme automatiquement & en `&`, et inversement, lorsque vous copiez et collez des chaînes dans du code `.java` .

d. Créez trois classes client Java, chacune avec une méthode principale:

   i) Créer un package. Cliquez avec le bouton droit de la souris sur **StockQuoteAxisClient/src** > **Nouveau package**. Nommez-le `soap.client` > **Terminer**.

   ii) Sélectionnez **soap.client** > **Nouveau** > **Classe**. Nommez la classe `SQAStaticClient` > Check **public static void main (string [ ] args)** > Finish

   iii) Répétez la procédure pour créer `SQADynamicClient.java` et `SQADIIClient.java`

e. Ecrivez le code client.

   Figure 177, à la page 1011 à Figure 181, à la page 1013 fournissent des exemples des trois styles de code client. Les exemples utilisent une URL HTTP pour tester le client à l'aide du

service Axis StockQuotedéployé sur un serveur HTTP. Pour exécuter les clients sur le service StockQuoteAxis déployé à l'aide du transport WebSphere MQ pour SOAP, remplacez l'URL par:

```
"jms:/queue?destination=REQUESTAXIS
connectionFactory=(connectQueueManager(QM1)binding(auto))
initialContextFactory=com.ibm.mq.jms.Nojndi
targetService=soap.server.StockQuoteAxis.java
replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
```

- Figure 177, à la page 1011 et Figure 179, à la page 1012 utilisent le proxy généré par Eclipse, qui dispose de la classe auxiliaire Axisproxy StockQuotesupplémentaire qui facilite un peu le codage.
- Figure 178, à la page 1012 et Figure 180, à la page 1012 utilisent le proxy généré par **amqwdeployWMQService** .
- Figure 181, à la page 1013 n'utilise aucune classe de proxy.

Chacun des clients appelle `com.ibm.mq.soap.Register.extension()` pour établir une liaison avec le transport WebSphere MQ pour SOAP. L'extension est enregistrée dans le descripteur de déploiement client. Le déploiement du client sur Axis 1.4 est décrit dans «Déploiement d'un client de service Web sur Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP», à la page 1043.

f. Exécutez les clients en envoyant la demande SOAP à l'axe StockQuotehébergé par le serveur WebSphere Application Server Community Edition configuré dans l'espace de travail.

   i) Vérifiez que le serveur est en cours d'exécution, que l'axe StockQuoteest déployé et synchronisé.

   ii) Sélectionnez ou ouvrez le client que vous souhaitez tester > Cliquez sur **Exécuter** dans la barre d'actions. Vous pouvez également cliquer sur l'icône d'exécution verte ou sur le client dans le navigateur > **Exécuter en tant que** > **Configurations d'exécution ...**. Configurez les paramètres dont vous avez besoin pour exécuter le client.

g. Exécutez le client à l'aide du transport WebSphere MQ pour SOAP.

   La procédure utilise **amqwdeployWMQService** pour déployer le service et ne fonctionne qu'avec le client qui utilise le WSDL ou les proxys générés par **amqwdeployWMQService** . Pour exécuter le client à l'aide du WSDL d'origine ou des proxys générés par eclipse, déployez le service avec son descripteur de déploiement généré par Eclipse. Démarrez manuellement **SimpleJavaListener** en utilisant le nom de liaison de port de service comme `targetServiceNom` .

   i) Suivez les instructions de la rubrique «Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService», à la page 1032 pour déployer le service sur le programme d'écoute SOAP Java simple WebSphere MQ . Le déploiement de service fonctionne uniquement pour le client à l'aide du WSDL ou des proxys client générés par **amqwdeployWMQService**.

   ii) Dans une fenêtre de commande, exécutez **amqwclientconfig** pour créer le fichier de descripteur de déploiement client, `client-deploy.wsdd` .

   iii) Importez `client-deploy.wsdd` dans la racine du projet Java que vous souhaitez tester à l'aide du transport WebSphere MQ pour SOAP.

      a) Cliquez avec le bouton droit de la souris sur le projet Java **StockQuoteAxisEclipseClient** > **Importer** > **Système de fichiers** > **Suivant** > **Parcourir ...**

      b) Accédez au répertoire contenant `client-deploy.wsdd` > **Ouvrir** > Sélectionnez le répertoire dans la page de l'assistant **Importation** > cochez `client-deploy.wsdd` dans le panneau de droite.

      c) Vérifiez que **Dans le dossier:** a `StockQuoteAxisEclipseClient` entré > **Terminer**.

   iv) Vérifiez que le répertoire de travail pour l'exécution d'une application Java dans ce projet est le répertoire `StockQuoteAxisEclipseClient` :

Cliquez avec le bouton droit de la souris sur le projet Java **StockQuoteAxisEclipseClient** > **Exécuter en tant que ....** > **Configurations d'exécution ...** > Sélectionnez l'onglet **(x) = Arguments** > Vérifiez que dans le répertoire de travail, le bouton d'option **Par défaut** est activé et que le chemin d'accès est `StockQuoteAxisEclipseClient` . Vous pouvez également choisir l'un des choix suivants pour sélectionner un autre emplacement ou fichier contenant la configuration du client:

- Cochez la case **Autre:** > entrez le chemin de répertoire de votre choix.
- Dans la fenêtre **VM arguments** , entrez `-Daxis.ClientConfigFile=`*`full path to client deployment descriptor file`*

v) Assurez-vous que l'URL est configurée pour pointer vers le service déployé à l'aide du transport WebSphere MQ pour SOAP. Exécutez le client comme décrit à l'étape ii.

Conseil : Généralement, vous pouvez rencontrer l'une des erreurs suivantes:

i) `Exception: No client transport named 'jms' found!` .

ii) Erreur de connexion JMS.

iii) `Exception: The AXIS engine could not find a target service to invoke! targetService is soap.server.StockQuoteAxis.java`

iv) `Exception: java.lang.InstantiationException: soap.server.StockQuoteAxis`

Explications:

i) `client-config.wsdd` est introuvable ou inclut la ligne `<transport name="jms" pivot="java:com.ibm.mq.soap.transport.jms.WMQSender"/>` dans `client-config.wsdd`.

ii) Il peut s'agir d'un problème de chemin de génération-sans inclure les fichiers .jar dans *`MQ_INSTALLATION_PATH`*`/java/lib` . *`MQ_INSTALLATION_PATH`* représente le répertoire dans lequel WebSphere MQ est installé.

iii) Problème de déploiement de service, avec `server-config-wsdd` ou avec des paramètres transmis à **`SimpleSoapListener`** .

iv) Non-concordance entre le descripteur de déploiement et l'implémentation du service.

Si vous rencontrez des difficultés lors de l'exécution du client dans Eclipse, essayez d'utiliser une fenêtre de commande:

i) Accédez au répertoire `StockQuoteAxisEclipseClient\bin` dans l'arborescence des répertoires de l'espace de travail.

ii) Exécutez **amqwsetcp** et **amqwclientconfig**

iii) Exécutez `java soap/client/SQAStaticClient`.

Exemples de clients de service Web JAX-RPC

Les exemples de clients de service Web Java fournis avec WebSphere MQ sont installés dans *`MQ_INSTALLATION_PATH`* `\tools\soap\samples\java\clients`. *`MQ_INSTALLATION_PATH`* est le répertoire dans lequel WebSphere MQ est installé.

`SQAxis2Axis.java`
`SQAxis2Axis.java`, Figure 174, à la page 1010 , est un client de proxy dynamique pour appeler le service `StockQuoteAxis` . Vous pouvez remplacer l'URL du service, qui est compilée dans le proxy dynamique, en fournissant une URL sur la ligne de commande.

`SQAxis2DotNet.java`
`SQAxis2DotNet.java`, Figure 175, à la page 1010, est un client de proxy dynamique pour appeler le service `StockQuoteDotNet` . Vous pouvez remplacer l'URL du service, qui est compilée dans le proxy dynamique, en fournissant une URL sur la ligne de commande.

WsdlClient.java

WsdlClient.java, Figure 176, à la page 1011, est un client d'appel dynamique pour appeler le service StockQuoteDotNet ou StockQuoteAxis . Le client appelle le service StockQuoteAxis par défaut. Ajoutez l'option de ligne de commande -D pour appeler le service StockQuoteDotNet et -w pour fournir un port différent de celui de .\generated\StockQuoteDotNet_Wmq.wsdl

```
package soap.clients;
import java.net.URL;
import soap.server.*;
public class SQAxis2Axis {
   public static void main(String[] args) {
      com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
         StockQuoteAxis service = null;
         if (args.length == 0)
            service = locator.getSoapServerStockQuoteAxis_Wmq();
         else
            service = locator.getSoapServerStockQuoteAxis_Wmq(
                new java.net.URL(args[0]));
         System.out.println("Response: " + service.getQuote("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
```

Figure 174. SQAxis2Axis.java

```
public class SQAxis2DotNet {
public static void main(String[] args) {
   com.ibm.mq.soap.Register.extension();
      try {
         StockQuoteDotNet locator = new StockQuoteDotNetLocator();
         StockQuoteDotNetSoap_PortType service = null;
         if (args.length == 0)
            service = locator.getStockQuoteDotNetSoap();
         else
            service = locator.getStockQuoteDotNetSoap(new java.net.URL(
                  args[0]));
         System.out.println("Response: " + service.getQuoteDOC("XXX"));
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING ProxyClient DEMO <<<\n");
                           e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figure 175. SQAxis2DotNet.java

```
package soap.clients;
import com.ibm.mq.soap.*;
import org.apache.axis.utils.Options;
import java.net.URL;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
import javax.xml.namespace.QName;
public class WsdlClient {
public static void main(String[] args) {
      String wsdlService, wsdlPort, namespace, wsdlSource, wsdlTargetURI, s;
      try {
         Register.extension();
         Options opts = new Options(args);
         if (opts.isFlagSet('D') != 0) {
            wsdlService = "StockQuoteDotNet";
            wsdlPort = "StockQuoteDotNetSoap";
            namespace = "http://stock.samples";
            wsdlSource = "file:generated/StockQuoteDotNet_Wmq.wsdl";
         } else {
            wsdlService = "StockQuoteAxisService";
            wsdlPort = "soap.server.StockQuoteAxis_Wmq";
            namespace = "soap.server.StockQuoteAxis_Wmq";
            wsdlSource = "file:generated/soap.server.StockQuoteAxis_Wmq.wsdl";
         }
         if (null != (s = (opts.isValueSet('w'))))
            wsdlPort = s;
         System.out.println("start WsdlClient demo, wsdl port " + wsdlPort
               + " resolving uri to ...");
         QName servQN = new QName(namespace, wsdlService);
         QName portQN = new QName(namespace, wsdlPort);
         Service service = ServiceFactory.newInstance().createService(
               new URL(wsdlSource), servQN);
         Call call = (Call) service.createCall(portQN, "getQuote");
         wsdlTargetURI = call.getTargetEndpointAddress().toString();
         System.out.println("  '" + wsdlTargetURI + "'");
         Object ret = call.invoke(new Object[] { "XXX" });
         System.out.println("Response: " + ret);
      } catch (Exception e) {
         System.out.println("\n>>> EXCEPTION WHILE RUNNING WsdlClient DEMO <<<\n");
         e.printStackTrace();
         System.exit(2);
      }
   }
}
```

Figure 176. WsdlClient.java

Exemples de clients utilisés dans cette tâche:

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy();
      System.out.println("Static client synchronous result is:"
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figure 177. Client statique utilisant le proxy généré par Eclipse

```
package soap.client;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQAStaticClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq();
      System.out.println("Static client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figure 178. Client statique utilisant le proxy généré par amqwdeployWMQService

```
package soap.client;
import soap.server.StockQuoteAxisProxy;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      StockQuoteAxisProxy sqa = new StockQuoteAxisProxy(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figure 179. Client dynamique utilisant le proxy généré par Eclipse

```
package soap.client;

import java.net.URL;
import soap.server.StockQuoteAxis;
import soap.server.StockQuoteAxisService;
import soap.server.StockQuoteAxisServiceLocator;
public class SQADynamicClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL sqaURL = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      StockQuoteAxisService locator = new StockQuoteAxisServiceLocator();
      StockQuoteAxis sqa = locator.getSoapServerStockQuoteAxis_Wmq(sqaURL);
      System.out.println("Dynamic client synchronous result is: "
          + sqa.getQuote("ibm"));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figure 180. Client dynamique utilisant le proxy généré par amqwdeployWMQService

```
package soap.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
public class SQADIIClient {
  public static void main(String[] args) {
    try {
      com.ibm.mq.soap.Register.extension();
      URL wsdl = new URL(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl");
      Service SQAService = (ServiceFactory.newInstance()).createService(wsdl,
          new QName("http://server.soap", "StockQuoteAxisService"));
      Call SQACall = SQAService.createCall(new QName("http://server.soap",
          "StockQuoteAxis"), "getQuote");
      System.out.println("DII client synchronous result is "
          + SQACall.invoke(new Object[] { "ibm" }));
    } catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```

Figure 181. Client DII (aucun proxy)

Tâches associées

Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse
Développer un client de service Web Axis2 à exécuter à l'aide de WebSphere MQ transport for SOAP. Les clients Axis2 exemples fournis avec WebSphere MQ transport for SOAP sont répertoriés, et la commande **wsimport** est utilisée pour générer des proxys.

Développement d'un client .NET 1 ou 2 pour le transport WebSphere pour SOAP à l'aide de Microsoft Visual Studio 2008
Développez un client de service Web .NET 1 ou 2 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse

Développer un client de service Web Axis2 à exécuter à l'aide de WebSphere MQ transport for SOAP. Les clients Axis2 exemples fournis avec WebSphere MQ transport for SOAP sont répertoriés, et la commande **wsimport** est utilisée pour générer des proxys.

Avant de commencer

Récupérez les bibliothèques Axis2, puis configurez un environnement de développement et de test pour exécuter le client.

Remarque : La désignation des versions et des éditions utilisées par Axis entraîne une confusion. En général, Axis 1.4 fait référence à l'implémentation JAX-RPC, et Axis2 à l'implémentation JAX-WS.

Axis 1.4 correspond à un niveau de version. Si vous faites une recherche sur Axis 1.4 sur Internet, vous êtes dirigé vers la page http://ws.apache.org/axis/. Celle-ci contient la liste des versions précédentes d'Axis (1.2, 1.3) et la publication finale d'Axis 1.4 du 22 avril 2006. Il existe des éditions plus récentes d'Axis 1.4, qui corrigent les bogues, mais elles portent toutes le nom d'Axis 1.4. L'édition fournie avec WebSphere MQ est l'une des éditions qui corrigent les bogues. Pour Axis 1.4, utilisez la version de `axis.jar` qui est fournie avec WebSphere MQ, plutôt que celle qui est proposée sur la page http://ws.apache.org/axis/.

Le site Web d'Axis indique également Axis 1.1 pour faire référence à toutes les versions de ce qui est généralement appelé Axis 1.4. Axis 1.2 fait référence à Axis2.

Axis 1.5 n'est pas une version ultérieure d'Axis 1.4, mais une édition d'Axis2. Si vous recherchez Axis 1.5, vous êtes dirigé vers la page http://ws.apache.org/axis2/.https://ws.apache.org/axis2/download.cgi Contient une liste de versions d'édition de Axis2, portant la mention 0.9 à 1.5.1 (et notamment la version

1.4). La version publiée d'Axis2 à utiliser avec WebSphere MQ transport for SOAP est la version 1.4.1. Téléchargez Axis2 1.4.1 à partir de la pagehttp://ws.apache.org/axis2/download/1_4_1/download.cgi.

Vous avez la possibilité de générer des proxys pour les clients de services Web de WebSphere MQ transport for SOAP à l'aide de la commande **wsimport** ou des outils fournis avec un environnement de développement intégré. Eclipse IDE for Java EE Developer 3.5 SR1 utilise **wsdl2java**.**wsimport** est fourni avec Java 6. Vous pouvez utiliser Java 5 pour exécuter des proxys client générés avec **wsimport** ou **wsdl2java**.

Les clients Axis2 exemples de service Web fournis avec WebSphere MQ transport for SOAP ont été développés à l'aide de la commande **wsimport**. Voir «Clients Axis2 exemples», à la page 1019.

La tâche qui suit explique comment générer et utiliser les proxys produits par l'assistant de services Web fourni avec Eclipse IDE for Java EE Developers. Les clients exemples montrent comment utiliser les proxys créés à l'aide de la commande **wsimport**.

Pour utiliser l'assistant de services Web, vous devez ajouter un serveur d'applications qui prend en charge Axis2 au plan de travail. Ces étapes indiquent comment configurer WASCE pour la prise en charge d'Axis2 à l'aide du plan de travail.

1. Configurez le serveur d'applications utilisé dans Eclipse IDE for Java EE Developers pour prendre en charge Axis2. Dans cet exemple configurez WASCE 2.1.4, le serveur d'application qui fait partie de l'espace de travail créé dans «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992.
   a. Ouvrez les préférences de l'espace de travail pour configurer le serveur: ouvrez **Fenêtre** > **Préférences**.
   b. Vérifiez que l'environnement d'exécution Java installé est bien `Java50`. Cliquez sur **Installed JREs**.
   c. Ajoutez WASCE en tant que serveur: cliquez sur **Serveur** > **Environnements d'exécution** > **Ajouter ...** > **IBM** > **WASCE v2.1** > Suivant. L'environnement d'exécution Java doit être Java50 > Accédez au répertoire d'installation WASCE > **OK** > **Terminer**. Vous devez avoir installé le plug-in WASCE pour Eclipse Java EE IDE for Web Developers.
   d. Ajoutez Axis2: Cliquez sur **Services Web** > **Axis2 Préférences**. Dans l'onglet **Axis2 Runtime** > **Browse ...** Ouvrez le répertoire contenant les nombreux fichiers JAR `Axis2` > **Appliquer**.
   e. Associez WASCE à Axis2: Cliquez sur **Web Services** > **Server and Runtime**. Sous **`Server`** , sélectionnez **IBM WASCE v2.1 Server**et sous **`Web service runtime`**, sélectionnez **Apache Axis2** > **Apply** > **OK** .
   f. Démarrez le serveur : Ouvrez la perspective Web et la vue Servers. Cliquez avec le bouton droit de la souris dans la vue Serveurs > **Nouveau** > **Serveur**. **IBM WASCE v2.1 Server** est sélectionné et configuré > **Terminer**. Démarrer le serveur.
2. Vérifiez que vous avez déployé le service StockQuoteAxis sur WASCE afin d'exécuter l'assistant de services Web.
3. Pour tester ce service avec le service WebSphere MQ transport for SOAP, déployez le service sur WebSphere MQ transport for SOAP listener for Axis 1.4. Voir «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992.

Pourquoi et quand exécuter cette tâche

Les développeurs Eclipse IDE for Java EE utilisent Java50 et l'assistant de services Web pour générer les classes de proxy du service. Les classes de proxy sont différentes des classes créées par l'outil **wsimport** fourni avec Java 6. Une autre approche consiste à générer les classes de proxy à l'aide de **wsimport** et à importer les packages qu'il crée dans votre environnement de développement intégré Eclipse Java EE pour les développeurs Web.

L'assistant de services Web dans l' Eclipse IDE pour les développeurs Java EE génère un client de service Web dans un projet Web. Vous pouvez exécuter le client en tant qu'application Java simple ; il ne nécessite pas de serveur d'applications. Vous pouvez également transférer le code vers un projet Java et configurer le chemin de génération pour inclure les fichiers JAR Axis2 .

Procédure

1. Créez un projet Web dans un nouveau projet Enterprise :
   a) Lorsque rien n'est sélectionné dans l'explorateur de projets > Cliquez avec le bouton droit de la souris sur l'espace blanc > **Nouveau** > **Projet d'application d'entreprise** > Nommez-le `StockQuoteAxis2EAR` > **Terminer**. Répondez No à la fenêtre vous offrant la possibilité d'ouvrir la perspective Java EE .

      Les paramètres par défaut utilisent WASCE.

   b) Cliquez avec le bouton droit de la souris sur `StockQuoteAxis2EAR` > **Nouveau** > **Projet Web dynamique**. Nommez le projet `StockQuoteAxis2WebClient` > Cochez la case d'appartenance EAR pour ajouter le projet à **StockQuoteAxis2EAR**. WASCE 2.1 est sélectionné en tant qu'exécution cible.

   c) Dans la section Configuration de la page **Nouveau projet Web dynamique** > **Modifier ...** > Vérifiez la facette du projet de services Web Axis2 . **Dynamic Web Module 2.5**, **Java 6.0**et **WASCE deployment 1.2** sont déjà vérifiés. > **OK** > **Terminer**. Répondez No à la fenêtre vous offrant la possibilité d'ouvrir la perspective Java EE .
2. Importez WSDL pour le service dans l'espace de travail et créez le proxy client :

   Dans cet exemple, le document WSDL contient la liaison de service HTTP et devient la cible du proxy client Web statique. Vous avez la possibilité de modifier l'URL dans la liaison de service Web afin qu'elle pointe vers l'URL de WebSphere MQ transport for SOAP avant la création du proxy client. Le proxy client Web statique est alors le service qui est déployé sur WebSphere MQ transport for SOAP.

   a) Lancez l'explorateur de services Web: utilisez l'icône de la barre d'actions ou **Exécutez** > **Lancez l'explorateur de services Web**.

   b) Sélectionnez l'explorateur WSDL en cliquant sur l'icône WSDL dans la fenêtre **Explorateur de services Web** > Cliquez sur **WSDL principal** dans la fenêtre Navigator > Entrez l'URL du fichier WSDL StockQuoteAxis > **Go**.
   Dans cet exemple, récupérez WSDL directement à partir du service HTTP : `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis?wsdl`

   c) Dans le navigateur, cliquez sur la ligne comportant l'URL du service Web. Dans la fenêtre **Actions** , cliquez sur **Import WSDL to Workbench** > Select a **StockQuoteAxis2WebClient** as **Workbench project** > Entrez le **nom du fichier WSDL**, `StockQuoteAxisHTTP.wsdl` > **Go**.

   d) Cliquez avec le bouton droit de la souris sur **StockQuoteAxisHTTP.wsdl** > **Services Web** > **Générer un client**. Vérifiez les informations de configuration concernant la page de services Web de l'assistant, comme suit : Server: IBM WASCE v2.1 Server, Web service runtime: Apache Axis2, Client project: StockQuoteAxis2WebClient, Client EAR project: StockQuoteAxisEAR. Pour modifier la configuration, cliquez sur les lignes qui ne vous conviennent pas.

   e) Cliquez sur **Suivant** > vérifiez les paramètres de génération de code > **Terminer**.

      Notez qu'un nouveau package, `soap.server`, est créé. Il contient les proxys requis.
3. Configurez le projet afin qu'il exécute WebSphere MQ transport for SOAP en tant que transport JMS.

   WebSphere MQ transport for SOAP fournit `transportSender`, mais pas `transportReceiver`. En d'autres termes, WebSphere MQ transport for SOAP prend en charge les clients Axis2. Il ne prend pas encore en charge les services Axis2.

   a) Dans le projet **StockQuoteAxis2WebClient** , cliquez avec le bouton droit de la souris sur `WebContent\WEB-INF\conf\axis2.xml` > **Ouvrir avec ...** > **Editeur XML**.

   b) Recherchez le dernier `transportSender` (vers la fin du fichier) et recherchez le JMS `transportSender` > cliquez avec le bouton droit de la souris sur la ligne > **Ajouter avant ...** > **transportSender**.

   c) Cliquez avec le bouton droit de la souris sur **transportSender** > **Ajouter un attribut** > **Nom** > Cliquez avec le bouton droit de la souris sur **transportSender** > **Ajouter un attribut** > **Classe**.

   d) Cliquez avec le bouton droit de la souris sur **Nom** > **Editer l'attribut** > Entrez la valeur **:** `jms`

   e) Cliquez avec le bouton droit de la souris sur **Classe** > **Editer l'attribut** > Entrez la **valeur:** `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender`. > Enregistrer.

f) Ajoutez `com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender` au chemin de génération: cliquez avec le bouton droit de la souris sur **StockQuoteAxis2WebClient** > **Chemin de génération** > **Configurer le chemin de génération ...** > Cliquez sur l'onglet **Bibliothèques** > **Ajouter des fichiers JAR externes ...**. Sélectionnez tous les fichiers JAR dans *MQ_INSTALLATION_PATH*`\java\lib` > **OK**.

MQ_INSTALLATION_PATH représente le répertoire dans lequel WebSphere MQ est installé.

4. Créez un client statique synchrone, testez-le à l'aide de HTTP, puis convertissez le proxy afin qu'il exécute ce client statique à l'aide de WebSphere MQ transport for SOAP.

   a) Cliquez avec le bouton droit de la souris sur **Java Resources: src** > **Nouveau** > **Package** > Nommez le package `soap.client` > Terminer

   b) Cliquez avec le bouton droit de la souris sur **soap.client** > **Nouveau** > **Classe** > Nommez la classe `SQA2StaticClient` > **Terminer**.

   c) Remplacez la classe par le code suivant, puis cliquez sur **Sauvegarder**.

   Figure 182. SQA2DynamicClient.java

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2StaticClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub();
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("Response is: "
          + (stub.getQuote(request)).getGetQuoteReturn());
    } catch (Exception e) {
      System.out.println("Exception: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

5. Testez le client avec le service StockQuoteAxis déployé sur WASCE et avec WebSphere MQ transport for SOAP.

   a) Dans l'explorateur de projets, cliquez avec le bouton droit de la souris sur **SQA2StaticClient** > **Exécuter en tant que ...** > **Application Java**.

   Le résultat, `Response is 55.25`, apparaît dans la vue Console. Vous pouvez également sélectionner la fenêtre de la console WASCE dans la vue Console et voir la sortie sur le serveur WASCE, `StockQuoteAxis called with parameter: ibm`.

   b) Le proxy a été créé avec l'adresse du service, `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis`, le client statique appelle donc le service exécuté sur HTTP. Vous pouvez modifier le client statique pour appeler le service à l'aide de WebSphere MQ transport for SOAP. Les instructions suivantes permettent de modifier l'adresse du service dans `StockQuoteAxisServiceStub.java` sans devoir recréer le proxy, et de configurer les paramètres d'exécution `SQA2StaticClient` pour charger `axis2.xml`. `axis2.xml` configure Axis2 pour utiliser WebSphere MQ transport for SOAP.

   c) Ouvrez `StockQuoteAxisServiceStub.java` > Remplacez les deux occurrences de `http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis` par

```
jms:/queue?destination=REQUESTAXIS@QM1
&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteAxis
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

   d) Si vous exécutez maintenant `SQA2StaticClient`, il renvoie un exception car il ne trouve pas de `transportSender` configuré pour JMS

L'exception est :

```
Exception: null java.lang.NullPointerException at
soap.server.StockQuoteAxisServiceStub.getQuote(StockQuoteAxisServiceStub.java:547)
at soap.client.SQA2StaticClient.main(SQA2StaticClient.java:11)
```

e) Dans l'explorateur de projets, cliquez avec le bouton droit de la souris sur **SQA2StaticClient** > **Exécuter en tant que ...** > **Configurations d'exécution ...**. Accédez à l'onglet **(x) = Arguments** et, dans la zone d'entrée **Arguments VM** , entrez le chemin d'accès au fichier `axis2.conf` > **Appliquer** > **Exécuter**.
L'argument VM est le suivant : `-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml`. Vous pouvez indiquer un chemin standard vers le fichier de configuration d'Axis2.

f) Exécutez à nouveau `SQA2StaticClient`. Lors de cette exécution, vous utilisez WebSphere MQ transport for SOAP. Vérifiez qu'aucune nouvelle sortie ne figure dans la console WASCE. Ouvrez la console ou la fenêtre de commande associée au programme d'écoute SimpleJavaet la sortie est `StockQuoteAxis called with parameter: ibm`.

6. Créez un client dynamique pour HTTP et WebSphere MQ transport for SOAP, puis testez-le.

a) Cliquez avec le bouton droit de la souris sur **soap.client** > **Nouveau** > **Classe** > Nommez la classe `SQA2DynamicClient` > **Terminer**.

b) Remplacez la classe par le code suivant, puis cliquez sur **Sauvegarder**.

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
public class SQA2DynamicClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
       System.out.println("JMS sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
     } catch (Exception e) {
       System.out.println("Exception: " + e.getMessage());
       e.printStackTrace();
     }
   }
 }
```

c) Créez une configuration d'exécution pour `SQA2DynamicClient.java`, puis ajoutez le chemin à `axis2.xml`:
`-Daxis2.xml=${workspace_loc:StockQuoteAxis2WebClient/WebContent/WEB-INF/conf}/axis2.xml`

d) Exécutez `SQA2DynamicClient`. Recherchez dans la sortie de la console `SQA2DynamicClient`, WASCE et **`SimpleJavaListener`**.

7. Créez un client asynchrone, puis accédez au résultat dans un gestionnaire d'appel et dans l'unité d'exécution principale du programme.

Les proxys client asynchrones créés par l'assistant de service Web pour Eclipse Java EE IDE for Web Developers diffèrent des proxys créés par **`wsimport`**. Les proxys **`wsimport`** utilisent les types génériques `Future`, `Response` et `AsyncHandler`.

L'assistant de service Web pour Eclipse Java EE IDE for Web Developers crée une classe abstraite `StockQuoteAxisServiceCallbackHandler` . Vous devez étendre `StockQuoteAxisServiceCallbackHandler` et créer un gestionnaire d'appel.

a) Cliquez avec le bouton droit de la souris sur **soap.client** > **Nouveau** > **Classe** > Nommez la classe SQA2CallbackHandler > **Terminer**.

b) Remplacez la classe par le code suivant.

```
package soap.client;
import soap.server.StockQuoteAxisServiceCallbackHandler;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
public class SQA2CallbackHandler
    extends StockQuoteAxisServiceCallbackHandler {
  private boolean complete = false;
  SQA2CallbackHandler() {
    super();
    System.out.println("Callback constructor");
  }
  public void receiveResultgetQuote(GetQuoteResponse response) {
    System.out.println("Result in Callback " + response.getGetQuoteReturn());
    super.clientData = response;
    complete = true;
  }
  public boolean isComplete() {
    return complete;
  }
}
```

c) Cliquez avec le bouton droit de la souris sur **soap.client** > **Nouveau** > **Classe** > Nommez la classe SQA2AsyncClient > **Terminer**.

d) Remplacez la classe par le code suivant.

Figure 183. SQA2AsyncClient.java

```
package soap.client;
import soap.server.StockQuoteAxisServiceStub;
import soap.server.StockQuoteAxisServiceStub.GetQuote;
import soap.server.StockQuoteAxisServiceStub.GetQuoteResponse;
import soap.server.StockQuoteAxisServiceCallbackHandler;
@SuppressWarnings("unused")
public class SQA2AsyncClient {
  public static void main(String[] args) {
    try {
      StockQuoteAxisServiceStub stub = new StockQuoteAxisServiceStub(
          "http://localhost:8080/StockQuoteAxis/services/StockQuoteAxis");
      GetQuote request = new GetQuote();
      request.setSymbol("ibm");
      System.out.println("HTTP Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      SQA2CallbackHandler callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      do {
        System.out.println("Waiting for HTTP callback");
        Thread.sleep(2000);
      } while (!callback.isComplete());
      System.out.println("HTTP poll: "
          + ((GetQuoteResponse) (callback.getClientData()))
                .getGetQuoteReturn());
      stub = new StockQuoteAxisServiceStub(
          "jms:/queue?destination=REQUESTAXIS@QM1"
          + "&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&targetService=StockQuoteAxis&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE");
      System.out.println("JMS Sync: "
          + (stub.getQuote(request)).getGetQuoteReturn());
      callback = new SQA2CallbackHandler();
      stub.startgetQuote(request, callback);
      while (!callback.isComplete()) {
        System.out.println("Waiting for JMS callback");
        Thread.sleep(2000);
      }
      System.out.println("JMS poll: "
        + ((GetQuoteResponse) (callback.getClientData())).getGetQuoteReturn());
    } catch (Exception e) {
        System.out.println("Exception: " + e.getMessage());
        e.printStackTrace();
     }
   }
 }
```

La sortie de la console est la suivante:

```
HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
```

Clients Axis2 exemples

Les exemples de proxy sont générés à l'aide de l'outil **wsimport** fourni avec Java 6. Six échantillons sont fournis:

1. DynamicProxyClientSync.java
2. DynamicProxyClientAsyncPolling.java
3. DynamicProxyClientAsyncCallback.java
4. DispatchClientSync.java
5. DispatchClientAsyncPolling.java
6. DispatchClientAsyncCallback.java

Les exemples de clients sont générés pour l'exemple de serveur StockQuoteAxis. Générez WSDL avec la commande **amqwdepoyWMQServer**, en indiquant le commutateur -w pour sélectionner le style `rpcLiteral`. Utilisez la commande suivante pour générer les proxys des exemples :

```
wsimport soap.server.StockQuoteAxis_Wmq.wsdl -d generated -keep -p com.ibm.mq.axis2.samples
```

Figure 184. DynamicProxyClientSync.java

```
package com.ibm.mq.axis2.samples;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientSync");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      service.getQuoteOneWay("48");
      System.out.println(" > getQuoteOneWay has returned");

      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      float result = service.getQuote("48");
      System.out.println(" > getQuote has returned result of " + result);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());
```

```
        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figure 185. DynamicProxyClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.CancellationException;

import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncPolling");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously by
polling...");
      Response<Float> response = service.getQuoteAsync("49");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** Retrieve the result **/
      try {
        Float result = response.get();
        System.out.println(" > getQuoteAsync call has returned result of " + result);
      }
      catch (CancellationException ce) {
        // processing was cancelled via response.cancel()
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figure 186. DynamicProxyClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Response;

import com.ibm.mq.axis2.samples.proxy.StockQuoteAxis;
import com.ibm.mq.axis2.samples.proxy.StockQuoteAxisService;

public class DynamicProxyClientAsyncCallback implements AsyncHandler<Float> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DynamicProxyClientAsyncCallback");

      System.out.println("Creating proxy instance for service StockQuoteAxisService");
      StockQuoteAxisService stub = new StockQuoteAxisService();
      StockQuoteAxis service = stub.getSoapServerStockQuoteAxisWmq();

      DynamicProxyClientAsyncCallback handler = new DynamicProxyClientAsyncCallback();

      System.out
          .println("Invoking getQuoteAsync Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = service.getQuoteAsync("50", handler);
      System.out.println(" > Invoke call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<Float> response) {
    try {
      Float result = response.get();
      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
```

```
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Figure 187. DispatchClientSync.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientSync {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientSync");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service **/
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /************************************
       * Create OneWay SOAPMessage request.
       ************************************/
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a OneWay SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuoteOneWay", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuoteOneWay OneWay operation synchronously...");
      dispatch.invokeOneWay(request);
```

```
      System.out.println(" > getQuoteOneWay call has returned");

      /*****************************************
       * Create Request Reply SOAPMessage request.
       *****************************************/
      mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("\nCreating a Request Reply SOAP Message");
      request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      part = request.getSOAPPart();
      env = part.getEnvelope();
      header = env.getHeader();
      body = env.getBody();

      /** Construct the message payload **/
      operation = body.addChildElement("getQuote", "ns1", "soap.server.StockQuoteAxis_Wmq");
      value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation synchronously...");
      SOAPMessage ans = dispatch.invoke(request);
      System.out.println(" > getQuote call has returned");

      /** Retrieve the result **/
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      System.out.println("Parsing SOAP response...");
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figure 188. DispatchClientAsyncPolling.java

```
package com.ibm.mq.axis2.samples;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
```

```
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncPolling {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncPolling");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);

      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint **/
      System.out.println("Invoking getQuote Request Reply operation asynchronously by
polling...");
      Response<SOAPMessage> response = dispatch.invokeAsync(request);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until response arrives **/
      System.out.println("Waiting for response to arrive...");
      while (!response.isDone()) {
        Thread.sleep(100);
      }
      System.out.println(" > Response received");

      /** retrieve the result **/
      SOAPMessage ans = response.get();
      part = ans.getSOAPPart();
      env = part.getEnvelope();
      body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
```

```
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String message = responseElement.getValue();
      System.out.println(" > Response contains result of " + message);

      System.out.println("End of sample");

    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }
}
```

Figure 189. DispatchClientAsyncCallback.java

```
package com.ibm.mq.axis2.samples;

import java.util.concurrent.Future;

import javax.xml.namespace.QName;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPConstants;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.ws.AsyncHandler;
import javax.xml.ws.Dispatch;
import javax.xml.ws.Response;
import javax.xml.ws.Service;
import javax.xml.ws.soap.SOAPBinding;

public class DispatchClientAsyncCallback implements AsyncHandler<SOAPMessage> {

  public static void main(String[] args) {
    try {
      System.out.println("Starting sample DispatchClientAsyncCallback");

      String endpointUrl = "jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM&"
          + "connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)"
          +
"&initialContextFactory=com.ibm.mq.jms.Nojndi&targetService=soap.server.StockQuoteAxis.java";

      QName serviceName = new QName("soap.server.StockQuoteAxis_Wmq", "StockQuoteAxisService");
      QName portName = new QName("soap.server.StockQuoteAxis_Wmq",
"soap.server.StockQuoteAxis_Wmq");

      Service service = Service.create(serviceName);
      service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);

      /** Create a Dispatch instance from a service.* */
      System.out.println("Creating dispatch instance for service StockQuoteAxisService");
      Dispatch<SOAPMessage> dispatch = service.createDispatch(portName, SOAPMessage.class,
          Service.Mode.MESSAGE);
      System.out.println(" > Dispatch instance created.");

      /** Create SOAPMessage request. * */
      MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
```

```
      System.out.println("Creating a Request Reply SOAP Message");
      SOAPMessage request = mf.createMessage();

      /** Obtain the SOAPEnvelope and header and body elements **/
      SOAPPart part = request.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPHeader header = env.getHeader();
      SOAPBody body = env.getBody();

      /** Construct the message payload. **/
      SOAPElement operation = body.addChildElement("getQuote", "ns1",
          "soap.server.StockQuoteAxis_Wmq");
      SOAPElement value = operation.addChildElement("in0");
      value.addAttribute(new QName("https://www.w3.org/2001/XMLSchema-instance", "type"),
"string");
      value.addTextNode("XXX");
      request.saveChanges();
      System.out.println(" > SOAP Message created.");

      /** Invoke the service endpoint. **/
      DispatchClientAsyncCallback handler = new DispatchClientAsyncCallback();

      System.out
          .println("Invoking getQuote Request Reply operation asynchronously using a
callback...");
      Future<?> monitor = dispatch.invokeAsync(request, handler);
      System.out.println(" > getQuote call has returned");

      /** Sleep main thread until handler has been notified **/
      System.out.println("Waiting for handler to be called...");
      while (!monitor.isDone()) {
        Thread.sleep(100);
      }

      System.out.println("End of sample");
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

  public void handleResponse(Response<SOAPMessage> response) {
    try {
      // retrieve the result
      SOAPMessage ans = response.get();
      SOAPPart part = ans.getSOAPPart();
      SOAPEnvelope env = part.getEnvelope();
      SOAPBody body = env.getBody();

      /** Define name of the SOAP folders we are interested in **/
      QName responseName = new QName("soap.server.StockQuoteAxis_Wmq", "getQuoteResponse");
      QName resultName = new QName("getQuoteReturn");

      /** Retrieve result from SOAP envelope **/
      SOAPElement bodyElement = (SOAPElement) body.getChildElements(responseName).next();
      SOAPElement responseElement = (SOAPElement)
bodyElement.getChildElements(resultName).next();
      String result = responseElement.getValue();

      System.out.println(" > Async Handler has received a result of " + result);
    }
    catch (Exception fault) {
      // Identify the cause of the Axis Fault
      System.err.println("Exception in handleResponce");
      System.err.println(fault.toString());
      Throwable e = fault.getCause();
      for (int i = 1; e != null; i++) {
```

```
        // The toString method on an MQAxisException will cause the message, explanation and
user
        // action.
        System.err.println("Exception(" + i + "): " + e.toString());

        if (e.getCause() != null) {
          e = e.getCause();
        }
        else {
          break;
        }
      } // end of for loop
    } // end of catch block
  }

}
```

Tâches associées

Développement d'un client JAX-RPC pour le transport WebSphere pour SOAP à l'aide d' Eclipse
Développez un client de service Web Axis 1.4 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Développement d'un client .NET 1 ou 2 pour le transport WebSphere pour SOAP à l'aide de Microsoft Visual Studio 2008
Développez un client de service Web .NET 1 ou 2 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Développement d'un client .NET 1 ou 2 pour le transport WebSphere pour SOAP à l'aide de Microsoft Visual Studio 2008

Développez un client de service Web .NET 1 ou 2 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Avant de commencer

Vous pouvez démarrer le développement d'un client .NET 1 ou 2 de différentes manières:

1. Utilisez **amqwdeployWMQService** pour générer des stubs client à partir d'un service Web et les importer dans Visual Studio.
2. Utilisez **java2wsdl** pour générer un WSDL à partir d'une implémentation Java d'un service Web, puis utilisez `wsdl.exe`, fourni avec .NET, pour générer des stubs client.
3. Générez WSDL à partir d'une implémentation .NET `.asmx` du service à l'aide de **amqswsdl**, puis utilisez `wsdl.exe`.
4. Si vous avez développé et déployé le service pour HTTP, utilisez **Ajouter une référence Web ...** dans Visual Studio pour configurer le client pour qu'il accède au service HTTP. Modifiez l'URL en vous référant au service déployé sur WebSphere MQ transport for SOAP.

La tâche utilise le service développé dans «Développement d'un service .NET 1 ou 2 pour WebSphere MQ transport for SOAP à l'aide de Microsoft Visual Studio 2008», à la page 996.

Pourquoi et quand exécuter cette tâche

Procédez comme suit pour créer un client .NET 1 ou 2 pour HTTP et un transport WebSphere MQ pour SOAP.

Procédure

1. Créez l'application de console client et modifiez-la pour appeler le service Web HTTP StockQuote .
   a) Cliquez avec le bouton droit de la souris sur **Solution'StockQuoteDotNet'** dans l' **Explorateur de solutions** > Ajouter ... > Nouveau projet. Sélectionnez le type de projet **C#** , **.NET Framework 2.0**et **Console Application**. Nommez le projet `StockQuoteClientDotNet` > **OK**
   b) Cliquez avec le bouton droit de la souris sur **Solution'StockQuoteDotNet'** dans l' **Explorateur de solutions** > Ajouter ... > Nouveau projet. Sélectionnez le type de projet **C#** , **.NET Framework 2.0**et **Console Application**. Nommez le projet `StockQuoteClientDotNet` > **OK**

c) Cliquez avec le bouton droit de la souris sur **StockQuoteClientDotNet** > **Définir comme projet de démarrage**.

d) Cliquez avec le bouton droit de la souris sur **StockQuoteClientDotNet** > **Ajouter une référence Web ...** > Parcourir les services Web de cette solution > Sélectionnez **StockQuote** > **Ajouter une référence**. Notez que vous avez ajouté une référence Web à l'hôte local et un nouveau fichier de configuration `app.config`.

e) Dans l'explorateur de solutions, remplacez le nom de l'application de console `Program.cs` par `StockQuoteClientDotNet.cs` > Cliquez sur **OK** pour modifier toutes les utilisations de `Program.cs` en `StockQuoteClientDotNet.cs`.

f) Remplacez le contenu de `StockQuoteClientDotNet.cs` par le code dans Figure 190, à la page 1028.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figure 190. Programme HTTP StockQuoteClientDotNet

g) Lancez StockQuoteClientDotNet pour tester le service `StockQuote.asmx` :

i) Appuyez sur **F5**, cliquez sur la flèche verte dans la barre d'actions ou sur **Déboguer** > **Démarrer le débogage (F5)**.

Si le projet StockQuoteDOtNet se trouve dans la même solution, il démarre automatiquement. Sinon, vous devez d'abord démarrer le service.

La fenêtre de commande contenant les résultats s'ouvre derrière l'espace de travail. L'instruction `Console.ReadLine();` l'empêche de se fermer tant que vous n'appuyez pas sur **Entrée**.

Conseil : Assurez-vous que `StockQuote.asmx` est la page de démarrage du projet `StockQuoteDotNet` .

2. Modifiez StockQuoteClientDotNet pour appeler le service `StockQuote.asmx` à l'aide du transport WebSphere MQ pour SOAP.

a) Ajoutez les lignes affichées en gras au client.

```
using System;
using StockQuoteClientDotNet.localhost;
namespace StockQuoteClientDotNet {
  class StockQuoteClientDotNet {
    static void Main(string[] args) {
      try {
        IBM.WMQSOAP.Register.Extension();
        StockQuote stockobj = new StockQuote();
        Console.WriteLine("http reply is: "
          + stockobj.getNonInlineQuote("http request");
        stockobj.Url = "jms:/queue?"
          + "initialContextFactory=com.ibm.mq.jms.Nojndi"
          + "&connectionFactory=()&destination=REQUESTDOTNET@QM1"
          + "&targetService=StockQuote.asmx";
        Console.WriteLine("jms reply is: "
          + stockobj.getNonInlineQuote("jms request"));
      }
      catch (System.Exception e) {
        Console.WriteLine("Exception thrown: " + e.ToString());
      }
      Console.ReadLine();
    }
  }
}
```

Figure 191. Programme StockQuoteClientDotNet modifié

Vous pouvez également modifier l'URL par défaut. Ouvrez **StockQuoteClientDotNet** > **Propriétés** > **Settings.settings** et remplacez la valeur de la propriété `StockQuoteClientDotNet_localhost_StockQuote` par WebSphere MQ transport for SOAP URL.

b) Ajout d'une référence à `amqsoap.dll`

i) Dans le projet **StockQuoteClientDotNet** de l' **Explorateur de solutions**, cliquez avec le bouton droit de la souris sur **Références** > **Ajouter une référence ...** > Cliquez sur l'onglet **Parcourir** > accédez à *MQ_INSTALLATION_PATH*\bin > Sélectionnez **amqsoap.dll** > **OK**. *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ est installé.

3. Testez le client avec le service `StockQuote.asmx` à l'aide de WebSphere MQ transport for SOAP.

a) Ouvrez une fenêtre de commande dans le répertoire de projet StockQuoteDotNet : `.\StockQuoteDotNet\StockQuoteDotNet` > Vérifiez que `.bin\StockQuoteDotNet.dll` existe. Si ce n'est pas le cas, régénérez la solution.

b) Entrez la commande **amqwRegisterdotNet**.

Vous n'avez besoin d'exécuter **amqwRegisterdotNet** qu'une seule fois par installation.

c) Si vous avez exécuté **amqwdeployWMQServer** avec `genAsmxWMQBits`, exécutez le programme d'écoute SOAP .NET:
`generated\server\startWMQNListener`

d) Vous pouvez également exécuter le programme d'écoute directement:

```
amqwSOAPNETListener -u "jms:/queue?
destination=REQUESTDOTNET@QM1
&connectionFactory=()&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuote.asmx&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE"
-w C:\IBM\ID\StockQuoteDotNet\StockQuoteDotNet -n 10
```

4. Dans Visual Studio 2008, appuyez sur **F5** pour exécuter StockQuoteClientDotNet.

Clients de service Web .NET Framework 1 et .NET Framework 2

Les exemples de clients .NET fournis avec le transport WebSphere MQ for SOAP utilisent des stubs générés pour appeler les exemples de services Axis et .NET.

Pour les clients .NET Framework 1 et .NET Framework 2, WebSphere MQ fournit un accès aux services Web à l'aide de clients .NET. La commande **amqwdeployWMQService** comporte une option,

genProxiestoDotNet, qui génère des stubs client .NET Framework 1 ou .NET Framework 2 pour un service Web. Vous pouvez également utiliser des stubs client générés par l'outil .NET **wsdl** ou par Microsoft Visual Studio 2005 ou 2008.

Les exemples de client de service Web .NET Framework 1 et .NET sont installés dans *MQ_INSTALLATION_PATH*\tools\soap\samples\dotnet. *MQ_INSTALLATION_PATH* est le répertoire dans lequel WebSphere MQ est installé.

SQVB2Axis.vb
SQVB2Axis.vb, Figure 192, à la page 1030, est le client Visual Basic pour appeler le service **StockQuoteAxisService** .

SQVB2DotNet.vb
QVB2DotNet.vb, Figure 193, à la page 1030, est le client Visual Basic pour appeler le service **StockQuoteDotNet** .

SQCS2Axis.cs
SQCS2Axis.cs, Figure 194, à la page 1031, est le client C# pour appeler le service **StockQuoteAxisService** . Vous pouvez remplacer l'URL du service en fournissant une URL sur la ligne de commande.

SQCS2DotNet.cs
SQCS2DotNet.cs, Figure 195, à la page 1031, est le client C# pour appeler le service **StockQuoteDotNet** . Vous pouvez remplacer l'URL du service en fournissant une URL sur la ligne de commande.

```
Module SQVB2Axis
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj As New StockQuoteAxisService()
      Dim res As Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2Axis: reply is: '{0}'", res)
   End Function
End Module
```

Figure 192. SQVB2Axis

```
Module SQVB2DotNet
   Function Main(ByVal CmdArgs() As String) As Integer
      IBM.WMQSOAP.Register.Extension()
      Dim obj as new StockQuoteDotNet()
      Dim res as Single = obj.getQuote("fromcs")
      System.Console.WriteLine("SQVB2DotNet: reply is: '{0}'", res)
   End Function
End Module
```

Figure 193. SQVB2DotNet

```
using System;
class SQCS2Axis {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteAxisService stockobj = new StockQuoteAxisService();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("SQCS2Axis RPC reply is: " + res);
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2Axis DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figure 194. SQCS2Axis

```
using System;
class SQCS2DotNet {
   [STAThread]
   static void Main(string[] args) {
      try {
         IBM.WMQSOAP.Register.Extension();
         StockQuoteDotNet stockobj = new StockQuoteDotNet();
         if (args.GetLength(0) >= 1)
            stockobj.Url = args[0];
         System.Single res = stockobj.getQuote("XXX");
         Console.WriteLine("RPC reply is: " + res);
         if (args.GetLength(0) == 0) {
            res = stockobj.getQuoteDOC("XXX");
            Console.WriteLine("DOC reply is: " + res);
         }
      }
      catch (System.Exception e) {
         Console.WriteLine("\n>>> EXCEPTION WHILE RUNNING SQCS2DotNet DEMO <<<\n"
            + e.ToString());
      }
   }
}
```

Figure 195. SQCS2DotNet

Tâches associées

Développement d'un client JAX-RPC pour le transport WebSphere pour SOAP à l'aide d' Eclipse
Développez un client de service Web Axis 1.4 à exécuter à l'aide du transport WebSphere MQ pour SOAP.

Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse
Développer un client de service Web Axis2 à exécuter à l'aide de WebSphere MQ transport for SOAP. Les clients Axis2 exemples fournis avec WebSphere MQ transport for SOAP sont répertoriés, et la commande **wsimport** est utilisée pour générer des proxys.

Déploiement de services Web à l'aide du transport WebSphere MQ pour SOAP

Déployez un service Web dans l'un des différents environnements de serveur et connectez-le à l'aide du transport WebSphere MQ pour SOAP.

Avant de commencer

Développez un service Web et testez ce dernier à l'aide de SOAP sur HTTP dans l'environnement cible.

Pourquoi et quand exécuter cette tâche

Vous pouvez déployer un service Web à exécuter avec WebSphere MQ transport for SOAP dans un certain nombre d'environnements d'exécution SOAP différents. Vous pouvez déployer un service sur Axis 1.4 uniquement à l'aide du logiciel installé avec WebSphere MQ. Pour les autres environnements d'exécution, vous devez installer des logiciels supplémentaires.

Vous n'êtes pas limité à l'exécution du transport WebSphere MQ pour SOAP sur les serveurs pour lesquels il existe des instructions de déploiement. Utilisez les instructions pour déployer un service dans l'un des environnements répertoriés.

Remarque : Certains environnements intégrés offrent SOAP sur JMS à l'aide de la liaison SOAP JMS recommandée par W3C , ainsi que du transport WebSphere MQ pour la liaison SOAP. Les éditions de WebSphere MQ, jusqu'à 7.0.1.2inclus, prennent en charge uniquement le transport WebSphere MQ pour la liaison SOAP. A partir de 7.0.1.3 , vous pouvez déployer des clients Axis2 à l'aide d'un URI conforme à la recommandation de candidat W3C pour SOAP sur JMS. Voir le tutoriel Développement d'une application de services Web JAX-WS SOAP/JMS avec WebSphere Application Server V7 et Rational Application Developer V7.5.

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService

Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Avant de commencer

1. Suivez les instructions d'installation de WebSphere MQ transport for SOAP
2. Vérifiez l'installation et votre environnement à l'aide de la commande **runivt** .
3. Pour redéployer un service:
   a. Supprimez le sous-répertoire `./generated` et tous ses sous-répertoires.
   b. Supprimez les demandes de la file d'attente de destination et supprimez-la.
   c. Suivez les instructions de l'étape «2», à la page 1032.

Pourquoi et quand exécuter cette tâche

Ces instructions permettent de déployer un service Axis 1.4 pour la première fois. Pour redémarrer un service Axis 1.4 , réexécutez le programme d'écoute SOAP Axis 1.4 : étape «11», à la page 1033.

Utilisez les instructions suivantes pour déployer un nouveau service Axis 1.4 sur WebSphere MQ transport for SOAP:

Procédure

1. Créez un répertoire *deployDir* pour stocker les fichiers de déploiement.
   L'utilitaire de déploiement requiert que chaque service soit déployé à partir d'un répertoire distinct.
2. Ouvrez une fenêtre de commande dans Windowsou un interpréteur de commandes à l'aide de X Window System sur les systèmes UNIX and Linux , dans *deployDir* pour exécuter **amqwdeployWMQService**.
3. Exécutez **amqwsetcp** pour définir le chemin d'accès aux classes.
   Le JRE et le JDK doivent se trouver dans le chemin d'accès aux classes, à la version 5.0 ou ultérieure, et au même niveau de version.
4. Copiez la source de classe, *className*`.java`, dans *deployDir*
5. Copiez tous les fichiers source Java du même package que *className* dans *deployDir*/*packageName*, où *packageName* est une arborescence de répertoires correspondant au nom du package.

6. Exécuter**javac** *packageName.className*.

   Vous devrez peut-être ajouter un chemin d'accès au répertoire en cours "."ou au répertoire *packageName* pour **javac** afin de trouver les autres classes.

7. Créez le WSDL Axis pour le service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWsdl
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

8. Créez les ressources WebSphere MQ pour le service:

```
amqwdeployWMQService -f packageName.className.java -c genAxisWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

   Conseil :

   Si vous souhaitez configurer un nouveau gestionnaire de files d'attente et les ressources dont il a besoin pour effectuer des opérations de développement et de test, exécutez **setupWMQSOAP**.

   Si vous souhaitez configurer le nouveau gestionnaire de files d'attente comme gestionnaire de files d'attente par défaut, effectuez une copie de **setupWMQSOAP** à partir du répertoire *WMQ install directory*\tools\soap\samples et ajoutez le paramètre -q à la ligne

```
call :try -q crtmqm %QMGR%
```

9. Créez le programme d'écoute Axis et déployez le service:

```
amqwdeployWMQService -f packageName.className.java -c AxisDeploy
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))"
```

10. Si vous devez générer le WSDL pour le service, générer des stubs client ou des proxys client, exécutez **amqwdeployWMQService** avec l'un des paramètres suivants:

    - genAsmxWsdl
    - genAxisWsdl
    - genProxiesToDotNet
    - AxegenProxiesto

    Remarque : Vous devez générer un fichier WSDL avant de générer les proxys. L'option AllAxis échoue si la variable CLASSPATH n'est pas configurée pour rechercher toutes les classes importées pour compiler *className*.java. S'il existe plusieurs fichiers Java dans le package contenant *className*.java, vous devez d'abord les compiler à l'aide de **javac**. **amqwdeployWMQService** -f *packageName.className*.java -c CompileJava compile uniquement *className*.java.

11. Démarrez le programme d'écoute Axis généré.

```
.\generated\server\startWMQJListener.cmd
```

Tâches associées

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP
Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP
WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP

WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP

Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Avant de commencer

1. Suivez les instructions d'installation de WebSphere MQ transport for SOAP
2. Vérifiez l'installation et votre environnement à l'aide de la commande **runivt** .
3. Le chemin d'accès aux fichiers .NET Framework `wsdl.exe` et `csc.exe` doit être défini. Les copies de `wsdl.exe` et de `csc.exe` identifiées par la variable PATH doivent être au même niveau de .NET Framework. Si plusieurs infrastructures .NET sont installées ou si vous utilisez Visual Studio, vérifiez soigneusement la variable PATH .
4. Pour redéployer un service:
   a. Supprimez le sous-répertoire `./generated` et tous ses sous-répertoires
   b. Supprimez les demandes de la file d'attente de destination et supprimez-la.
   c. Suivez les instructions de l'étape «2», à la page 1034.

Pourquoi et quand exécuter cette tâche

Ces instructions permettent de déployer un service .NET pour la première fois. Pour redémarrer un service .NET, réexécutez le programme d'écoute SOAP .NET, étape «9», à la page 1035.

Utilisez les instructions suivantes pour déployer un nouveau service .NET Framework 1 ou .NET Framework 2 dans le transport WebSphere MQ pour SOAP:

Procédure

1. Créez un répertoire *deployDir* pour stocker les fichiers de déploiement.
   L'utilitaire de déploiement requiert que chaque service soit déployé à partir d'un répertoire distinct.
2. Ouvrez une fenêtre de commande dans *deployDir* pour exécuter **amqwdeployWMQService**.

```
C:\IBM\ID\QuoteClient>
```

3. Exécutez **amqwsetcp** pour définir le chemin d'accès aux classes.
   Un chemin d'accès aux classes est nécessaire uniquement pour les clients Axis.
4. Copiez le service .NET, *className*`.asmx`, dans *deployDir*
5. Générez l'implémentation de service dans une bibliothèque (`.dll`).

L'implémentation du service en ligne se trouve dans *className*.asmx. L'implémentation du service code-behind peut être *className*.asmx.cs.

La Figure 196, à la page 1035 illustre un exemple de commande permettant de générer un service .NET Framework V2 en tant que bibliothèque.

```
c:\WINDOWS\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.configuration.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Drawing.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Web.Services.dll
/reference:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll
/debug:pdbonly /filealign:512 /optimize+
/out:obj\Quote.dll /target:library Properties\AssemblyInfo.cs Quote.asmx.cs
```

Figure 196. Commande de génération pour le service V2 de .NET Framework

6. Copiez *className*.dll dans *deployDir**bin*.
7. Configurez les ressources WebSphere MQ et créez le programme d'écoute requis pour le service:

```
amqwdeployWMQService -f className.asmx -c genAsmxWMQBits
-v -u  "jms:/queue?destination=queueName
&initialContextFactory=com.ibm.mq.jms.Nojndi
&connectionFactory=(connectQueueManager(QmgrName)binding(auto))
&targetService=className.asmx"
```

8. Si vous devez générer le WSDL pour le service, générer des stubs client ou des proxys client, exécutez **amqwdeployWMQService** avec l'un des paramètres suivants:
   - genAsmxWsdl
   - genAxisWsdl
   - genProxiesToDotNet
   - AxegenProxiesto

   Remarque : Vous devez générer un fichier WSDL avant de générer les proxys.
9. Démarrez le programme d'écoute .NET généré.

```
.\generated\server\startWMQNListener.cmd
```

Tâches associées

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService
Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP
WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP

WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Avant de commencer

Utilisez les mêmes outils pour le développement d'un client ou d'un service pour WebSphere MQque pour le développement pour HTTP. CICS possède des outils correspondant à **Java2wsdl** et **wsdl2Java**:

- **DFHWS2LS** utilise une description de service Web comme point de départ. Il utilise les descriptions des messages et les types de données utilisés dans ces messages pour construire des structures de données en langage évolué. Vous pouvez utiliser dans les structures des programmes d'application écrits dans différents langages.
- **DFHLS2WS** prend une structure de données en langage évolué comme point de départ. Il utilise la structure pour construire une description de services Web qui contient les descriptions des messages. Il crée également des schémas pour les messages à partir de la structure de données de langage.

Suivez les instructions de la rubrique Création d'un service Web dans la documentation du produit CICS pour créer un service Web.

Pourquoi et quand exécuter cette tâche

Suivez les instructions Configuration de CICS pour utiliser le WebSphere MQ dans la documentation du produit CICS . A l'aide des instructions, vous pouvez déployer le service Web sur WebSphere MQ transport for SOAP.

Tâches associées

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService
Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP
Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP

WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Avant de commencer

Utilisez Rational Application Developer, WebSphere Integration Developer ou un kit d'outils de services Web pour développer le service Web.

Pourquoi et quand exécuter cette tâche

Utilisez les instructions suivantes pour déployer un service à l'aide du transport WebSphere MQ pour SOAP en tant que transport SOAP sur WebSphere Application Server.

Procédure

1. Configurez WebSphere MQ en tant que fournisseur de messagerie JMS pour le bus d'intégration de services sur WebSphere Application Server.
2. Configurez les ressources WebSphere MQ requises par le service.
3. Suivez les instructions, Configuration des ressources JMS pour le programme d'écoute de noeud final SOAP sur JMS synchrone, dans la documentation du produit WebSphere Application Server Network Deployment.

   Il existe des instructions correspondantes pour les autres plateformes WebSphere Application Server.
4. Modifiez l'URI de service pour qu'il soit conforme au transport WebSphere MQ pour l'URI SOAP.
5. Déployez le service sur WebSphere Application Server.

Que faire ensuite

Déployez le service avec HTTP en tant que transport afin que les clients puissent interroger le service et recevoir le WSDL en réponse.

Tâches associées

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService
Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP
Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP
WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS

Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Avant de commencer

La tâche requiert WebSphere Application Server v7.0.0.9 et WebSphere MQ v7.0.1.3.

Pourquoi et quand exécuter cette tâche

La tâche comporte deux étapes:

Procédure

1. «Configuration des ressources WebSphere MQ», à la page 1038
2. «Configuration des ressources WebSphere Application Server», à la page 1039

Que faire ensuite

«Configuration des ressources WebSphere MQ», à la page 1038

Tâches associées

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService
Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP
Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP
WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Configuration des ressources WebSphere MQ

Avant de commencer

Pour le support Axis2 , vous avez requis WebSphere MQ 7.0.1.3 ou version ultérieure.

Pourquoi et quand exécuter cette tâche

Par souci de simplicité, la tâche suppose que WebSphere MQ est installé sur le même poste de travail que les autres logiciels et utilise des connexions de liaisons. WebSphere Application Server et les configurations client Axis2 fonctionnent avec les connexions client. Pour exécuter une tâche à l'aide de connexions client, vérifiez que vous pouvez insérer et extraire des messages vers et depuis les files

d'attente de demandes et de réponses à partir du client Axis2 et des ordinateurs WebSphere Application Server.

Là encore, par souci de simplicité, aucune configuration de sécurité n'est utilisée. L'ID utilisateur dispose des droits `mqm` complets.

Procédure

1. Créez un gestionnaire de files d'attente par défaut, `QM1`.

   Utilisez WebSphere MQ Explorer pour créer `QM1` comme gestionnaire de files d'attente par défaut. Configurez-le pour qu'il démarre automatiquement et sélectionnez l'option de création d'un programme d'écoute. Vous pouvez également utiliser les commandes suivantes:

   ```
   crtmqm -q -sa QM1
   strmqm
   echo define listener (LISTENER.TCP) trptype(tcp) ipaddr(localhost) port(1414)
              control(qmgr) replace | runmqsc
   echo start listener(LISTENER.TCP) | runmqsc
   ```

2. Définissez une file d'attente de demandes, REQUESTAXIS, et une file d'attente de réponses, REPLYAXIS.

   Utilisez l'explorateur ou les commandes suivantes:

   ```
   echo define ql(REQUESTAXIS) replace | runmqsc
   echo define ql(REPLYAXIS) replace | runmqsc
   ```

Que faire ensuite

«Configuration des ressources WebSphere Application Server», à la page 1039

Configuration des ressources WebSphere Application Server

Avant de commencer

Pour la prise en charge de W3C SOAP sur JMS, vous avez besoin de WebSphere Application Server v7. Cette configuration a été effectuée sur WebSphere Application Server Version 7.0 Test Environment v7.0.0.9 Mise à jour 1. WebSphere Application Server a été fourni avec Rational Software Architect for WebSphere Software 7.5.4. Rational Software Architect a été mis à jour vers v7.5.5.1, en appliquant les dernières mises à jour disponibles.

Dans le cadre du processus d'installation, créez un profil pour WebSphere Application Server. Dans la tâche, la sécurité administrative n'est pas activée. Le nom de profil par défaut est `was70profile1`et le serveur est `server1`.

Pourquoi et quand exécuter cette tâche

Configurez WebSphere Application Server. Vous pouvez démarrer le serveur à partir de Rational Application Developer et démarrer la console d'administration à partir de la vue Serveurs ou vous pouvez démarrer le serveur à l'aide d'un fichier de commandes et administrer le serveur à l'aide d'un navigateur. La tâche utilise la deuxième méthode.

Les fichiers de commandes du serveur se trouvent dans le dossier *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\bin`. Les fichiers journaux que vous souhaitez inspecter se trouvent dans *`Rational Installation Root`*`\SDP\runtimes\base_v7\profiles\was70profile1\logs\server1`.

Par convention, tous les noms d'objet WebSphere MQ sont en majuscules et tous les noms JNDI faisant référence aux objets WebSphere MQ sont en minuscules.

Procédure

1. Démarrer le serveur.

```
startServer server1
```

2. Démarrez un navigateur, ouvrez la console d'administration et connectez-vous.

```
http://localhost:9061/ibm/console/unsecureLogon.jsp
```

   Entrez une chaîne dans la zone ID utilisateur.

3. Créez une fabrique de connexions, qm1
   a) Dans le navigateur, ouvrez **Ressources** > **JMS** > **Fabriques de connexions**.
   b) Dans la fenêtre Fabriques de connexions, sélectionnez la portée **Node =*nodename***et cliquez sur **Nouveau**.
   c) Sélectionnez **WebSphere MQ** > **OK**.
   d) Indiquez les informations de connexion du gestionnaire de files d'attente à partir de Tableau 143, à la page 1040 > **Suivant**.

Tableau 143. Informations de connexion du gestionnaire de files d'attente	
Nom de zone	**Valeur**
Nom	qm1
Nom JNDI	qm1

   e) Sélectionnez **Entrez toutes les informations requises dans cet assistant** comme méthode de connexion > **Suivant**.
   f) Entrez QM1 comme détails de connexion de file d'attente > **Suivant**.
   g) Entrez les détails de connexion à partir de Tableau 144, à la page 1040 > **Suivant**.

Tableau 144. Détails de connexion	
Nom de zone	**Valeur**
Transport	Bindings, then client
Nom d'hôte	localhost
Port	1414
Canal de connexion serveur	SYSTEM.DEF.SVRCONN

   h) **Tester la connexion** > **Suivant** > **Terminer** > **Sauvegarder**.

4. Créez la file d'attente des demandes JMS, requestaxis.
   a) Dans le Navigator, ouvrez **Ressources** > **JMS** > **Files d'attente**.
   b) Dans la fenêtre Fabriques de connexions, sélectionnez la portée **Node =*nodename***et cliquez sur **Nouveau**.
   c) Sélectionnez **WebSphere MQ** > **OK**.
   d) Entrez les détails de la file d'attente dans Tableau 145, à la page 1040 > **OK** > **Sauvegarder**.

Tableau 145. Détails de file d'attente	
Nom de zone	**Valeur**
Nom	requestaxis
Nom JNDI	requestaxis
Nom de la file d'attente	REQUESTAXIS
Nom gest. de files	QM1

5. Répétez l'étape «4», à la page 1040 pour créer la file d'attente de réponses JMS, replyaxis.

6. Créez une spécification d'activation, `qm1as`.

   La spécification d'activation déclenche le bean géré par message (MDB) du routeur de service Web lorsqu'un message arrive dans la file d'attente des demandes. Le bean géré par message est défini dans le descripteur de déploiement du service Web créé par l'assistant de service Web Rational Application Developer.

   a) Dans le Navigator, ouvrez **Ressources** > **JMS** > **Spécifications d'activation**.

   b) Dans la fenêtre Fabriques de connexions, sélectionnez la portée **Node =*nodename***et cliquez sur **Nouveau**.

   c) Sélectionnez **WebSphere MQ** > **OK**.

   d) Entrez les attributs de base de la spécification d'activation à partir de Tableau 146, à la page 1041 > **Suivant**.

   Tableau 146. Nom de la spécification d'activation

Nom de zone	**Valeur**
Nom	`qm1as`
Nom JNDI	`qm1as`

   e) Spécifiez ses informations MDB à partir de Tableau 147, à la page 1041 > **Suivant**.

   Tableau 147. Informations MDB

Nom de zone	**Valeur**
Nom JNDI de destination	`requestaxis`
Sélecteur de message	*laissé vide*
Type de destination	`Queue`

   f) Sélectionnez **Entrez toutes les informations requises dans cet assistant** comme méthode de connexion > **Suivant**.

   g) Entrez `QM1` comme détails de connexion de file d'attente > **Suivant**.

   h) Entrez les détails de connexion à partir de Tableau 144, à la page 1040 > **Suivant**.

   Tableau 148. Détails de connexion

Nom de zone	**Valeur**
Transport	`Bindings, then client`
Nom d'hôte	`localhost`
Port	`1414`
Canal de connexion serveur	`SYSTEM.DEF.SVRCONN`

   i) **Tester la connexion** > **Suivant** > **Terminer** > **Sauvegarder**.

7. Créez une fabrique de connexions de file d'attente, `jms/WebServicesReplyQCF`, pour la file d'attente de réponses.

   Le routeur de services Web utilise une fabrique de connexions de file d'attente pour accéder à une file d'attente de réponses. Dans le descripteur de déploiement du service Web, la fabrique de connexions de file d'attente reçoit le nom JNDI par défaut `jms/WebServicesReplyQCF`. Vous pouvez modifier le nom dans le descripteur de déploiement. Dans cette tâche, ajoutez le nom par défaut aux définitions de ressource JMS.

   a) Dans le Navigator, ouvrez **Ressources** > **JMS** > **Fabriques de connexions de file d'attente**.

   b) Dans la fenêtre Fabriques de connexions, sélectionnez la portée **Node =*nodename***et cliquez sur **Nouveau**.

c) Sélectionnez **WebSphere MQ** > **OK**.

d) Entrez les attributs de base de la fabrique de connexions de file d'attente à partir de Tableau 149, à la page 1042 > **Suivant**.

Tableau 149. Nom de fabrique de connexions de file d'attente	
Nom de zone	**Valeur**
Nom	`WebServicesReplyQCF`
Nom JNDI	`jms/WebServicesReplyQCF`

e) Sélectionnez **Entrez toutes les informations requises dans cet assistant** comme méthode de connexion > **Suivant**.

f) Entrez `QM1` comme détails de connexion de file d'attente > **Suivant**.

g) Entrez les détails de connexion à partir de Tableau 144, à la page 1040 > **Suivant**.

Tableau 150. Détails de connexion	
Nom de zone	**Valeur**
Transport	`Bindings, then client`
Nom d'hôte	`localhost`
Port	`1414`
Canal de connexion serveur	`SYSTEM.DEF.SVRCONN`

h) **Tester la connexion** > **Suivant** > **Terminer** > **Sauvegarder**.

Que faire ensuite

«Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS», à la page 999

Déploiement d'un service sur le noeud final de service WebSphere ESB et Process Server pour utiliser WebSphere Transport for SOAP

WebSphere MQ transport for SOAP n'est pas directement pris en charge par WebSphere ESB et Process Server. Vous devez configurer une exportation personnalisée.

Pourquoi et quand exécuter cette tâche

WebSphere Integration Developer fournit une transformation de données SOAP que vous pouvez lier à l'exportation JMS WebSphere MQ pour créer une exportation SOAP personnalisée WebSphere MQ JMS.

Suivez les instructions de création d'une exportation personnalisée pour recevoir des demandes SOAP sur WebSphere MQ transport for SOAP.

Procédure

1. Lisez Présentation des importations et des exportations et Comment vous connecter à WebSphere MQ dans la documentation du produit WebSphere Process Server for Multiplatforms V6.2 .
2. Suivez la tâche Génération d'une liaison d'exportation JMS MQ dans la documentation du produit IBM Business Process Manager, version 8.6 .

   Utilisez la liaison de données SOAP décrite dans Transformations de format de données JMS préintégrées pour formater le message SOAP.

Tâches associées

Déploiement d'un service sur Axis 1.4 à utiliser pour le transport WebSphere pour SOAP à l'aide de amqwdeployWMQService

Déployez un service Axis 1.4 dans WebSphere MQ transport for SOAP en créant un répertoire de déploiement, en exécutant la commande **amqwdeployWMQService** et en démarrant le programme d'écoute Axis 1.4 .

Déploiement d'un service sur .NET Framework 1 ou 2 pour utiliser le transport WebSphere MQ pour SOAP
Déployez un service .NET Framework 1 ou 2 sur WebSphere MQ transport for SOAP. Créez un répertoire de déploiement, exécutez la commande **amqwdeployWMQService** et démarrez le programme d'écoute .NET.

Déploiement d'un service sur CICS Transaction Server pour utiliser WebSphere Transport for SOAP
WebSphere Le transport MQ pour SOAP est intégré à la prise en charge des services Web CICS Transaction Server 4.1 .

Déploiement d'un service sur WebSphere Application Server pour utiliser WebSphere Transport for SOAP
WebSphere MQ transport for SOAP est intégré au bus d'intégration de services sur WebSphere Application Server.

Configuration de WebSphere Application Server pour l'utilisation de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 1 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur le serveur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Configurez les ressources WebSphere MQ et WebSphere Application Server pour développer et déployer un service Web lié à W3C SOAP sur JMS en tant que transport.

Déploiement de clients de service Web pour utiliser le transport WebSphere MQ pour SOAP

Déployez un client de service Web dans l'un des différents environnements client et connectez-vous à un service à l'aide du transport WebSphere MQ pour SOAP.

Avant de commencer

Développez le service Web et déployez-le pour utiliser le transport WebSphere MQ pour SOAP.

Pourquoi et quand exécuter cette tâche

Vous pouvez déployer un client de service Web à exécuter avec WebSphere MQ transport for SOAP dans un certain nombre d'environnements client différents. Vous pouvez déployer un client Java sur Axis 1.4 en utilisant uniquement le logiciel installé avec WebSphere MQ. Pour les autres environnements client, vous devez installer des logiciels supplémentaires.

Vous n'êtes pas limité à l'exécution du transport WebSphere pour SOAP dans les environnements client pour lesquels il existe des instructions de déploiement. Utilisez les instructions pour déployer un client dans l'un des environnements pris en charge.

Remarque : Certains environnements intégrés offrent SOAP sur JMS à l'aide de la liaison SOAP JMS recommandée par W3C , ainsi que du transport WebSphere MQ pour la liaison SOAP. Les éditions de WebSphere MQ, jusqu'à 7.0.1.2inclus, prennent en charge uniquement le transport WebSphere MQ pour la liaison SOAP. A partir de 7.0.1.3 , vous pouvez déployer des clients Axis2 à l'aide d'un URI conforme à la recommandation de candidat W3C pour SOAP sur JMS. Voir le tutoriel Développement d'une application de services Web JAX-WS SOAP/JMS avec WebSphere Application Server V7 et Rational Application Developer V7.5.

Déploiement d'un client de service Web sur Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP

Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux IBM WebSphere MQ , démarrez le service et testez le client.

Avant de commencer

Conseil : Déployez le service sur HTTP, développez et testez le client pour HTTP, puis modifiez le client pour le transport IBM WebSphere MQ pour SOAP:

1. Ajoutez l'appel Register.extension() au client.
2. Modifiez l'adresse de service Web statique dans la classe du localisateur de proxy client afin d'utiliser l'URI du transport IBM WebSphere MQ pour SOAP.

Pourquoi et quand exécuter cette tâche

Le déploiement d'un client Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP nécessite une étape de déploiement supplémentaire par rapport à un client HTTP. Vous devez créer un descripteur de déploiement client, `client-config.wsdd`, pour mapper le transport `jms:` à la classe d'expéditeur `com.ibm.mq.soap.transport.jms.WMQSender`.

Si vous utilisez la commande **amqwdeployWMQService** pour générer des proxys client, vous pouvez déployer le client à l'aide des répertoires générés par la commande.

Procédure

1. Créez un répertoire *deployDir* pour stocker les fichiers de déploiement du client.
2. Ouvrez une fenêtre de commande sur les systèmes Windows ou un interpréteur de commandes à l'aide de X Window System sur les systèmes UNIX and Linux , dans *deployDir*.
3. Exécutez la commande **amqwsetcp.cmd** pour définir la variable CLASSPATH
4. Exécutez la commande **amqwclientconfig.cmd** pour créer un descripteur de déploiement client Axis 1.4 , `client-config.wsdd` , dans *deployDir*.
5. Vérifiez que les classes du package client, les classes du proxy client et les bibliothèques utilisées par le client se trouvent dans la variable CLASSPATH.

   amqwdeployWMQService place les proxys client .NET dans `./generated/server/soap/client/remote/dotnetService` et les proxys Axis 1.4 dans `./generated/server/soap/client/remote/`*`client package`*.

Exemple

L'exemple illustre la configuration et la sortie, Figure 199, à la page 1045, d'un client Axis 1.4 Java . Le client, Figure 198, à la page 1045, appelle un service Web qui fait écho à son paramètre d'entrée. La définition de service, Figure 197, à la page 1044, affiche l'URI provenant du WSDL du service.

```
<wsdl:service name="QuoteSOAPImplService">
    wsdl:port binding="intf:org.example.www.QuoteSOAPImplBindingSoap"
                        name="org.example.www.QuoteSOAPImpl_Wmq">
       <wsdlsoap:address location="jms:/queue?destination=REQUESTAXIS
           &connectionFactory=(connectQueueManager(QM1)binding(server))
           &initialContextFactory=com.ibm.mq.jms.Nojndi
           &targetService=org.example.www.QuoteSOAPImpl.java
           &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE" />
    </wsdl:port>
</wsdl:service>
```

Figure 197. Définition du service

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Figure 198. Client Axis 1.4 Java

```
C:\IBM\ID\Test>dir /s /b
C:\IBM\ID\Test\client-config.wsdd
C:\IBM\ID\Test\org
C:\IBM\ID\Test\org\example
C:\IBM\ID\Test\org\example\www
C:\IBM\ID\Test\org\example\www\GetQuoteFaultMsg.class
C:\IBM\ID\Test\org\example\www\OrgExampleWwwQuoteSOAPImplBindingSoapStub.class
C:\IBM\ID\Test\org\example\www\QuoteClient.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImpl.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplService.class
C:\IBM\ID\Test\org\example\www\QuoteSOAPImplServiceLocator.class

C:\IBM\ID\Test>amqwsetcp
C:\IBM\ID\Test>java org.example.www.QuoteClient.class
Response = IBM
```

Figure 199. Configuration et sortie du client

Que faire ensuite

1. Si vous déployez le client en tant que client IBM WebSphere MQ , configurez le canal de connexion client et serveur.
2. Si vous déployez le client sur un gestionnaire de files d'attente différent pour le service, vous devez rendre la file d'attente de destination disponible pour le client. Configurez la file d'attente de destination sur le gestionnaire de files d'attente de service en tant que file d'attente de cluster ou sur le gestionnaire de files d'attente client en tant que définition de file d'attente éloignée.

Tâches associées

Déploiement d'un client de service Web sur Axis2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un fichier de configuration Axis2 pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 4 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Modifiez l'URL dans le client Axis2 développé pour WebSphere MQ transport for SOAP afin d'utiliser la recommandation de candidat W3C pour SOAP sur JMS.

Déploiement d'un client de service Web sur .NET Framework 1 et 2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez le proxy client et la classe client. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement d'un client de service Web sur Axis2 pour utiliser le transport WebSphere MQ pour SOAP

Préparez un répertoire de déploiement et un fichier de configuration Axis2 pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Avant de commencer

Conseil : Déployez le service sur HTTP. Développez et testez le client pour HTTP, puis modifiez l'URL pour référencer le service à l'aide du transport WebSphere MQ pour SOAP.

Cette tâche explique comment déployer un client Axis2 non géré dans Java Standard Edition. Vous pouvez déployer un client Axis2 dans un conteneur Web. Dans «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013, vous avez développé un client dans un conteneur Web et vous l'avez déployé dans WebSphere Application Server Community Edition. Dans le cadre de la configuration du serveur, vous avez activé la facette Axis2 et l'avez incluse dans la configuration du conteneur Web. Pour configurer des conteneurs Web sur d'autres serveurs d'applications, reportez-vous à la documentation Axis2 , http://ws.apache.org/axis2/1_4_1/installationguide.html#servlet_containerou à la documentation fournie avec le serveur Web.

Remarque : Axis2 utilise le terme conteneur de servlet. Un conteneur de servlet est identique à un conteneur Web.

Pourquoi et quand exécuter cette tâche

Le déploiement d'un client Axis2 pour utiliser WebSphere MQ transport for SOAP est similaire au déploiement d'un client Axis2 pour utiliser HTTP. Des étapes supplémentaires sont requises pour fournir un chemin d'accès aux classes aux fichiers JAR WebSphere MQ et pour modifier le fichier de configuration Axis2 . Le fichier de configuration Axis2 requiert une entrée supplémentaire pour JMS. L'entrée fait référence au fichier JAR WebSphere MQ transport for SOAP qui implémente JMS `transportSender`.

Axis2 fournit un script, `axis2.bat` ou `axis2.sh`, qui simplifie le déploiement client ; voir les exemples dans Figure 203, à la page 1048 et Figure 204, à la page 1048.

Remarque :

1. `axis2.bat` comporte un bogue qui doit être corrigé. La chaîne `-Djava.ext.dirs="%AXIS2_HOME%\lib\"` doit être remplacée par `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`.
2. Dans `axis2.bat` et `axis2.sh`, `-Djava.ext.dirs` est utilisé pour référencer rapidement tous les fichiers JAR Axis2 au lieu de les ajouter séparément au chemin d'accès aux classes. Malheureusement, cette approche est imparfaite et ne fonctionne qu'avec certains JRE. Il ne fonctionne pas avec les environnements d'exécution Java IBM .

   Le paramètre JVM, `-Djava.ext.dirs="%AXIS2_HOME%\lib\\"`, met les fichiers JAR Axis à la disposition de la machine virtuelle Java. La machine virtuelle Java tente d'instancier certains des fichiers JAR Axis et génère une erreur dont les détails dépendent de la machine virtuelle Java. En règle générale, vous pouvez voir l'une des lignes suivantes dans la trace de pile:

   ```
   org.apache.axiom.om.util.UUIDGenerator.getInitialUUID(UUIDGenerator.java:76)
   ```

   ou `org.apache.axis2.deployment.DeploymentException: java.security.NoSuchAlgorithmException: MD5 MessageDigest not available`

   La méthode correcte pour exécuter un client Axis2 non géré consiste à ajouter les fichiers JAR Axis2 au chemin d'accès aux classes. Le chemin d'accès aux classes est disponible uniquement pour l'application client et non pour la machine virtuelle Java.

La procédure décrit les étapes générales d'exécution d'un client Axis2 non géré sans utiliser le script axis2 . Les exemples dans Figure 201, à la page 1048 et Figure 202, à la page 1048 sont des scripts pour Windows et Linux.

Procédure

1. Téléchargez Axis2 1.4.1 à partir de http://ws.apache.org/axis2/download/1_4_1/download.cgi et décompressez-le dans un dossier, `Axis2-1.4.1`.
2. Mettez à jour `axis2.xml` dans `Axis2-1.4.1\conf`.
   a) Mettez à jour `axis2.xml` dans `Axis2-1.4.1\conf`. Ajoutez WebSphere MQ transport for SOAP en tant que `transportSender`:

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender"/>
   ```

   b) Si nécessaire, modifiez la taille du pool de connexions par défaut de 10.

   ```
   <transportSender name="jms"
   class="com.ibm.mq.axis2.transport.jms.WMQJMSTransportSender">
   <parameter name="ResourcePoolCapacity">20</parameter>
   </transportSender>
   ```

   La `capacitéResourcePool` définit le nombre d'entrées de noeud final de service qui sont conservées dans le cache. La valeur doit être au moins égale à 1. Si le nombre d'entrées de noeud final de service dépasse la taille du cache, les entrées sont supprimées pour laisser de la place aux nouvelles entrées. La taille d'une entrée de noeud final varie. Définissez un nombre suffisamment grand pour éviter l'emballement du cache.

   Voir l'étape 3 dans «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013.
3. Créez un répertoire *deployDir*. Sous ce répertoire, copiez la structure de dossiers contenant le client et les proxys client. *deployDir* est équivalent au dossier *`project`*`\bin` dans un projet Java Eclipse .
4. Ouvrez une fenêtre de commande sous Windowsou un interpréteur de commandes à l'aide de X Window System sur les systèmes UNIX and Linux , dans *deployDir*.
5. Mettez à jour le chemin d'accès aux classes pour inclure le répertoire en cours, les fichiers JAR Axis2 , `com.ibm.mqjms.jar` et `com.ibm.mq.axis2.jar`.

   `com.ibm.mqjms.jar` fait référence à tous les autres fichiers JAR WebSphere MQ requis.
6. Utilisez la commande **Java** pour démarrer le programme client.

Exemples

Quatre exemples d'exécution d'un client Axis2 sont répertoriés dans Figure 202, à la page 1048 à Figure 204, à la page 1048. La Figure 200, à la page 1047 présente la sortie de l'exécution du client asynchrone répertorié dans le Figure 183, à la page 1018.

```
cd C:\IBM\ID\Workspaces\Axis2docs\StockQuoteAxis2PojoClient\bin>
runpojo soap/client/SQA2AsyncClient

HTTP Sync: 55.25
Callback constructor
Waiting for HTTP callback
Result in Callback 55.25
HTTP poll: 55.25
JMS: Sync: 55.25
Callback constructor
Waiting for JMS callback
Result in Callback 55.25
JMS poll: 55.25
Press any key to continue . . .
```

Figure 200. Sortie de l'exécution de SQA2AsyncClient

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

setlocal EnableDelayedExpansion
set CLASSPATH=
set AXIS2_CLASS_PATH=
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

"%JAVA_HOME%\bin\java" -Daxis2.repo="%AXIS2_HOME%\repository"
-Daxis2.xml="%AXIS2_HOME%\conf\axis2.xml" -cp
".;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar;%AXIS2_CLASS_PATH%" %1

pause
```

Figure 201. runpojo.bat: Windows, à l'aide d'un chemin d'accès aux classes

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm
# update classpath
AXIS2_CLASSPATH=""
for f in "$AXIS2_HOME"/lib/*.jar
do
  AXIS2_CLASSPATH="$AXIS2_CLASSPATH":$f
done
AXIS2_CLASSPATH="$AXIS2_HOME":"$JAVA_HOME/lib/tools.jar":"$AXIS2_CLASSPATH":"$CLASSPATH"
java -cp /home/alex/dev/sandbox/Soap/axis2:/opt/mqm/java/lib/com.ibm.mqjms.jar:
/opt/mqm/java/lib/com.ibm.mq.axis2.jar:$AXIS2_CLASSPATH
-Daxis2.xml=/home/alex/dev/sandbox/axis2-1.4.1/conf/axis2.xml %1
```

Figure 202. runpojo.sh: Linux, utilisation d'un chemin d'accès aux classes.

```
@echo off
set AXIS2_HOME=C:\OpenSource\axis2-1.4.1
set JAVA_HOME=C:\IBM\Java50
set WMQ_HOME=C:\IBM\MQ\java\lib

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
pause
```

Figure 203. runaxis2.bat: Windows, utilisation de axis2.bat

Important

```
export AXIS2_HOME=/home/OpenSource/axis2-1.4.1
export JAVA_HOME=/usr/lib/j2sdk1.5-ibm

%AXIS2_HOME%\bin\axis2 -cp .;%WMQ_HOME%\com.ibm.mqjms.jar;%WMQ_HOME%\com.ibm.mq.axis2.jar; %1
```

Figure 204. runaxis2.sh: Linux, utilisation de axis2.sh

Important

Tâches associées

Déploiement d'un client de service Web sur Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP

Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux IBM WebSphere MQ , démarrez le service et testez le client.

Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS

Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 4 de la connexion d'un client

de service Web Axis2 et d'un service Web déployé sur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Modifiez l'URL dans le client Axis2 développé pour WebSphere MQ transport for SOAP afin d'utiliser la recommandation de candidat W3C pour SOAP sur JMS.

Déploiement d'un client de service Web sur .NET Framework 1 et 2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez le proxy client et la classe client. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS

Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 4 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Modifiez l'URL dans le client Axis2 développé pour WebSphere MQ transport for SOAP afin d'utiliser la recommandation de candidat W3C pour SOAP sur JMS.

Avant de commencer

Vous devez d'abord effectuer la tâche «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013 pour appeler **SimpleJavaListener** à l'aide d'un client Axis2 et du protocole WebSphere MQ transport for SOAP.

Vous devez également avoir créé le service Web et configuré WebSphere MQ et WebSphere Application Server dans les tâches précédentes:

1. «Configuration des ressources WebSphere MQ», à la page 1038.
2. «Configuration des ressources WebSphere Application Server», à la page 1039.
3. «Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS», à la page 999.

Dans la tâche, le client s'exécute dans Eclipse Galileo. Vous pouvez exécuter le client à partir de la ligne de commande en modifiant le fichier `Axis2.bat` fourni avec Axis2.

Pourquoi et quand exécuter cette tâche

La seule modification que vous devez apporter au client statique Axis2 `StockQuoteAxis` existant pour appeler le service Axis StockQuotehébergé par WebSphere Application Server consiste à modifier l'URL transmise au client. Comme le WSDL n'a pas été modifié, vous pouvez utiliser les mêmes classes de proxy dans le package `soap.server`.

Vous disposez de deux approches pour définir l'URL à transmettre au client. Vous pouvez utiliser la même URL que dans le `StockQuoteAxis.wsdl`généré. Vous devez ajouter les paramètres `jndiInitialContextFactory` et `jndiURL` pour accéder au répertoire JNDI de WebSphere Application Server. Une autre approche consiste à modifier l'URL et à donner au client un accès direct aux files d'attente `REQUESTAXIS` et `REPLYAXIS` sur `QM1`, sans utiliser de recherche JNDI.

Les paramètres de connexion que vous définissez dans l'URL transmise au client Axis2 sont utilisés pour la connexion au gestionnaire de files d'attente WebSphere MQ et aux files d'attente requises pour l'envoi et la réception de messages SOAP. Les paramètres de connexion transmis au client Axis2 ne sont pas nécessairement utilisés par le service. Vous pouvez utiliser les fonctions de mise en file d'attente répartie de WebSphere MQ pour dissocier le client et le service de l'utilisation du même gestionnaire de files d'attente ou du même serveur de noms.

Procédure

1. Sauvegardez l'URL à partir du `StockQuoteAxis.wsdl` généré et fermez Rational Application Developer pour sauvegarder la mémoire.

Si vous n'avez pas modifié la configuration du serveur, la fermeture de Rational Application Developer arrête le serveur d'applications. Dans ce cas, démarrez le serveur à l'aide de la commande suivante:

```
startserver server1
```

2. Ouvrez Eclipse Galileo dans l'espace de travail avec le projet client Axis2 .
3. Ouvrez SQA2StaticClient.java.

   Voir SQA2StaticClient.java.
4. Appelez le service à l'aide de la variante queue de l'URI.
   a) Modifiez l'URL.

      Le nouvel URI est:

      ```
      jms:queue:REQUESTAXIS
                ?replyToName=REPLYAXIS
                &connectionFactory=connectQueueManager(QM1)Bind(Server)
                &targetService=StockQuoteAxis;
      ```

      Comparez ceci à l'URL de StockQuoteAxis.wsdl:

      ```
      jms:jndi:requestaxis
               ?jndiConnectionFactoryName=qm1
               &targetService=StockQuoteAxis
      ```

      Figure 205. URL de StockQuoteAxis.wsdl

      - REQUESTAXIS est désormais en majuscules car il s'agit d'un nom de file d'attente et non d'un nom JNDI.
      - La connexion à QM1 est simple.
      - L'URI ne contient pas le nom de la destination de réponse. Le client doit définir la file d'attente dans laquelle il attend des réponses.

   b) Exécutez SQA2StaticClient.java en utilisant le même **Exécuter en tant que ...** comme vous l'avez fait dans la tâche «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013.
5. Appelez le service à l'aide de la variante jndi de l'URI, en utilisant WebSphere Application Server comme serveur de noms.
   a) Utilisez l'URL de StockQuoteAxis.wsdl, Figure 205, à la page 1050, en fournissant les paramètres manquants pour utiliser le service de nommage dans WebSphere Application Server.

      Les paramètres et valeurs manquants que vous devez fournir sont les suivants:

Tableau 151. Paramètres JNDI supplémentaires

Paramètre	Valeur utilisée dans cet exemple	Description
&jndiURL	iiop://localhost:2810 ou corbaname:iiop:localhost:2810	URI du fournisseur de noms. Pour WebSphere Application Server, la valeur par défaut est 2809. Il est également appelé numéro de port du connecteur RMI et port d'amorçage. La valeur est répertoriée dans le SystemOut.log `00000000 NameServerImp A NMSV0018I: Name server available on bootstrap port 2810`

Tableau 151. Paramètres JNDI supplémentaires (suite)		
Paramètre	**Valeur utilisée dans cet exemple**	**Description**
`&jndiInitialContextFactory`	`com.ibm.websphere.naming.WsnInitialContextFactory`	Nom de la fabrique de contexte initial utilisée par WebSphere Application Server.
`&replyToName`	`replyaxis`	Nom JNDI de la file d'attente REPLYAXIS .

```
jms:jndi:requestaxis?
         &jndiURL=iiop//localhost:2810
         &jndiConnectionFactoryName=qm1
         &jndiInitialContextFactory=com.ibm.websphere.naming.WsnInitialContextFactory
         &targetService=StockQuoteAxis
         &replyToName=replyaxis;
```

b) Ajoutez les fichiers JAR requis par la recherche JNDI.

Dans cette configuration, les fichiers JAR suivants ont été ajoutés au chemin de génération pour exécuter la tâche à l'aide de la variante `jndi` de l'URL JMS:

- `com.ibm.jaxws.thinclient_7.0.0.jar` à partir de *`Rational install directory`*`\SDP\runtimes\base_v7\runtimes`.
- `com.ibm.ws.runtime.jar` depuis *`Rational install directory`*`\SDP\runtimes\base_v7\plugins`

Pour un fournisseur JNDI différent, vous avez besoin de fichiers JAR différents.

Les autres fichiers JAR du chemin de génération sont les suivants:

i) Tous les fichiers JAR dans *`WebSphere MQ Install directory`*`\java\lib`.

ii) Tous les fichiers JAR dans *`Axis2-1.5.1`*`\lib`.

iii) Java 6.0 JRE.

c) Exécutez `SQA2StaticClient.java` en utilisant le même **Exécuter en tant que ...** comme vous l'avez fait dans la tâche «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013.

Résultats

Dans les deux cas, la réponse du service s'affiche dans la vue de la console client.

Tâches associées

Déploiement d'un client de service Web sur Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux IBM WebSphere MQ , démarrez le service et testez le client.

Déploiement d'un client de service Web sur Axis2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un fichier de configuration Axis2 pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement d'un client de service Web sur .NET Framework 1 et 2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez le proxy client et la classe client. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement d'un client de service Web sur .NET Framework 1 et 2 pour utiliser le transport WebSphere MQ pour SOAP

Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez le proxy client et la classe client. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Avant de commencer

Conseil : Développez et testez le service et le client à l'aide de Visual Studio. Modifiez ensuite le client pour WebSphere MQ transport for SOAP.

1. Si vous déployez un service à l'aide de .NET Framework 1 ou 2, générez le service en tant que bibliothèque (`.dll`). Déploiement à l'aide du transport WebSphere MQ pour SOAP.
2. Ajoutez l'appel Register.Extension() au client.
3. Ajoutez une référence à amqsoap.dll, qui se trouve dans *MQ_Install*`\bin`.
4. Remplacez la propriété statique `Url` du constructeur de classe de proxy client par l'URI `jms:/` , pour WebSphere MQ transport for SOAP.

Pourquoi et quand exécuter cette tâche

Le déploiement d'un client de service Web pour .NET Framework 1 ou 2 afin d'utiliser WebSphere MQ transport for SOAP nécessite une étape de déploiement supplémentaire. Vous devez enregistrer `amqsoap.dll` avec .NET Framework. `amqsoap.dll` est automatiquement enregistré dans le cadre de l'installation de WebSphere MQ transport for SOAP, mais vous devrez peut-être l'enregistrer à nouveau.

Si vous utilisez la commande **amqwdeployWMQService** pour générer des proxys client, vous pouvez déployer le client à l'aide des répertoires générés par la commande.

Procédure

1. Créez un répertoire *deployDir* pour stocker les fichiers de déploiement du client.
2. Ouvrez une fenêtre de commande dans *deployDir*.
3. Exécutez **amqwsetcp** pour définir CLASSPATH si le service doit s'exécuter sur Axis 1.4.
4. Si nécessaire, exécutez **amqwRegisterDotNet** pour enregistrer `amqsoap.dll` auprès de .NET Framework.

Exemple

L'exemple illustre la configuration et la sortie, Figure 208, à la page 1053, d'un client .NET Framework V2 . Le client, Figure 207, à la page 1053, appelle un service Web qui fait écho à son paramètre d'entrée. La définition d'URL statique, Figure 206, à la page 1052, affiche le constructeur du proxy client.

```
public Quote() {
    this.Url = "jms:/queue?destination=REQUESTDOTNET
                &connectionFactory=(connectQueueManager(QM1)binding(server))
                &initialContextFactory=com.ibm.mq.jms.Nojndi
                &targetService=Quote.asmx
                &replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE";
}
```

Figure 206. Constructeur de proxy client statique

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
            } catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
            }
        }
    }
}
```

Figure 207. Programme client

```
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>dir /s /b
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram\QuoteClientProgram.exe
C:\IBM\ID\DotNet\QuoteClientProgram\QuoteClientProgram>quoteclientprogram
Response is: IBM
```

Figure 208. Configuration et sortie

Que faire ensuite

1. Si vous déployez le client en tant que client WebSphere MQ MQI, configurez le canal de connexion client et serveur.
2. Si vous déployez le client sur un gestionnaire de files d'attente différent pour le service, vous devez rendre la file d'attente de destination disponible pour le client. Configurez la file d'attente de destination sur le gestionnaire de files d'attente de service en tant que file d'attente de cluster ou sur le gestionnaire de files d'attente client en tant que définition de file d'attente éloignée.

Tâches associées

Déploiement d'un client de service Web sur Axis 1.4 pour utiliser le transport IBM WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un descripteur de déploiement pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux IBM WebSphere MQ , démarrez le service et testez le client.

Déploiement d'un client de service Web sur Axis2 pour utiliser le transport WebSphere MQ pour SOAP
Préparez un répertoire de déploiement et un fichier de configuration Axis2 pour le client. Indiquez les proxys client et la classe client, puis configurez la variable CLASSPATH. Configurez les files d'attente et les canaux WebSphere MQ , démarrez le service et testez le client.

Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS
Un service Web lié à la recommandation de candidat W3C pour SOAP sur JMS doit s'exécuter dans le conteneur d'EJB d'un serveur d'applications Java EE . Cette tâche est l'étape 4 de la connexion d'un client de service Web Axis2 et d'un service Web déployé sur WebSphere Application Server à l'aide du protocole W3C SOAP sur JMS. Modifiez l'URL dans le client Axis2 développé pour WebSphere MQ transport for SOAP afin d'utiliser la recommandation de candidat W3C pour SOAP sur JMS.

Connexion d'un client Axis2 à un service JAX-WS à l'aide de W3C SOAP sur JMS et de WebSphere Application Server

Une fois cette tâche terminée, vous aurez appelé un service Web JAX-WS s'exécutant dans WebSphere Application Server à partir d'un client Axis2 . Le client Axis2 et WebSphere Application Server utilisent la recommandation de candidat W3C pour le protocole SOAP sur JMS exécuté sur WebSphere MQ. Utilisez

Eclipse Galileo et Rational Application Developer pour générer le client de service Web et le service Web, respectivement.

Avant de commencer

La tâche requiert la version 7 de Rational Software Development Environment et WebSphere Application Server. La tâche a été créée à l'aide de Rational Application Developer fourni avec Rational Software Architect for WebSphere Software v7.5.5.1et WebSphere Application Server version 7.0 Test Environment v7.0.0.9 Mise à jour 1. Vous avez également besoin de WebSphere MQ v7.0.1.3.

La tâche s'appuie sur deux autres tâches, «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992et «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013. Pour effectuer ces tâches, votre environnement de développement a déjà installé Eclipse Galileo, WASCE, le plug-in Eclipse pour WASCE et Axis2 1.4.1 . Vous n'avez pas besoin de WASCE pour cette tâche.

Certaines des étapes sont complexes. Les étapes supposent que vous êtes familiarisé avec le développement d'applications de service Web pour WebSphere Application Server à l'aide de Rational Application Developer. Les demandes de processeur et de mémoire de la tâche sont importantes. La tâche a été effectuée dans une machine virtuelle VMWare Windows XP SP3 allouée 1.8GB de mémoire.

Installez tous les logiciels avant de démarrer la tâche. Le logiciel prend environ une journée à télécharger et une journée à installer, en fonction de la bande passante. La tâche prend au moins une demi-journée.

Pourquoi et quand exécuter cette tâche

Le scénario de cette tâche est que vous avez développé un service Web de cotation boursière, `StockQuoteAxis`, à l'aide d'un outil open source, Eclipse Galileo. `StockQuoteAxis` est déployé à l'aide de SOAP sur HTTP s'exécutant sur un serveur open source, WASCE.

Vous souhaitez lier les services Web que vous déployez à un transport de messagerie standard, tel que SOAP sur JMS, ou à une messagerie fiable des services Web, ainsi qu'à SOAP sur HTTP. Vous souhaitez que le client et le service utilisent des interfaces basées sur des normes. Pour cette raison, bien que votre future équipe de développement de projets ait implémenté une solution à l'aide du transport WebSphere MQ pour SOAP, vous n'êtes pas passé en production.

Le client Axis2 a supprimé le problème selon lequel le client SOAP pour le transport WebSphere MQ pour SOAP nécessitait une modification du client HTTP. Le problème persiste: le service connecté par le transport IBM WebSphere MQ pour SOAP est hébergé par un programme d'écoute spécial fourni par WebSphere MQ: `SimpleJavaListener`.

Avec la norme W3C SOAP sur JMS dans le statut des recommandations candidates, certains fournisseurs prennent en charge W3C SOAP sur JMS. La prise en charge vous permet de déployer un service Web sur un serveur d'applications et de vous connecter au même service à l'aide de divers protocoles de connectivité. La prise en charge fournie par WebSphere Application Server v7 supprime le problème d'avoir à héberger le service Web séparément afin d'utiliser un transport SOAP basé sur les messages. L'utilisation d'une interface de transport de messages basée sur des normes, JMS, signifie que vous pouvez développer des solutions à l'aide d'outils de différents fournisseurs. Vous espérez que les outils de services Web d' Eclipse incluront des liaisons SOAP sur JMS à l'avenir.

La plupart des étapes sont effectuées à l'aide d' Eclipseou des outils de gestion fournis avec les produits WebSphere . Les étapes sont décrites pour un environnement Windows . Avec de légères modifications apportées à certaines commandes, vous pouvez effectuer les étapes sur d'autres plateformes.

Les étapes préliminaires de création du service Web HTTP et de connexion à ce service à l'aide de Axis2 sont répertoriées. Le client et WSDL, à partir de ces étapes, sont utilisés pour créer la solution

Procédure

1. Connexion au service Web StockQuoteAxis à l'aide d'un client Axis2 et du transport IBM WebSphere MQ pour SOAP

a) «Développement d'un service JAX-RPC pour WebSphere MQ transport for SOAP à l'aide d' Eclipse», à la page 992

b) «Développement d'un client JAX-WS pour WebSphere transport for SOAP à l'aide d'Eclipse», à la page 1013

c) «Déploiement d'un client de service Web sur Axis2 pour utiliser le transport WebSphere MQ pour SOAP», à la page 1046

2. Connectez-vous au service Web Axis StockQuoteà l'aide d'un client Axis2 et de la recommandation candidate W3C pour SOAP sur JMS.

a) «Configuration des ressources WebSphere MQ», à la page 1038

b) «Configuration des ressources WebSphere Application Server», à la page 1039

c) «Développement d'un service Web EJB JAX-WS pour W3C SOAP sur JMS», à la page 999

d) «Déploiement sur un client Axis2 à l'aide de W3C SOAP sur JMS», à la page 1049

WebSphere MQ Bridge for HTTP

Avec le pont WebSphere MQ pour HTTP, les applications client peuvent échanger des messages avec WebSphere MQ sans qu'il soit nécessaire d'installer un client WebSphere MQ MQI. Vous pouvez appeler WebSphere MQ à partir de n'importe quelle plateforme ou langue avec des fonctions HTTP.

Présentation du pont WebSphere MQ pour HTTP

Le pont WebSphere MQ pour HTTP est une application Web Java, Enterprise Environment (JEE). Les clients HTTP peuvent lui envoyer des demandes **POST**, **GET**et **DELETE** pour insérer, parcourir et supprimer des messages des files d'attente WebSphere MQ . Le pont WebSphere MQ pour HTTP ne convient pas aux messages, si une distribution assurée est requise.

Avantages

Avec le pont WebSphere MQ pour HTTP, vous pouvez envoyer et recevoir des messages WebSphere MQ en utilisant HTTP à partir d'une grande variété d'environnements:

- Environnements qui prennent en charge HTTP, mais pas WebSphere MQ.
- Environnements dont l'espace de stockage est insuffisant pour installer un client WebSphere MQ MQI.
- Environnements trop nombreux pour installer le client WebSphere MQ MQI sur chaque système nécessitant un accès à WebSphere MQ.
- Applications Web à partir desquelles vous souhaitez envoyer ou recevoir des messages sans coder votre propre pont vers WebSphere MQ.
- Applications Web que vous souhaitez améliorer, à l'aide de techniques asynchrones telles que AJAX. WebSphere Le pont MQ pour HTTP rend disponibles les files d'attente et les rubriques WebSphere MQ à l'aide de REST (Representation State Transfer) sur HTTP.

La prise en charge HTTP peut être utilisée avec des topologies de messagerie point-à-point et publication / abonnement.

Comment la prise en charge HTTP fonctionne-t-elle?

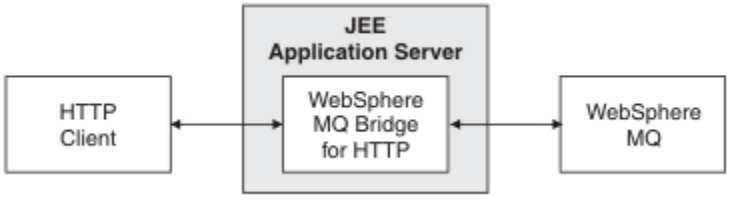


Figure 209. WebSphere MQ Bridge for HTTP

Le pont WebSphere MQ pour l'application Web HTTP reçoit les demandes HTTP d'un ou de plusieurs clients. Il interagit avec WebSphere MQ pour leur compte et leur renvoie des réponses HTTP.

Le pont WebSphere MQ pour HTTP est un servlet JEE qui est connecté à WebSphere MQ à l'aide d'un adaptateur de ressources. Le servlet HTTP gère trois types différents de demandes HTTP: **POST**, **GET**et **DELETE**.

Tableau 152. WebSphere Pont MQ pour les instructions HTTP

Demande HTTP	**Résultat**
POST	Insère un message dans une file d'attente ou une rubrique.
GET	Parcourt le premier message d'une file d'attente. Conformément au protocole HTTP, **GET** ne supprime pas le message de la file d'attente. N'utilisez pas **GET** avec la messagerie de publication / abonnement.
SUPPRIMER	Extrait et supprime un message d'une file d'attente ou d'une rubrique.

Exemple HTTP POST

HTTP **POST** place un message dans une file d'attente ou une publication dans une rubrique. L'exemple Java **HTTPPOST** est un exemple de demande HTTP **POST** d'un message dans une file d'attente. Au lieu d'utiliser Java, vous pouvez créer une demande HTTP **POST** à l'aide d'un formulaire de navigateur ou d'un kit d'outils AJAX.

La Figure 210, à la page 1056 illustre une demande HTTP d'insertion d'un message dans une file d'attente appelée `myQueue`. Cette demande contient l'en-tête HTTP `x-msg-correlId` pour définir l'ID de corrélation du message WebSphere MQ .

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

*Figure 210. Exemple de demande HTTP **POST** dans une file d'attente*

Figure 211, à la page 1056 montre la réponse renvoyée au client. Le contenu de la réponse est inexistant.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Figure 211. Exemple de réponse HTTP POST

Exemple HTTP DELETE

HTTP **DELETE** obtient un message d'une file d'attente et supprime le message, ou extrait et supprime une publication. L'exemple **HTTPDELETE** Java est un exemple de demande HTTP **DELETE** qui lit un message à partir d'une file d'attente. Au lieu d'utiliser Java, vous pouvez créer une demande HTTP **DELETE** à l'aide d'un formulaire de navigateur ou d'un kit d'outils AJAX.

Figure 212, à la page 1057 est une demande HTTP de suppression du message suivant dans la file d'attente appelée `myQueue`. En réponse, le corps du message est renvoyé au client. Dans les termes WebSphere MQ , HTTP **DELETE** est un get destructeur.

La demande contient l'en-tête de demande HTTP `x-msg-wait`, qui indique à WebSphere MQ le temps d'attente d'un message dans la file d'attente. Elle contient également l'en-tête de demande `x-msg-require-headers`, qui spécifie que le client doit recevoir l'ID de corrélation du message dans la réponse.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figure 212. Exemple d'une demande HTTP **DELETE***

Figure 213, à la page 1057est la réponse renvoyée au client. L'ID de corrélation est renvoyé au client, comme demandé dans l'en-tête `x-msg-require-headers` de la demande.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

*Figure 213. Exemple de réponse HTTP **DELETE***

Exemple HTTP GET

HTTP **GET** envoie un message depuis une file d'attente. Le message reste dans la file d'attente. Dans les termes WebSphere MQ , HTTP **GET** est une demande de navigation. Vous pouvez créer une demande HTTP **GET** avec un client Java, un formulaire de navigateur ou un kit d'outils AJAX.

Figure 214, à la page 1057 est une demande HTTP permettant de parcourir le message suivant dans la file d'attente appelée `myQueue`.

La demande contient l'en-tête de demande HTTP `x-msg-wait`, qui indique à WebSphere MQ le temps d'attente d'un message dans la file d'attente. Elle contient également l'en-tête de demande `x-msg-require-headers`, qui spécifie que le client doit recevoir l'ID de corrélation du message dans la réponse.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

*Figure 214. Exemple d'une demande HTTP **GET***

Figure 215, à la page 1058 est la réponse renvoyée au client. L'ID de corrélation est renvoyé au client, comme demandé dans l'en-tête `x-msg-require-headers` de la demande.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

*Figure 215. Exemple de réponse HTTP **GET***

Installation, configuration et vérification du pont WebSphere MQ pour HTTP

Procurez-vous le pont WebSphere MQ pour HTTP en installant "Java Messaging and Web Services" à partir des éléments d'installation du client ou du serveur WebSphere MQ MQI. Déployez le pont WebSphere MQ pour HTTP sur un serveur d'applications approprié.

Avant de commencer

Vérifiez les produits prérequis à l'adresse Configuration système requise pour IBM WebSphere MQ. Le processus d'installation ne vérifie pas la présence et la disponibilité des logiciels prérequis pour l'exécution du pont WebSphere MQ pour HTTP. Vous devez vérifier que les prérequis sont installés.

WebSphere MQ for HTTP s'exécute sur n'importe quel serveur d'applications compatible Java EE 1.4 , en installant l'adaptateur de ressources WebSphere MQ . Vous pouvez également exécuter le pont WebSphere MQ pour HTTP sur une édition WebSphere Application Server antérieure à la version 6.0.2.1. Utilisez WebSphere Application Server Message Listener Port (MLP) pour intégrer WebSphere MQ en tant que fournisseur JMS.

La prise en charge du pont WebSphere MQ pour HTTP est fournie uniquement pour les serveurs d'applications suivants:

- WebSphere Application Server 6.0.2.1 et versions ultérieures.
- WebSphere Application Server Community Edition Version 1.1 et versions ultérieures.

Pourquoi et quand exécuter cette tâche

WebSphere Le pont MQ pour HTTP est fourni sous la forme d'un fichier `.war` , `WMQHTTP.war`.

- Sur les plateformes UNIX et Linux,
  - `WMQHTTP.war` est inclus dans l'option d'installation "Java Messaging and Web Services" . L'option est disponible dans les éléments d'installation du client et du serveur.
  - `WMQHTTP.war` est installé dans *<mqmtop>*`/java/http/WMQHTTP.war`. *< mqmtop>* est le répertoire dans lequel WebSphere MQ est installé.
  - `WMQHTTP.samples` est installé dans *<mqmtop>*`/java/http/samples`. *< mqmtop>* est le répertoire dans lequel WebSphere MQ est installé.

Effectuez les étapes d'installation suivantes pour installer le pont WebSphere MQ pour HTTP, le déployer et le configurer, puis vérifiez la configuration. Les détails des étapes de configuration varient en fonction des différents serveurs d'applications. Utilisez «Déploiement et vérification du pont WebSphere MQ for HTTP sur WebSphere Application Server V6.1.0.9», à la page 1059 comme modèle pour les étapes à suivre sur votre serveur d'applications.

Procédure

1. Obtenez `WMQHTTP.war` en installant le client ou le serveur WebSphere MQ MQI.
2. Copiez `WMQHTTP.war` sur un serveur à partir duquel il peut être déployé sur un serveur d'applications.
3. Déployez `WMQHTTP.war` sur un serveur d'applications.

4. Si nécessaire, installez WebSphere MQ en tant qu'adaptateur de ressources sur votre serveur d'applications.

   Vérifiez si WebSphere MQ est déjà configuré en tant que fournisseur de messagerie sur votre serveur d'applications. Utilisez l'outil d'administration ou de gestion fourni avec votre serveur d'applications pour rechercher WebSphere MQ. WebSphere MQ se trouve sous le chemin suivant: **Ressources > JMS > Fournisseurs de messagerie**.
5. Configurez une fabrique de connexions sur le serveur d'applications pour la connexion à un gestionnaire de files d'attente qui utilise le transport client WebSphere MQ MQI[12].
6. Configuration de l'application Web `WMQHTTP.war` sur le serveur d'applications pour utiliser la fabrique de connexions
7. Vérifiez la configuration.
   a) Configurez le gestionnaire de files d'attente nommé dans la fabrique de connexions et une file d'attente locale.
   b) Placez un message dans la file d'attente locale.
   c) Créez le canal de connexion serveur nommé dans la fabrique de connexions, avec le droit de lire et d'écrire dans la file d'attente locale.
   d) Démarrez le gestionnaire de files d'attente et le programme d'écoute.
   e) Démarrez le serveur d'applications et `WMQHTTP.war`, s'ils ne sont pas déjà en cours d'exécution.
   f) Ouvrez un navigateur et entrez `http://`*`hostname`*`:`*`web port`*`/`*`Context root`*`/msg/queue/`*`local queue`*

Résultats

La fenêtre du navigateur affiche le message que vous avez placé dans la file d'attente locale.

Que faire ensuite

1. Essayez l'exemple «Déploiement et vérification du pont WebSphere MQ for HTTP sur WebSphere Application Server V6.1.0.9», à la page 1059.
2. Exécutez les exemples d'applications Java HTTP.

Déploiement et vérification du pont WebSphere MQ for HTTP sur WebSphere Application Server V6.1.0.9

Utilisez l'exemple suivant pour préparer un déploiement de la passerelle WebSphere MQ pour HTTP afin d'exécuter les exemples de programmes Java HTTP. Le déploiement se trouve sur WebSphere Application Server V6.1.0.9.

Avant de commencer

1. Suivez les instructions de la rubrique «Installation, configuration et vérification du pont WebSphere MQ pour HTTP», à la page 1058pour copier `WMQHTTP.war` sur un serveur accessible à votre installation de WebSphere Application Server.
2. Configurez un gestionnaire de files d'attente et une file d'attente à utiliser pour tester la configuration:
   - Dans l'exemple, le gestionnaire de files d'attente est configuré en utilisant les valeurs dans Tableau 153, à la page 1059:

Tableau 153. Configuration du gestionnaire de files d'attente	
Objet	**Valeur**
Nom d'hôte	`itso-01`

[12] Au départ, configurez au moins le transport client. Certains serveurs d'applications peuvent se connecter à WebSphere MQ à l'aide de connexions directes ou en mode liaisons.

Tableau 153. Configuration du gestionnaire de files d'attente (suite)	
Objet	**Valeur**
Gestionnaire de files d'attente	QM1
File d'attente locale	HTTPTESTQ
Canal de connexion serveur	MYSVRCON: Configurez un ID utilisateur MCA disposant des droits suffisants pour lire et écrire dans HTTPTESTQ.
Port d'écoute	1414

3. Démarrage du gestionnaire de files d'attente et du programme d'écoute
4. Placez un message de test dans HTTPTESTQ. Exemple :
   a. Démarrez WebSphere MQ Explorer.
   b. Dans la liste des files d'attente locales pour QM1, cliquez avec le bouton droit de la souris sur **HTTPTESTQ > Insertion d'un message de test > type `First Message` > Insertion d'un message > Fermer**
5. Démarrez le serveur d'applications et connectez-vous à la console Integrated Solutions Console.

Pourquoi et quand exécuter cette tâche

L'exemple montre les étapes à suivre si vous exécutez WebSphere Application Server V6.1.0.9 en tant que serveur d'applications. Si vous exécutez une autre version de WebSphere Application Server ou un autre serveur d'applications, les étapes sont différentes. WebSphere Application Server V6.1.0.9 est préconfiguré avec WebSphere MQ installé en tant que fournisseur de messages, à l'aide des bibliothèques client WebSphere MQ MQI. Si WebSphere MQ n'est pas préconfiguré en tant que fournisseur de messagerie ou si vous souhaitez utiliser des liaisons de serveur WebSphere MQ , vous devez installer et configurer l'adaptateur de ressources WebSphere MQ pour JEE sur votre serveur d'applications.

Suivez les instructions pour déployer le pont WebSphere MQ pour HTTP sur WebSphere Application Server V6.1.0.9et vérifiez le déploiement à l'aide d'un navigateur:

Procédure

1. Dans le panneau de navigation, cliquez sur **Ressources > Fournisseurs JMS > WebSphere MQ**.

   Vous pouvez effectuer la configuration au niveau du noeud, de la cellule ou du serveur, en fonction de votre déploiement WebSphere Application Server. L'exemple utilise le déploiement au niveau du serveur.
2. Sous **Propriétés supplémentaires**, cliquez sur **Fabriques de connexions > Nouveau**.
3. Dans le formulaire des fournisseurs JMS, indiquez les informations dans Tableau 154, à la page 1060, ou des alternatives de votre choix, cliquez sur **Appliquer > Sauvegarder**.

Tableau 154. Définissez ou modifiez les zones suivantes	
Zone	**Valeur**
Nom	WMQHTTPBridge
Nom JNDI	jms/WMQHTTPJCAConnectionFactory
Gestionnaire de files d'attente	QM1
Hôte	itso-01
Port	1414
Canal	MYSVRCON

Tableau 154. Définissez ou modifiez les zones suivantes (suite)	
Zone	**Valeur**
Type de transport	client

4. Dans le panneau de navigation, cliquez sur **Applications > Installer une nouvelle application**.
5. Insérez le chemin d'accès à `WMQHTTP.war` dans le formulaire et indiquez une racine de contexte, puis cliquez sur **Suivant**.
   a) La racine de contexte est facultative. `mq` est la racine de contexte par défaut pour les exemples d'applications HTTP.
   b) La racine de contexte fait partie de l'URI identifiant le pont WebSphere MQ pour HTTP. Vous pouvez omettre la racine de contexte ou la modifier ultérieurement.
6. Dans la page **Sélectionner les options d'installation** de l'assistant d'installation, il n'est pas nécessaire de modifier les valeurs par défaut. Cliquez sur **Suivant**.
7. Dans la page **Mappage des modules vers les serveurs** , sélectionnez un cluster ou un serveur, cochez la case Sélectionner, puis cliquez sur **Appliquer > Suivant**.
8. Sur la page **Mappage des références de ressource vers les ressources** , dans le formulaire **javax.jms.ConnectionFactory** , cliquez sur **Parcourir ...** sur la ligne IBM WebSphere MQ Bridge for HTTP.
9. Dans la page **Applications d'entreprise > Ressources disponibles** , sélectionnez **WMQHTTPBridge**et cliquez sur **Appliquer**.
10. Dans le formulaire **javax.jms.ConnectionFactory** , sélectionnez la méthode d'authentification.
    a) Pour l'exemple, choisissez **Aucun**, puis cliquez sur **Appliquer**. Les autres options nécessitent une configuration supplémentaire.
11. Cochez la case **Sélectionner** pour IBM WebSphere MQ Bridge for HTTP, cliquez sur **Suivant > Suivant > Terminer > Sauvegarder** .
12. Dans le panneau de navigation, cliquez sur **Applications > Applications d'entreprise**.
13. Cochez la case de sélection pour `WMQHTTP.war`, puis cliquez sur **Démarrer**.
14. Ouvrez une fenêtre de navigateur. Entrez `http://itso-01:9080/mq/msg/queue/HTTPTESTQ`en utilisant le nom d'hôte et le port appropriés.

Résultats

La fenêtre du navigateur affiche `First Message`si la configuration aboutit.

Que faire ensuite

Exécutez les exemples d'applications Java HTTP.

Publication / abonnement à l'aide du pont WebSphere MQ pour HTTP

WebSphere MQ Bridge for HTTP utilise l'interface de publication / abonnement WebSphere MQ classes for JMS. HTTP **POST** crée une publication. HTTP **DELETE** crée un abonnement géré non durable. Vous devez configurer la publication / l'abonnement pour JMS avant d'utiliser l'URI de rubrique.

La fonction de publication / abonnement est entièrement intégrée à WebSphere MQ dans la version 7. Avant la version 7, un courtier de publication / abonnement distinct traitait les publications et les abonnements. Il est appelé publication / abonnement "en file d'attente" , pour le distinguer de la publication / abonnement entièrement intégrée dans la version 7. La version 7 émule l'abonnement de publication en file d'attente à l'aide de la publication / abonnement intégré. L'émulation permet aux applications de publication / abonnement en file d'attente existantes de coexister avec des applications intégrées s'exécutant sur le même gestionnaire de files d'attente. Les applications de publication / abonnement en file d'attente peuvent également interagir avec des applications intégrées, partageant les mêmes rubriques. Dans la version 6, le courtier était livré avec WebSphere MQ; avant la version 6, il était disponible en tant que SupportPack.

Configuration

Le pont WebSphere MQ pour HTTP utilise l'interface JMS pour la publication et l'abonnement. Dans la version 7, vous pouvez contrôler si les classes WebSphere MQ for JMS utilisent la publication / l'abonnement en file d'attente ou intégrée, à l'aide de la propriété JMS `PROVIDERVERSION` .

Vous pouvez également utiliser des bibliothèques client WebSphere MQ MQI avec le pont WebSphere MQ pour HTTP ou des bibliothèques serveur. Les bibliothèques client de la version 6 ne prennent en charge que la publication / l'abonnement en file d'attente, tandis que les bibliothèques de la version 7 prennent en charge à la fois la publication / l'abonnement en file d'attente et la publication / l'abonnement intégré. La plupart des serveurs Web ou d'applications qui utilisent WebSphere MQ comme fournisseur de messagerie utilisent des bibliothèques client. Pour utiliser la publication / l'abonnement intégré, les bibliothèques client et serveur WebSphere MQ MQI doivent être au moins à la version 7. Si l'un ou l'autre exécute une version antérieure de WebSphere à 7, vous devez configurer la publication / l'abonnement en file d'attente ; voir Tableau 155, à la page 1062. Vérifiez quelles bibliothèques sont installées ou configurées avec le serveur Web ou le serveur d'applications que vous utilisez.

Tableau 155. Modes de configuration de publication / abonnement

	Client V6 ou version antérieure	**Client V7 ou version ultérieure**
Serveur V6 ou version antérieure	1. Exécutez le script `\java\bin\MQJMS_PSQ.mqsc`	Non pris en charge
Serveur V7 ou version ultérieure	1. Exécutez le script `\java\bin\MQJMS_PSQ.mqsc` 2. Définissez le gestionnaire de files d'attente sur `PSMODE=ENABLED`	1. Si PROVIDERVERSION = 7 a. Définissez le gestionnaire de files d'attente sur `PSMODE=ENABLED` ou `PSMODE=COMPAT` 2. Si PROVIDERVERSION = 6 a. Définissez le gestionnaire de files d'attente sur `PSMODE=ENABLED`

Publication

Envoyez une demande HTTP **POST** avec l'URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

Le contenu du message est publié à l'aide de la chaîne de rubrique *`topicString`*.

S'abonner

Envoyez une demande HTTP **DELETE** avec l'URI:

```
http://hostname:port/context_root/msg/topic/topicString
```

Le pont WebSphere MQ pour HTTP crée un abonnement non durable géré à la chaîne de rubrique *`topicString`*. L'abonnement est supprimé dès qu'une publication est renvoyée ou jusqu'à l'expiration de l'intervalle d'attente défini par l'en-tête d'entité personnalisé, `x-msg-wait`.

Exécution du pont WebSphere MQ pour les exemples HTTP

Le pont WebSphere MQ pour les exemples HTTP ne peut être utilisé que sur le système d'exploitation Windows . Les exemples montrent comment soumettre des commandes HTTP **POST** et HTTP **DELETE** à un pont WebSphere MQ pour HTTP à partir de programmes Java.

Avant de commencer

Vérifiez votre pont WebSphere MQ pour l'installation HTTP en exécutant l'étape «7», à la page 1059 dans «Installation, configuration et vérification du pont WebSphere MQ pour HTTP», à la page 1058.

Les exemples HTTP sont installés dans les répertoires indiqués dans le Tableau 156, à la page 1063. Dans chaque cas, le code source est installé dans le sous-répertoire `/src` .

Tableau 156. Emplacement des exemples HTTP

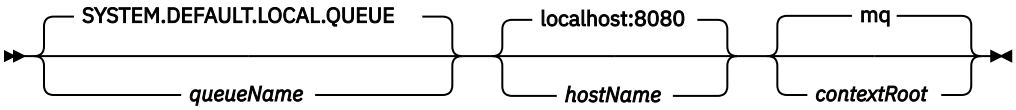
Plateforme	**Emplacement**
Windows	*MQ_INSTALLATION_PATH*/tools/http/samples
Toutes les autres plateformes	*MQ_INSTALLATION_PATH*/samp/http
MQ_INSTALLATION_PATH représente le répertoire dans lequel WebSphere MQ est installé.	

Pourquoi et quand exécuter cette tâche

Les exemples simulent les modèles d'application WebSphere MQ AMQSPUT et AMQSGET. Ils illustrent les fonctions suivantes dans un environnement de messagerie point-à-point:

- **HTTPPOST** -Envoie des demandes HTTP **POST** dans une application Java pour placer des messages dans une file d'attente WebSphere MQ , à l'aide du pont WebSphere MQ pour HTTP et gère les réponses.
- **HTTPDELETE** -Envoie des demandes HTTP **DELETE** dans une application Java pour obtenir des messages à partir d'une file d'attente WebSphere MQ , à l'aide du pont WebSphere MQ pour HTTP et gère les réponses contenant le message WebSphere MQ .

Paramètres pour HTTPPOST et HTTPDELETE



Pour exécuter l'exemple **HTTPPOST** , procédez comme suit:

Procédure

1. Dans une fenêtre de commande, accédez au répertoire des exemples HTTP.
2. Exécutez l'exemple **HTTPPOST** .

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   Lorsque l'exemple **HTTPPOST** démarre, la sortie suivante s'affiche:

   ```
   HTTP POST Sample start
   Target server is 'hostName'
   Target queue is 'queueName'
   Target context-root is 'contextRoot'
   ```

3. Dans l'invite de commande, entrez le texte devant former le corps du message.
4. Appuyez sur Entrée pour envoyer le message à la file d'attente WebSphere MQ .
   a) Si vous souhaitez envoyer un autre message, entrez du texte supplémentaire.
      Les textes forment le corps d'un deuxième message WebSphere MQ .
   b) Appuyez sur Entrée pour envoyer le message à la file d'attente WebSphere MQ .
5. Appuyez deux fois sur Entrée pour arrêter **HTTPPOST**.
   La sortie suivante s'affiche :

```
HTTP POST Sample end
```

Que faire ensuite

L'exemple **HTTPDELETE** effectue une extraction destructive de tous les messages que vous avez placés dans la file d'attente WebSphere MQ .

Exécutez l'exemple **HTTPDELETE** en procédant comme suit:

1. Dans une fenêtre de commande, accédez à *MQ_INSTALLATION_PATH*/tools/samples. *MQ_INSTALLATION_PATH* représente le répertoire dans lequel WebSphere MQ est installé.
2. Exécutez l'exemple **HTTPDELETE** .

   ```
   java -classpath . HTTPPOST [parameters]
   ```

   Lorsque l'exemple **HTTPDELETE** démarre, la sortie suivante s'affiche:

```
HTTP DELETE Sample start
Target server is 'host:port'
Target queue is 'your queue name'
Target context-root is 'your context-root'
message
message
...
```

Remarques relatives à la sécurité pour le pont WebSphere pour HTTP

Les considérations de sécurité Web standard s'appliquent à l'authentification d'un client de navigation Web. L'autorisation d'accès aux ressources WebSphere MQ est au niveau de l'utilisateur qui exécute le servlet WebSphere Bridge for HTTP, et non au niveau du client de navigateur Web individuel. Les considérations de sécurité WebSphere MQ standard s'appliquent à WebSphere MQ.

Le flux de données d'un navigateur Web vers une application WebSphere MQ à l'aide d'un pont WebSphere pour HTTP et versions ultérieures, s'effectue en trois étapes:

Connexion client
Depuis le navigateur vers WebSphere Bridge for HTTP sur une connexion TCP/IP utilisant HTTP.

Connexion de l'adaptateur de ressources à WebSphere MQ
La connexion est établie entre WebSphere Bridge for HTTP et un gestionnaire de files d'attente WebSphere MQ . La connexion est soit une connexion client, via TCP/IP, soit une connexion de liaisons WebSphere MQ locale. Une fois la connexion établie, la demande HTTP est placée dans une file d'attente locale standard ou dans une file d'attente de transmission.

De la file d'attente locale WebSphere MQ sur un ou plusieurs canaux vers la file d'attente cible.
Appliquez des techniques standard pour sécuriser les files d'attente, les rubriques, les gestionnaires de files d'attente et les canaux.

La réponse prend les étapes en sens inverse.

Connexion client

Sécurisez les connexions entre les clients HTTP et le serveur d'applications à l'aide du conteneur Web. Utilisez des techniques de serveur HTTP standard, telles que l'utilisation de HTTPS. Pour plus d'informations, reportez-vous à la documentation de votre serveur d'applications.

Connexion de l'adaptateur de ressources à WebSphere MQ

La connexion entre l'adaptateur de ressources et le gestionnaire de files d'attente est autorisée à l'aide d'un seul ID utilisateur. Affectez un ID utilisateur unique pour identifier les demandes provenant de WebSphere Bridge for HTTP. L'ID utilisateur doit avoir des autorisations WebSphere MQ restreintes uniquement aux ressources auxquelles les utilisateurs externes doivent avoir accès. Vous

devez authentifier le client réel séparément et établir une relation de confiance pour les interactions successives avec le client, à l'aide de techniques standard pour la sécurité Web.

Sécurisez la connexion entre l'adaptateur de ressources et le gestionnaire de files d'attente à l'aide de l'ID utilisateur unique. Limitez les droits dont dispose l'ID utilisateur à ceux dont il a besoin pour lire et écrire des messages dans les files d'attente et les rubriques. WebSphere Bridge for HTTP est un point d'attaque entre Internet et votre intranet.

La façon dont vous sécurisez la connexion entre votre adaptateur de ressources et WebSphere MQ dépend de votre adaptateur de ressources spécifique. Voir la documentation de l'adaptateur de ressources.

Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX)

Les WebSphere MQ Automation Classes for ActiveX (MQAX) sont des composants ActiveX qui fournissent des classes que vous pouvez utiliser dans votre application pour accéder à WebSphere MQ.

MQAX requiert un environnement WebSphere MQ et une application WebSphere MQ correspondante avec laquelle communiquer.

Il permet à votre application ActiveX d'exécuter des transactions et d'accéder aux données sur n'importe quel système d'entreprise auquel vous pouvez accéder via WebSphere MQ.

WebSphere MQ Automation Classes for ActiveX:

- Vous permet d'accéder aux fonctions et aux fonctions de l'API WebSphere MQ , ce qui permet une interconnectivité complète avec d'autres plateformes WebSphere MQ .
- Conformez aux conventions normales attendues d'un composant ActiveX .
- Conformez au modèle d'objet WebSphere MQ , également disponible pour .NET, C + +, Java et LotusScript.

Des exemples de module de démarrage MQAX sont fournis. Vous pouvez utiliser ces exemples initialement pour vérifier que votre installation de MQAX a abouti et que vous disposez de l'environnement WebSphere MQ de base. Les exemples montrent également comment MQAX peut être utilisé.

Scriptage COM et ActiveX

Le modèle COM (Component Object Model) est un modèle de programmation basé sur des objets défini par Microsoft. Il indique comment les composants logiciels peuvent être fournis de manière à leur permettre de se localiser et de communiquer entre eux, quel que soit le langage informatique dans lequel ils sont écrits ou leur emplacement.

ActiveX est un ensemble de technologies, basé sur COM, qui intègre le développement d'applications, les composants réutilisables et les technologies Internet sur les plateformes Microsoft Windows . Les composants ActiveX fournissent des interfaces accessibles de manière dynamique par les applications. Un client de scriptage ActiveX est une application, par exemple un compilateur, qui peut générer ou exécuter un programme ou un script qui utilise les interfaces fournies par les composants ActiveX (ou COM).

WebSphere MQ

WebSphere MQ Automation Classes for ActiveX ne peut être utilisé qu'avec des clients de scriptage **32 bits** ActiveX .

Le composant COM ne peut être utilisé que pour les applications **32 bits** . Si vous souhaitez écrire une application COM 64 bits, vous pouvez utiliser l'interface .NET.

Pour exécuter MQAX dans un environnement de serveur WebSphere MQ , Windows 2000 ou version ultérieure doit être installé sur votre système.

Pour exécuter MQAX dans un environnement client MQI WebSphere MQ , vous avez besoin du client MQI WebSphere MQ sous Windows 2000 ou version ultérieure installé sur votre système:

Le client WebSphere MQ MQI requiert l'accès à au moins un serveur WebSphere MQ . Lorsque le client WebSphere MQ MQI et le serveur WebSphere MQ sont installés sur votre système, les applications MQAX s'exécutent toujours sur le serveur. L'interface ActiveX de MQAI est uniquement disponible dans les environnements de serveur WebSphere MQ .

Conception et programmation à l'aide de WebSphere MQ Automation Classes for ActiveX

Conception d'applications MQAX qui accèdent à des applications nonActiveX

Les classes d'automatisation WebSphere MQ permettent d'accéder aux fonctions de l'API WebSphere MQ . Vous pouvez donc bénéficier de tous les avantages que l'utilisation de WebSphere MQ peut apporter à votre application Windows .

La conception globale de votre application est la même que pour toute application WebSphere MQ . Par conséquent, prenez en compte tous les aspects de conception décrits dans la section «Développement d'applications», à la page 7 .

Pour utiliser les classes d'automatisation WebSphere MQ , vous devez coder les programmes Windows dans votre application à l'aide d'un langage qui prend en charge la création et l'utilisation d'objets COM. Par exemple, Visual Basic, Java et d'autres clients de scriptage ActiveX . Les classes peuvent ensuite être facilement intégrées à votre application car les objets WebSphere MQ dont vous avez besoin peuvent être codés à l'aide de la syntaxe native du langage d'implémentation.

Utilisation de WebSphere MQ Automation Classes for ActiveX

Lors de la conception d'une application ActiveX qui utilise WebSphere MQ Automation Classes for ActiveX, l'élément d'information le plus important est le message envoyé ou reçu depuis le système WebSphere MQ distant. Par conséquent, vous devez connaître le format des éléments insérés dans le message. Pour un script MQAX destiné à un travail, ce dernier et l'application WebSphere MQ qui récupère ou envoie le message doivent connaître la structure du message.

Si vous envoyez un message avec une application MQAX et que vous souhaitez effectuer une conversion de données à l'extrémité MQAX, vous devez également savoir:

- Page de codes utilisée par le système distant
- Codage utilisé par le système distant

Pour conserver votre code portable, il est recommandé de définir la page de codes et le codage, même s'ils sont actuellement identiques dans les deux systèmes d'envoi et de réception.

Lorsque vous envisagez de structurer l'implémentation du système que vous concevez, n'oubliez pas que vos scripts MQAX s'exécutent sur la même machine que celle sur laquelle le gestionnaire de files d'attente WebSphere MQ ou le client WebSphere MQ est installé.

Conseils et astuces pour la programmation

Les conseils et astuces suivants ne sont pas dans un ordre significatif. Ce sont des sujets qui, s'ils sont pertinents pour le travail que vous faites, pourraient vous faire gagner du temps.

Propriétés du descripteur de message

Si vous manipulez des propriétés de descripteur de message dans un programme, il est préférable d'utiliser les équivalents hexadécimaux des zones.

Les informations de cette section font référence aux propriétés suivantes:

- AccountingToken

- CorrelationId
- GroupId
- MessageId

Lorsqu'une application WebSphere MQ est à l'origine d'un message et que WebSphere MQ génère ces propriétés, il est préférable d'utiliser les propriétés hexadécimales AccountingToken, CorrelationIdHex, GroupIdHex et MessageIdHex si vous souhaitez examiner leurs valeurs ou les manipuler de quelque manière que ce soit, inclusion de leur renvoi dans un message à WebSphere MQ. En effet, les valeurs générées par WebSphere MQ sont des chaînes d'octets dont la valeur est comprise entre 0 et 255 inclus et ne sont pas des chaînes de caractères imprimables.

Lorsque votre script MQAX est à l'origine d'un message, vous pouvez utiliser les propriétés AccountingToken, CorrelationId, GroupIdet MessageId ou leurs équivalents hexadécimal.

Constantes WebSphere MQ

WebSphere Les constantes MQ sont fournies en tant que membres de l'énumération WebSphere MQ dans la bibliothèque MQAX200.

Constantes de chaîne WebSphere MQ

WebSphere MQ constantes de chaîne et leurs chaînes de caractères correspondantes.

Les constantes de chaîne WebSphere MQ ne sont pas disponibles lors de l'utilisation de WebSphere MQ Automation Classes for ActiveX. Vous devez utiliser la chaîne de caractères explicite pour les éléments affichés dans la liste suivante et pour les autres éléments dont vous pourriez avoir besoin. Les commandes doivent être remplies à huit caractères à l'aide d'espaces:

MQFMT_AUCUN	" "
MQFMT_ADMIN	"MQADMIN"
MQFMT_CHANNEL_COMPLETED	"MQCHCOM"
MQFMT_CICS	"MQCICS"
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD"
MQFMT_DIST_HEADER	"MQHDIST"
MQFMT_EVENT	"MQEVENT"
MQFMT_IMS	"MQIMS"
MQFMT_IMS_VAR_STRIsNG	"MQIMSVS"
MQFMT_MD_EXTENSION	"MQHMDE"
MQFMT_PCF	"MQPCF"
MQFMT_REF_MSG_HEADER	"MQHREF"
MQFMT_RF_HEADER	"MQHRF"
MQFMT_STRING	"MQSTR"
MQFMT_TRIGGER	"MQTRIG"
MQFMT_WORK_INFO_HEADER	"MQHWIH"
MQFMT_XMIT_Q_HEADER	"MQXMIT"

Constantes de chaîne null

Les constantes WebSphere MQ , utilisées pour l'initialisation de quatre propriétés MQMessage, MQMI_NONE (24 caractères NULL), MQCI_NONE (24 caractères NULL), MQGI_NONE (24 caractères NULL) et MQACT_NONE (32 caractères NULL), ne sont pas prises en charge par WebSphere MQ Automation Classes for ActiveX. Leur définition sur des chaînes vides a le même effet.

Par exemple, pour définir les différents ID d'un MQMessage sur les valeurs suivantes: *mymessage*.**MessageId** = "" *mymessage*.**CorrelationId** = "" *mymessage*.**AccountingToken** = ""

Réception d'un message de WebSphere MQ

Il existe plusieurs façons de recevoir un message de WebSphere MQ:

- Interrogation en émettant une requête GET suivie d'une requête Wait, à l'aide de la fonction Visual Basic TIMER.
- Emission d'une commande GET avec l'option Attendre ; vous spécifiez la durée d'attente en définissant la propriété WaitInterval . Tenez compte de cette situation lorsque, même si vous avez configuré votre système pour qu'il s'exécute dans un environnement à unités d'exécution multiples, le logiciel exécuté à ce moment-là peut ne s'exécuter qu'avec une seule unité d'exécution. Cela permet d'éviter que votre système ne se verrouille indéfiniment.

  Les autres unités d'exécution ne sont pas affectées. Toutefois, si vos autres unités d'exécution ont besoin d'accéder à WebSphere MQ, elles ont besoin d'une deuxième connexion à WebSphere MQ à l'aide d'objets de file d'attente et de gestionnaire de files d'attente MQAX supplémentaires.

L'émission d'une opération GET avec l'option Wait et la définition de la valeur MQWI_UNLIMITED pour WaitInterval entraîne le verrouillage de votre système jusqu'à la fin de l'appel GET, si le processus est à unité d'exécution unique.

Utilisation de la conversion de données

Deux formes de conversion de données sont prises en charge par WebSphere MQ Automation Classes for ActiveX -codage numérique et conversion de jeu de caractères.

Codage numérique

Si vous définissez la propriété MQMessage Encoding, les méthodes suivantes sont converties entre différents systèmes de codage numérique:

- Méthode ReadDecimal2
- Méthode ReadDecimal4
- Méthode ReadDouble
- Méthode ReadDouble4
- Méthode ReadFloat
- Méthode ReadInt2
- Méthode ReadInt4
- Méthode ReadLong
- Méthode ReadShort
- Méthode ReadUInt2
- Méthode WriteDecimal2
- Méthode WriteDecimal4
- Méthode WriteDouble
- Méthode WriteDouble4
- Méthode WriteFloat

- Méthode WriteInt2
- Méthode WriteInt4
- Méthode WriteLong
- Méthode WriteShort
- Méthode WriteUInt2

La propriété Codage peut être définie et interprétée à l'aide des constantes WebSphere MQ fournies. Figure 216, à la page 1069 montre un exemple de ces éléments:

```
/* Encodings for Binary Integers */
        MQENC_INTEGER_UNDEFINED
        MQENC_INTEGER_NORMAL
        MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
        MQENC_DECIMAL_UNDEFINED
        MQENC_DECIMAL_NORMAL
        MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
        MQENC_FLOAT_UNDEFINED
        MQENC_FLOAT_IEEE_NORMAL
        MQENC_FLOAT_IEEE_REVERSED
        MQENC_FLOAT_S390
```

Figure 216. Constantes WebSphere MQ fournies pour le codage

Par exemple, pour envoyer un entier d'un système Intel à un système d'exploitation System/390 dans le codage System/390 :

```
Dim msg As New MQMessage   'Define a WebSphere MQ message for our use..
Print msg.Encoding    'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg.Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg.Encoding    'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg.WriteLong(local_num)         'Write the number into the message
```

Conversion de jeu de caractères

La conversion du jeu de caractères est nécessaire lorsque vous envoyez un message d'un système à un autre système où les pages de codes sont différentes. La conversion de page de codes est utilisée par:

- Méthode ReadString
- Méthode ReadNullTerminatedString
- Méthode WriteString
- Méthode WriteNullTerminatedString
- Propriété MessageData

Vous devez définir la propriété MQMessage CharacterSet sur une valeur de jeu de caractères (CCSID) prise en charge.

WebSphere MQ Automation Classes for ActiveX utilise des tables de conversion pour effectuer une conversion de jeu de caractères.

Par exemple, pour convertir automatiquement des chaînes en page de codes 437:

```
Dim msg As New MQMessage                 'Define a WebSphere MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

La méthode WriteString reçoit les données de chaîne ("A character string" dans l'exemple) sous la forme d'une chaîne Unicode. Il convertit ensuite ces données d'Unicode en page de codes 437 à l'aide de la table de conversion 34B001B5.TBL.

Les caractères de la chaîne Unicode qui ne sont pas pris en charge par la page de codes 437 reçoivent le caractère de substitution standard de la page de codes 437.

De même, lorsque vous utilisez la méthode ReadString , le message entrant possède un jeu de caractères établi par la valeur WebSphere MQ Message Descriptor (MQMD) et il y a une conversion de cette page de codes en Unicode avant qu'elle ne soit retransmise à votre langage de script.

Utilisation d'unités d'exécution

WebSphere MQ Automation Classes for ActiveX implémente un modèle d'unités d'exécution libres dans lequel des objets peuvent être utilisés entre les unités d'exécution.

Alors que MQAX autorise l'utilisation des objets MQQueue et MQQueueManager , WebSphere MQ ne permet pas actuellement le partage de descripteurs entre différentes unités d'exécution.

Les tentatives d'utilisation de ces éléments sur une autre unité d'exécution génèrent une erreur et WebSphere MQ renvoie le code retour MQRC_HCONN_ERROR.

Remarque : Il n'existe qu'un seul objet MQSession par processus. L'utilisation de MQSession CompletionCode et ReasonCode n'est pas recommandée dans les environnements à unités d'exécution multiples. Les valeurs d'erreur MQSession peuvent être écrasées par une seconde unité d'exécution entre une erreur générée et vérifiée sur la première unité d'exécution. Les unités d'exécution sont sérialisées pour la durée de chaque appel de méthode ou accès aux propriétés. Par conséquent, l'émission d'une commande Get avec l'option Wait entraîne l'interruption des autres unités d'exécution accédant aux objets MQAX jusqu'à la fin de l'opération.

Traitement des erreurs

Ces informations décrivent les propriétés de l'objet MQAX, le fonctionnement du traitement des erreurs, les règles décrivant le traitement des exceptions et l'obtention d'une propriété.

Chaque objet MQAX inclut des propriétés pour stocker les informations d'erreur et une méthode pour les réinitialiser ou les effacer. Les propriétés sont les suivantes :

- CompletionCode
- ReasonCode
- ReasonName

La méthode est la suivante:

- Codes ClearError

Fonctionnement du traitement des erreurs

Votre script ou application MQAX appelle la méthode d'un objet MQAX ou accède à une propriété de l'objet MQAX ou la met à jour:

1. Les codes ReasonCode et CompletionCode de l'objet concerné sont mis à jour.
2. Les codes ReasonCode et CompletionCode de l'objet MQSession sont également mis à jour avec les mêmes informations.

   Remarque : Pour connaître les restrictions d'utilisation des codes d'erreur MQSession dans les applications à unités d'exécution, voir «Utilisation d'unités d'exécution», à la page 1070 .

   Si CompletionCode est supérieur ou égal à la propriété ExceptionThreshold de MQSession, MQAX émet une exception (numéro 32000). Utilisez-le dans votre script à l'aide de l'instruction On Error (ou équivalente) pour le traiter.
3. Utilisez la fonction Erreur pour extraire la chaîne d'erreur associée, qui se présente sous la forme suivante:

```
MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
```

Pour plus d'informations sur l'utilisation des instructions On Error, voir la documentation de votre langage de script ActiveX .

L'utilisation de CompletionCode et ReasonCode dans l'objet MQSession est pratique pour les gestionnaires d'erreurs simples.

La propriété ReasonName renvoie le nom symbolique WebSphere MQ pour la valeur en cours de ReasonCode.

Création d'exceptions

Les règles suivantes décrivent la façon dont les exceptions sont traitées:

- Chaque fois qu'une propriété ou une méthode définit le code achèvement sur une valeur supérieure ou égale au seuil d'exception (généralement définie sur 2), une exception est émise.
- Tous les appels de méthode et les ensembles de propriétés définissent le code achèvement.

Obtention d'une propriété

Il s'agit d'un cas particulier car les codes CompletionCode et ReasonCode ne sont pas toujours mis à jour:

- Si une propriété aboutit, l'objet et l'objet MQSession ReasonCode et CompletionCode restent inchangés.
- Si une propriété get échoue avec un CompletionCode d'avertissement, les valeurs ReasonCode et CompletionCode restent inchangées.
- Si une propriété get échoue avec un code d'erreur CompletionCode , les codes ReasonCode et CompletionCode sont mis à jour pour refléter les valeurs true et le traitement des erreurs se poursuit comme décrit.

La classe MQSession comporte une méthode *ReasonCodeName* qui peut être utilisée pour remplacer un code anomalie WebSphere MQ par un nom symbolique. Cela est particulièrement utile lors du développement de programmes dans lesquels des erreurs inattendues peuvent se produire. Cependant, le nom n'est pas idéal pour la présentation aux utilisateurs.

Chaque classe possède également une propriété *ReasonName*, qui renvoie le nom symbolique du code anomalie en cours pour cette classe.

Référence WebSphere MQ Automation Classes for ActiveX

Cette section décrit les classes de WebSphere MQ Automation Classes for ActiveX (MQAX), développées pour ActiveX. Les classes vous permettent d'écrire des applications ActiveX qui peuvent accéder à d'autres applications s'exécutant dans des environnements nonActiveX , à l'aide de WebSphere MQ.

Interface WebSphere MQ Automation Classes for ActiveX

WebSphere MQ Automation Classes for ActiveX fournit des constantes ActiveX numériques prédéfinies (telles que MQMT_REQUEST) nécessaires à l'utilisation des classes.

Les classes d'automatisation ActiveX sont les suivantes:

- «Classe MQSession», à la page 1073
- «Classe MQQueueManager», à la page 1076
- «classe MQQueue», à la page 1087
- «Classe MQMessage», à la page 1102
- «Classe MQPutMessageOptions», à la page 1124
- «Classe d'options MQGetMessage», à la page 1126
- «Classe MQDistributionList», à la page 1128

- «Classe d'élément MQDistributionList», à la page 1132

En outre, WebSphere MQ Automation Classes for ActiveX fournit des constantes ActiveX numériques prédéfinies (telles que MQMT_REQUEST) nécessaires à l'utilisation des classes. Ils sont fournis dans l'énumération MQ dans la bibliothèque MQAX200. Les constantes sont un sous-ensemble de celles définies dans les fichiers d'en-tête C WebSphere MQ (cmqc * .h) avec des codes anomalie WebSphere MQ Automation Classes for ActiveX supplémentaires.

A propos des classes WebSphere MQ Automation pour les classes ActiveX

Lisez ces informations en même temps que les rubriques de référence de la rubrique Développement de références d'applications.

Pour plus d'informations, voir Fonctions disponibles uniquement avec l'installation principale sous Windows .

La classe MQSession fournit un objet racine qui contient le statut de la dernière action effectuée sur l'un des objets MQAX. Pour plus d'informations, voir «Traitement des erreurs», à la page 1070.

Les classes MQQueueManager et MQQueue permettent d'accéder aux objets WebSphere MQ sous-jacents. Les méthodes ou les accès aux propriétés de ces classes entraînent généralement des appels dans l'interface MQI WebSphere MQ .

Les classes MQMessage, MQPutMessageOptions et MQGetMessageOptions encapsulent les structures de données MQMD, MQPMO et MQGMO et sont utilisées pour vous aider à envoyer des messages aux files d'attente et à en extraire des messages.

La classe MQDistributionList encapsule une collection de files d'attente (locales, distantes ou alias) pour la sortie. La classe d'élément MQDistributionListencapsule les structures MQOR, MQRR et MQPMR et les associe à une liste de distribution propriétaire.

Transmission de paramètres

Les paramètres des appels de méthode sont tous transmis par valeur, sauf lorsque ce paramètre est un objet, auquel cas il s'agit d'une référence transmise.

Les définitions de classe fournies répertorient le type de données pour chaque paramètre ou propriété. Pour de nombreux clients ActiveX , tels que Visual Basic, si la variable utilisée n'est pas du type requis, la valeur est automatiquement convertie vers ou depuis le type requis, à condition qu'une telle conversion soit possible. Cela suit les règles standard du client ; MQAX ne fournit aucune conversion de ce type.

De nombreuses méthodes utilisent des paramètres de chaîne de longueur fixe ou renvoient une chaîne de caractères de longueur fixe. Les règles de conversion sont les suivantes:

- Si l'utilisateur fournit une chaîne de longueur fixe de longueur incorrecte, en tant que paramètre d'entrée ou en tant que valeur de retour, la valeur est tronquée ou complétée avec des espaces de fin, selon les besoins.
- Si l'utilisateur fournit une chaîne de longueur variable de longueur incorrecte en tant que paramètre d'entrée, la valeur est tronquée ou complétée avec des espaces de fin.
- Si l'utilisateur fournit une chaîne de longueur variable de longueur incorrecte en tant que valeur de retour, la chaîne est ajustée à la longueur requise (car le renvoi d'une valeur détruit de toute façon la valeur précédente de la chaîne).
- Les chaînes fournies en tant que paramètres d'entrée peuvent contenir des valeurs nulles imbriquées.

Ces classes sont disponibles dans la bibliothèque MQAX200 .

Méthodes d'accès aux objets

Ces méthodes ne sont pas directement liées à un seul appel WebSphere MQ . Chacune de ces méthodes crée un objet dans lequel sont conservées les informations de référence, puis se connecte à ou ouvre un objet WebSphere MQ :

Lorsqu'une connexion est établie à un gestionnaire de files d'attente, il contient l'attribut'descripteur de connexion'généré par WebSphere MQ.

Lorsqu'une file d'attente est ouverte, elle contient l'attribut'descripteur d'objet'généré par WebSphere MQ.

Ces attributs WebSphere MQ ne sont pas directement disponibles pour le programme MQAX.

Erreurs

Les erreurs de syntaxe lors de la transmission des paramètres peuvent être détectées lors de la compilation et de l'exécution par le client ActiveX . Les erreurs peuvent être interceptées à l'aide de On Error dans Visual Basic.

Les classes WebSphere MQ ActiveX contiennent toutes deux propriétés spéciales en lecture seule: ReasonCode et CompletionCode. Ces propriétés peuvent être lues à tout moment.

Une tentative d'accès à une autre propriété ou d'émission d'un appel de méthode peut générer une erreur à partir de WebSphere MQ.

Si un ensemble de propriétés ou un appel de méthode aboutit, le ReasonCode de l'objet propriétaire est défini sur MQRC_NONE et CompletionCode est défini sur MQCC_OK.

Si l'accès à la propriété ou l'appel de méthode n'aboutit pas, les codes anomalie et d'achèvement sont définis dans ces zones.

Classe MQSession

Il s'agit de la classe racine pour WebSphere MQ Automation Classes for ActiveX.

Il n'existe toujours qu'un seul objet MQSession par processus client ActiveX . Une tentative de création d'un deuxième objet crée une deuxième référence à l'objet d'origine.

Création

Nouveau crée un nouvel objet MQSession.

Syntaxe

Dim *mqsess* **As New MQSession Set** *mqsess* = **New MQSession**

Propriétés

- «Propriété CompletionCode», à la page 1073.
- «Propriété ExceptionThreshold», à la page 1074.
- «Propriété ReasonCode», à la page 1074.
- «Propriété ReasonName», à la page 1075.

Méthode

- «Méthode AccessGetMessageOptions», à la page 1075.
- «Méthode AccessMessage», à la page 1075.
- «Méthode AccessPutMessageOptions», à la page 1075.
- «Méthode du gestionnaire AccessQueue», à la page 1075.
- «Méthode des codes ClearError», à la page 1076.
- «Méthode de nom ReasonCode», à la page 1076.

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement WebSphere MQ défini par la méthode ou l'ensemble de propriétés le plus récent émis pour tout objet WebSphere MQ .

Elle est réinitialisée sur MQCC_OK lorsqu'une méthode ou un ensemble de propriétés est appelé avec succès sur un objet MQAX.

Un gestionnaire d'événements Erreur peut inspecter cette propriété pour diagnostiquer l'erreur, sans avoir à savoir quel objet a été impliqué.

L'utilisation de CompletionCode et ReasonCode dans l'objet MQSession est très pratique pour les gestionnaires d'erreurs simples.

Remarque : Pour connaître les restrictions d'utilisation des codes d'erreur MQSession dans les applications à unités d'exécution, voir «Utilisation d'unités d'exécution», à la page 1070 .

Défini dans: Classe MQSession

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe :

Pour obtenir: *completioncode & = MQSession*.CompletionCode

Propriété ExceptionThreshold

Lecture-écriture. Définit le niveau de l'erreur WebSphere MQ pour laquelle MQAX émet une exception. La valeur par défaut est MQCC_FAILED. Une valeur supérieure à MQCC_FAILED empêche efficacement le traitement des exceptions, ce qui permet au programmeur d'effectuer des vérifications sur CompletionCode et ReasonCode.

Défini dans: Classe MQSession

Type de données: Long

Valeurs :

- Prenez en compte MQCC_WARNING, MQCC_FAILED ou une valeur supérieure.

Syntaxe :

Pour obtenir: *ExceptionThreshold& = MQSession*.**ExceptionThreshold**

A définir: *MQSession.***ExceptionThreshold** = *ExceptionThreshold$*

Propriété ReasonCode

Lecture seule. Renvoie le code anomalie défini par la méthode ou l'ensemble de propriétés le plus récent émis pour tout objet WebSphere MQ .

Un gestionnaire d'événements Erreur peut inspecter cette propriété pour diagnostiquer l'erreur, sans avoir à savoir quel objet a été impliqué.

L'utilisation de CompletionCode et ReasonCode dans l'objet MQSession est très pratique pour les gestionnaires d'erreurs simples.

Remarque : Pour connaître les restrictions d'utilisation des codes d'erreur MQSession dans les applications à unités d'exécution, voir «Utilisation d'unités d'exécution», à la page 1070 .

Défini dans: Classe MQSession

Type de données: Long

Valeurs :

- Voir Raison (MQLONG) et les valeurs MQAX supplémentaires répertoriées sous «Codes raison», à la page 1140.

Syntaxe: Pour obtenir: *reasoncode & = MQSession*.**ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Remarque : Pour connaître les restrictions d'utilisation des codes d'erreur MQSession dans les applications à unités d'exécution, voir «Utilisation d'unités d'exécution», à la page 1070 .

Défini dans: Classe MQSession

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQSession*.**ReasonName**

Méthode AccessGetMessageOptions

Crée un nouvel objet d'options MQGetMessage.

Défini dans: Classe MQSession

Syntaxe: *gmo = MQSession*.**AccessGetMessageOptions()**

Méthode AccessMessage

Crée un objet MQMessage.

Défini dans: Classe MQSession

Syntaxe: *msg = MQSession*.**AccessMessage()**

Méthode AccessPutMessageOptions

Crée un nouvel objet d'options MQPutMessage.

Défini dans: Classe MQSession

Syntaxe: *pmo = MQSession*.**AccessPutMessageOptions()**

Méthode du gestionnaire AccessQueue

Crée un objet MQQueueManager et le connecte à un gestionnaire de files d'attente réel à l'aide du client WebSphere MQ MQI ou du serveur WebSphere MQ . En plus d'effectuer une connexion, cette méthode effectue également une ouverture pour l'objet gestionnaire de files d'attente.

Lorsque le client WebSphere MQ MQI et le serveur WebSphere MQ sont installés sur votre système, les applications MQAX s'exécutent sur le serveur par défaut. Pour exécuter MQAX sur le client, la bibliothèque de liaisons du client doit être spécifiée dans la variable d'environnement `GMQ_MQ_LIB` , par exemple `GMQ_MQ_LIB=mqic.dll`.

Pour une installation client uniquement, il n'est pas nécessaire de définir la variable d'environnement `GMQ_MQ_LIB` . Lorsque cette variable n'est pas définie, WebSphere MQ tente de charger amqzst.dll. Si cette DLL n'est pas présente (comme dans une installation client uniquement), WebSphere MQ tente de charger mqic.dll.

Si elle aboutit, elle définit le ConnectionStatus de MQQueueManagersur TRUE.

Un gestionnaire de files d'attente peut être connecté à au plus un objet MQQueueManager par instance ActiveX .

Si la connexion au gestionnaire de files d'attente échoue, un événement d'erreur est émis et les attributs ReasonCode et CompletionCode de l'objet MQSession sont définis.

Défini dans: Classe MQSession

Syntaxe: *set qm = MQSession*.**AccessQueueManager** (*Name$*)

Paramètre:*Name$* Chaîne. Nom du gestionnaire de files d'attente auquel la connexion doit être établie.

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE.

Défini dans: Classe MQSession

Syntaxe: Call *MQSession*.**ClearErrorCodes ()**

Méthode de nom ReasonCode

Renvoie le nom du code raison avec la valeur numérique indiquée. Il est utile de donner des indications plus claires sur les conditions d'erreur aux utilisateurs. Le nom est encore quelque peu cryptique (par exemple, ReasonCodeName (2059) is **MQRC_Q_MGR_NOT_AVAILABLE**), de sorte que les erreurs éventuelles doivent être interceptées et remplacées par un texte descriptif approprié à l'application.

Défini dans: Classe MQSession

Syntaxe: *errname $ = MQSession*.**ReasonCodeName**(*reasonCode&*)

Paramètre:*code anomalie &* Long. Code anomalie pour lequel le nom symbolique est requis.

Classe MQQueueManager

Cette classe représente une connexion à un gestionnaire de files d'attente. Le gestionnaire de files d'attente peut être exécuté en local (serveur WebSphere MQ) ou à distance avec un accès fourni par le client WebSphere MQ . Une application doit créer un objet de cette classe et le connecter à un gestionnaire de files d'attente. Lorsqu'un objet de cette classe est détruit, il est automatiquement déconnecté de son gestionnaire de files d'attente.

Confinement

Les objets de classe MQQueue sont associés à cette classe.

Un nouvel objet MQQueueManager est créé et toutes les propriétés sont associées à des valeurs initiales. Vous pouvez également utiliser la méthode AccessQueueManager de la classe MQSession.

Création

New crée un objet **new** MQQueueManager et affecte à toutes les propriétés des valeurs initiales. Vous pouvez également utiliser la méthode AccessQueueManager de la classe MQSession.

Syntaxe

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

Propriétés

- «Propriété d'ID AlternateUser», à la page 1078.
- «Propriété AuthorityEvent», à la page 1078.
- «Propriété BeginOptions», à la page 1078.
- «Propriété de définition ChannelAuto», à la page 1079.
- «Propriété ChannelAutoDefinitionEvent», à la page 1079.

- «Propriété ChannelAutoDefinitionExit», à la page 1079.
- «Propriété CharacterSet», à la page 1079.
- «Propriété CloseOptions», à la page 1079.
- «Propriété CommandInputQueueName», à la page 1080.
- «Propriété CommandLevel», à la page 1080.
- «Propriété CompletionCode», à la page 1080.
- «Propriété ConnectionHandle», à la page 1080.
- «Propriété ConnectionStatus», à la page 1080.
- «Propriété ConnectOptions», à la page 1081.
- «Propriété DeadLetterQueueName», à la page 1081.
- «DefaultTransmissionPropriétéQueueName», à la page 1081.
- «Propriété Description», à la page 1081.
- «Propriété DistributionLists», à la page 1081.
- «Propriété InhibitEvent», à la page 1082.
- «Propriété IsConnected», à la page 1082.
- «Propriété IsOpen», à la page 1082.
- «Propriété LocalEvent», à la page 1082.
- «Propriété MaximumHandles», à la page 1083.
- «Propriété MaximumMessageLength», à la page 1083.
- «Propriété MaximumPriority», à la page 1083.
- «Propriété MaximumUncommittedMessages», à la page 1083.
- «propriété Name», à la page 1083.
- «Propriété ObjectHandle», à la page 1083.
- «Propriété PerformanceEvent», à la page 1084.
- «Propriété de plateforme», à la page 1084.
- «Propriété ReasonCode», à la page 1084.
- «Propriété ReasonName», à la page 1084.
- «Propriété RemoteEvent», à la page 1084.
- «Propriété d'événement StartStop», à la page 1085.
- «Propriété de disponibilité SyncPoint», à la page 1085.
- «Propriété TriggerInterval», à la page 1085.

Des méthodes

- «Méthode AccessQueue», à la page 1085.
- «Méthode de liste AddDistribution», à la page 1086.
- «Méthode d'annulation», à la page 1086.
- «Méthode de début», à la page 1086.
- «Méthode des codes ClearError», à la page 1086.
- «Méthode de validation», à la page 1087.
- «Méthode de connexion», à la page 1087.
- «Méthode de déconnexion», à la page 1087.

Accès aux propriétés

Les propriétés suivantes sont accessibles à tout moment.

- «Propriété d'ID AlternateUser», à la page 1078.
- «Propriété CompletionCode», à la page 1080.
- «Propriété ConnectionStatus», à la page 1080.
- «Propriété ReasonCode», à la page 1084.

Les propriétés restantes ne sont accessibles que si l'objet est connecté à un gestionnaire de files d'attente et que l'ID utilisateur est autorisé à interroger ce gestionnaire de files d'attente. Si un ID utilisateur de remplacement est défini et que l'ID utilisateur en cours est autorisé à l'utiliser, l'ID utilisateur de remplacement est vérifié pour l'autorisation d'interrogation à la place.

Si ces conditions ne s'appliquent pas, WebSphere MQ Automation Classes for ActiveX tente de se connecter au gestionnaire de files d'attente et de l'ouvrir pour l'interroger automatiquement. En cas d'échec, l'appel définit un CompletionCode de MQCC_FAILED et l'un des ReasonCodessuivants:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

Propriété d'ID AlternateUser

Lecture-écriture. ID utilisateur de remplacement à utiliser pour valider l'accès aux attributs du gestionnaire de files d'attente.

Cette propriété ne doit pas être définie si IsConnected a pour valeur TRUE.

Cette propriété ne peut pas être définie lorsque l'objet est ouvert.

Classe **Defined in:** MQQueueManager

Data Type: Chaîne de 12 caractères

Syntax: Pour obtenir: *altuser $ = MQQueueManager*.**AlternateUserId** A définir: *MQQueueManager*.**AlternateUserId** = *altuser $*

Propriété AuthorityEvent

Lecture seule. L'attribut MQI AuthorityEvent .

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *authevent = MQQueueManager*.**AuthorityEvent**

Propriété BeginOptions

Lecture-écriture. Il s'agit des options qui s'appliquent à la méthode Begin. Initialement MQBO_NONE.

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQBO_AUCUN

Syntaxe: Pour obtenir: *beginoptions & =MQQueueManager*.**BeginOptions**

A définir: *MQQueueManager*.**BeginOptions**=*beginoptions &*

Propriété de définition ChannelAuto

Lecture seule. Indique si la définition de canal automatique est autorisée.

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQCHAD_DISABLED
- MQCHAD_ENABLED

Syntaxe: Pour obtenir: *channelautodef & = MQQueueManager*.**ChannelAutoDéfinition**

Propriété ChannelAutoDefinitionEvent

Lecture seule. Cette option contrôle si les événements de définition de canal automatique sont générés.

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *channelautodefevent & =MQQueueManager*.**ChannelAutoDefinitionEvent**

Propriété ChannelAutoDefinitionExit

Lecture seule. Nom de l'exit utilisateur utilisé pour la définition de canal automatique.

Défini dans:
Classe MQQueueManager

Type de données :
String

Syntaxe: Pour obtenir: *channelautodefexit$= MQQueueManager*.**ChannelAutoDefinitionExit**

Propriété CharacterSet

Lecture seule. Attribut MQI CodedCharSetId .

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *characterset & = MQQueueManager*.**CharacterSet**

Propriété CloseOptions

Lecture-écriture. Options utilisées pour contrôler ce qui se passe lorsque le gestionnaire de files d'attente est fermé. La valeur initiale est MQCO_NONE.

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQCO_AUCUN

Syntaxe: Pour obtenir: *closeopt & = MQQueueManager*.**CloseOptions**

A définir: *MQQueueManager*.**CloseOptions** =*closeopt &*

Propriété CommandInputQueueName

Lecture seule. Attribut QName CommandInputde MQI.

Défini dans: classe MQQueueManager

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *commandinputqname $ = MQQueueManager*.**CommandInputQueueName**

Propriété CommandLevel

Lecture seule. Renvoie la version et le niveau de l'implémentation du gestionnaire de files d'attente WebSphere MQ (attribut MQI CommandLevel)

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *level & = MQQueueManager*.**CommandLevel**

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: To get: *completioncode & = MQQueueManager*.**CompletionCode**

Propriété ConnectionHandle

Lecture seule. Descripteur de connexion de l'objet gestionnaire de files d'attente WebSphere MQ .

Défini dans:
Classe MQQueueManager

Type de données :
Long

Syntaxe: Pour obtenir: *hconn & = MQQueueManager.***ConnectionHandle**

Propriété ConnectionStatus

Lecture seule. Indique si l'objet est connecté ou non à son gestionnaire de files d'attente.

Défini dans: classe MQQueueManager

Type de données: Booléen

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *status = MQQueueManager*.**ConnectionStatus**

Propriété ConnectOptions

Lecture-Ecriture. Ces options s'appliquent à la méthode Connect. Initialement MQCNO_NONE.

Défini dans:
Classe MQQueueManager

Type de données :
Long

Valeurs :

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_AUCUN

Syntaxe: To get: *connectoptions & =MQQueueManager*.**ConnectOptions**

A définir: *MQQueueManager*.**ConnectOptions**=*connectoptions &*

Propriété DeadLetterQueueName

Lecture seule. Attribut QName DeadLetterde MQI.

Défini dans: classe MQQueueManager

Type de données: chaîne de 48 caractères

Syntaxe: pour obtenir: *dlqname $ = MQQueueManager*.**DeadLetterQueueName**

DefaultTransmissionPropriétéQueueName

Lecture seule. Attribut QName DefXmitde MQI.

Défini dans: classe MQQueueManager

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *defxmitqname $ = MQQueueManager*.**DefaultTransmissionQueueName**

Propriété Description

Lecture seule. Attribut QMgrDesc MQI.

Défini dans: classe MQQueueManager

Type de données: chaîne de 64 caractères

Syntaxe: Pour obtenir: *description $ = MQQueueManager*.**Description**

Propriété DistributionLists

Lecture seule. Il s'agit de la capacité du gestionnaire de files d'attente à prendre en charge les listes de distribution.

Défini dans:
Classe MQQueueManager

Type de données :
Boolean

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *distributionlists= MQQueueManager*.**DistributionLists**

Propriété InhibitEvent

Lecture seule. Attribut InhibitEvent de l'interface MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *inhibevent & = MQQueueManager*.**InhibitEvent**

Propriété IsConnected

Valeur indiquant si le gestionnaire de files d'attente est actuellement connecté.

Lecture seule.

Défini dans: classe MQQueueManager

Type de données: Booléen

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *isconnected = MQQueueManager*.**IsConnected**

Propriété IsOpen

Valeur indiquant si le gestionnaire de files d'attente est actuellement ouvert pour l'interrogation.

Lecture seule.

Défini dans:
Classe MQQueueManager

Type de données :
Boolean

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *IsOpen = MQQueueManager*.**IsOpen**

Propriété LocalEvent

Lecture seule. Attribut LocalEvent de l'interface MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *localevent & = MQQueueManager*.**LocalEvent**

Propriété MaximumHandles

Lecture seule. Attribut MaxHandles de l'interface MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *maxdescripteurs & = MQQueueManager*.**MaximumHandles**

Propriété MaximumMessageLength

Lecture seule. Attribut MQI MaxMsgLongueur du gestionnaire de files d'attente.

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *maxmessagelength & = MQQueueManager*.**MaximumMessageLongueur**

Propriété MaximumPriority

Lecture seule. Attribut MaxPriority de l'interface MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *maxpriority & = MQQueueManager*.**MaximumPriority**

Propriété MaximumUncommittedMessages

Lecture seule. Attribut MQI MaxUncommitted.

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: Pour obtenir: *maxuncommitted & = MQQueueManager*.**MaximumUncommittedMessages**

propriété Name

Lecture-écriture. Attribut MQI QMgrName . Cette propriété ne peut pas être écrite une fois que MQQueueManager est connecté.

Défini dans: classe MQQueueManager

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *name $ = MQQueueManager*.**name**

A définir: *MQQueueManager*.**name** = *nom $*

Remarque : Visual Basic réserve la propriété "Name" à utiliser dans l'interface visuelle. Par conséquent, lorsque vous utilisez Visual Basic, utilisez des minuscules, c'est-à-dire "name".

Propriété ObjectHandle

Lecture seule. Descripteur d'objet de l'objet gestionnaire de files d'attente WebSphere MQ .

Défini dans:
Classe MQQueueManager

Type de données
Long

Syntaxe: To get: *hobj & = MQQueueManager.***ObjectHandle**

Propriété PerformanceEvent

Lecture seule. Attribut MQI PerformanceEvent .

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: To get: *perfevent & = MQQueueManager*.PerformanceEvent

Propriété de plateforme

Lecture seule. Attribut de plateforme MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

Syntaxe: Pour obtenir: *platform & = MQQueueManager***.Plateforme**

Propriété ReasonCode

Lecture seule. Renvoie le code raison défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: To get: *reasoncode & = MQQueueManager***.ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Défini dans: classe MQQueueManager

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQQueueManager***.ReasonName**

Propriété RemoteEvent

Lecture seule. Attribut MQI RemoteEvent .

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *remoteevent & = MQQueueManager*.**RemoteEvent**

Propriété d'événement StartStop

Lecture seule. Attribut d'événement MQI StartStop.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: To get: *strstpevent & = MQQueueManager*.**StartStopEvent**

Propriété de disponibilité SyncPoint

Lecture seule. Attribut SyncPoint de l'interface MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Valeurs :

- MQSP_XX_ENCODE_CASE_ONE disponible
- MQSP_NON DISPONIBLE

Syntaxe: Pour obtenir: *syncpointavailability & = MQQueueManager*.**SyncPointDisponibilité**

Propriété TriggerInterval

Lecture seule. Attribut TriggerInterval de MQI.

Défini dans: classe MQQueueManager

Type de données: Long

Syntaxe: To get: *trigint & = MQQueueManager*.**TriggerInterval**

Méthode AccessQueue

Crée un objet MQQueue et l'associe à cet objet MQQueueManager en définissant la propriété de référence de connexion de la file d'attente. Il définit les propriétés Name, OpenOptions, DynamicQueueName et AlternateUserId de l'objet MQQueue sur les valeurs fournies et tente de l'ouvrir.

Si l'ouverture échoue, l'appel échoue. Un événement d'erreur est émis sur l'objet. Les attributs ReasonCode et CompletionCode, ainsi que les attributs MQSession ReasonCode et CompletionCode de l'objet sont définis.

Les paramètres DynamicQueueName, QueueManagerName et AlternateUserId sont facultatifs et ont la valeur par défaut "".

L'option OpenOption MQOO_INQUIRE doit être spécifiée en plus des autres options si les propriétés de file d'attente doivent être lues.

Ne définissez pas le nom QueueManagerou définissez-le sur "" si la file d'attente à ouvrir est locale. Sinon, définissez-la sur le nom du gestionnaire de files d'attente éloignées propriétaire de la file d'attente et une tentative d'ouverture d'une définition locale de la file d'attente éloignée est effectuée. Pour plus d'informations sur la résolution de nom de file d'attente éloignée et l'attribution d'alias au gestionnaire de files d'attente, voir Que sont les alias? :

Si la propriété Nom est définie sur un nom de file d'attente modèle, indiquez le nom de la file d'attente dynamique à créer dans le paramètre DynamicQueueName$. Si la valeur fournie dans le paramètre DynamicQueueName$ est "", la valeur définie dans l'objet file d'attente et utilisée dans l'appel ouvert est "AMQ. *". Pour plus d'informations sur la désignation des files d'attente dynamiques, voir «Création de files d'attente dynamiques», à la page 233 .

Définition

Défini dans: classe MQQueueManager .

Syntaxe

Syntaxe: set queue = MQQueueManager.**AccessQueue**(Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Paramètres

Name$ Chaîne. Nom de la file d'attente WebSphere MQ .

OpenOptions: Long. Options à utiliser lors de l'ouverture de la file d'attente. Voir OpenOptions (MQLONG).

QueueManagerNom$ Chaîne. Nom du gestionnaire de files d'attente propriétaire de la file d'attente à ouvrir. La valeur "" implique que le gestionnaire de files d'attente est local.

*DynamicQueueNom$* Chaîne. Nom attribué à la file d'attente dynamique lors de l'ouverture de la file d'attente lorsque le paramètre Name$ spécifie une file d'attente modèle.

AlternateUserId$ Chaîne. ID utilisateur de remplacement utilisé pour valider l'accès lors de l'ouverture de la file d'attente.

Méthode de liste AddDistribution

Crée un nouvel objet MQDistributionList et définit sa référence de connexion au gestionnaire de files d'attente propriétaire.

Défini dans:
Classe MQQueueManager

Syntaxe: *set distributionlist* = **MQQueueManager.***AddDistributionListe*

Méthode d'annulation

Annule les insertions de message non validées et les extractions qui se sont produites dans le cadre d'une unité de travail depuis le dernier point de synchronisation.

Défini dans: classe MQQueueManager

Syntaxe: Call *MQQueueManager.***Annulation ()**

Méthode de début

Démarre une unité d'oeuvre coordonnée par le gestionnaire de files d'attente. Les options de début affectent le comportement de cette méthode.

Défini dans:
Classe MQQueueManager

Syntaxe: Call *MQQueueManager*.**Begin ()**

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQQueueManager et la classe MQSession.

Défini dans: classe MQQueueManager

Syntaxe: Call *MQQueueManager*.**ClearErrorCodes ()**

Méthode de validation

Valide les insertions et extractions de message qui se sont produites dans le cadre d'une unité de travail depuis le dernier point de synchronisation.

Défini dans: classe MQQueueManager

Syntaxe: Call *MQQueueManager*.**Validation ()**

Méthode de connexion

Connecte l'objet MQQueueManager à un gestionnaire de files d'attente réel via le client ou le serveur WebSphere MQ MQI. En plus d'établir la connexion, cette méthode ouvre également l'objet gestionnaire de files d'attente pour qu'il puisse être interrogé.

Définit IsConnected sur TRUE.

Un objet MQQueueManager au maximum par instance ActiveX est autorisé à se connecter à un gestionnaire de files d'attente.

Défini dans: classe MQQueueManager

Syntaxe: Call *MQQueueManager*.**Connexion ()**

Méthode de déconnexion

Déconnecte l'objet MQQueueManager du gestionnaire de files d'attente.

Définit IsConnected sur FALSE.

Tous les objets de file d'attente associés à l'objet MQQueueManager sont rendus inutilisables et ne peuvent pas être rouverts.

Toutes les modifications non validées (insertions et extractions de messages) sont validées.

Défini dans: classe MQQueueManager

Syntaxe: Call *MQQueueManager*.**Déconnexion ()**

classe MQQueue

Cette classe représente l'accès à une file d'attente WebSphere MQ . Cette connexion est fournie par un objet MQQueueManager associé. Lorsqu'un objet de cette classe est détruit, il est automatiquement fermé.

Confinement

La classe MQQueue est contenue dans la classe MQQueueManager .

Création

New crée un nouvel objet MQQueue et définit toutes les propriétés sur des valeurs initiales. Vous pouvez également utiliser la méthode AccessQueue de la classe MQQueueManager .

Syntaxe

```
Dim que As New MQQueue Set que  = New MQQueue
```

Propriétés

- «Propriété d'ID AlternateUser», à la page 1090.
- «Propriété de nom BackoutRequeue», à la page 1090.
- «Propriété BackoutThreshold», à la page 1090.
- «Propriété de nom BaseQueue», à la page 1091.
- «Propriété CloseOptions», à la page 1091.
- «Propriété CompletionCode», à la page 1091.
- «Propriété ConnectionReference», à la page 1091.
- «Propriété CreationDateTime», à la page 1091.
- «Propriété CurrentDepth», à la page 1092.
- «Propriété DefaultInputOpenOption», à la page 1092.
- «Propriété DefaultPersistence», à la page 1092.
- «Propriété DefaultPriority», à la page 1092.
- «Propriété DefinitionType», à la page 1092.
- «Propriété d'événement DepthHigh», à la page 1093.
- «Propriété DepthHighLimit», à la page 1093.
- «Propriété d'événement DepthLow», à la page 1093.
- «Propriété DepthLowLimit», à la page 1093.
- «Propriété d'événement DepthMaximum», à la page 1093.
- «Propriété d'événement DepthHigh», à la page 1093.
- «Propriété DepthHighLimit», à la page 1093.
- «Propriété d'événement DepthLow», à la page 1093.
- «Propriété DepthLowLimit», à la page 1093.
- «Propriété d'événement DepthMaximum», à la page 1093.
- «Propriété Description», à la page 1093.
- «Propriété de nom DynamicQueue», à la page 1094.
- «Propriété d'annulation HardenGet», à la page 1094.
- «Propriété InhibitGet», à la page 1094.
- «Propriété InhibitPut», à la page 1094.
- «Propriété de nom InitiationQueue», à la page 1095.
- «Propriété IsOpen», à la page 1095.
- «Propriété MaximumDepth», à la page 1095.
- «Propriété MaximumMessageLength», à la page 1095.
- «Propriété de séquence MessageDelivery», à la page 1095.
- «Propriété ObjectHandle», à la page 1096.
- «Propriété OpenInputCount», à la page 1096.
- «Propriété OpenOptions», à la page 1096.
- «Propriété OpenOutputCount», à la page 1096.
- «Propriété OpenStatus», à la page 1096.
- «Propriété ProcessName», à la page 1097.
- «Propriété de nom QueueManager», à la page 1097.
- «Propriété QueueType», à la page 1097.
- «Propriété ReasonCode», à la page 1097.

- «Propriété ReasonName», à la page 1097.
- «Propriété RemoteQueueManagerName», à la page 1098.
- «Propriété de nom RemoteQueue», à la page 1098.
- «Propriété ResolvedQueueManagerName», à la page 1098.
- «Propriété de nom ResolvedQueue», à la page 1098.
- «Propriété RetentionInterval», à la page 1098.
- «Propriété de portée», à la page 1098.
- «Propriété ServiceInterval», à la page 1099.
- «Propriété d'événement ServiceInterval», à la page 1099.
- «Propriété de partageabilité», à la page 1099.
- «Propriété de nom TransmissionQueue», à la page 1099.
- «Propriété TriggerControl», à la page 1099.
- «Propriété TriggerData», à la page 1100.
- «Propriété TriggerDepth», à la page 1100.
- «Propriété de priorité TriggerMessage», à la page 1100.
- «Propriété TriggerType», à la page 1100.
- «propriété Usage», à la page 1100.

Des méthodes

- «Méthode des codes ClearError», à la page 1101
- «méthode Close», à la page 1101
- «Méthode GET», à la page 1101
- «méthode Open», à la page 1102
- «PUT (méthode)», à la page 1102

Accès aux propriétés

Si l'objet file d'attente n'est pas connecté à un gestionnaire de files d'attente, vous pouvez lire les propriétés suivantes:

- «Propriété CompletionCode», à la page 1091
- «Propriété OpenStatus», à la page 1096
- «Propriété ReasonCode», à la page 1097

et vous pouvez lire et écrire dans:

- «Propriété d'ID AlternateUser», à la page 1090
- «Propriété CloseOptions», à la page 1091
- «Propriété ConnectionReference», à la page 1091
- «propriété Name», à la page 1095
- «Propriété OpenOptions», à la page 1096

Si l'objet file d'attente est connecté à un gestionnaire de files d'attente, vous pouvez lire toutes les propriétés.

Propriétés d'attribut de file d'attente

Les propriétés non répertoriées dans la section précédente sont tous les attributs de la file d'attente WebSphere MQ sous-jacente. Ils ne sont accessibles que si l'objet est connecté à un gestionnaire de files d'attente et que l'ID utilisateur de l'utilisateur est autorisé à effectuer une interrogation ou une définition

sur cette file d'attente. Si un autre ID utilisateur est défini et que l'ID utilisateur en cours est autorisé à l'utiliser, l'autorisation est vérifiée à la place.

La propriété doit être une propriété appropriée pour le QueueTypeindiqué. Pour plus d'informations, voir Attributs des files d'attente .

Si ces conditions ne s'appliquent pas, l'accès à la propriété définit un CompletionCode de MQCC_FAILED et l'un des ReasonCodessuivants:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode est MQCC_WARNING)

Ouverture d'une file d'attente

La seule façon de créer un objet MQQueue consiste à utiliser la méthode MQQueueManager AccessQueue ou par Nouveau. Un objet MQQueue ouvert reste ouvert (OpenStatus= TRUE) jusqu'à ce qu'il soit fermé ou supprimé ou jusqu'à ce que l'objet de gestionnaire de files d'attente en cours de création soit supprimé ou que la connexion au gestionnaire de files d'attente soit perdue. La valeur de la propriété MQQueue CloseOptions contrôle le comportement de l'opération de fermeture qui a lieu lorsque l'objet MQQueue est supprimé.

La méthode MQQueueManager AccessQueue ouvre la file d'attente à l'aide du paramètre OpenOptions . La méthode MQQueue.Open ouvre la file d'attente à l'aide de la propriété OpenOptions . WebSphere MQ valide OpenOptions par rapport à l'autorisation utilisateur dans le cadre du processus d'ouverture de file d'attente.

Propriété d'ID AlternateUser

Lecture-écriture. ID utilisateur de remplacement utilisé pour valider l'accès à la file d'attente lorsqu'elle est ouverte.

Cette propriété ne peut pas être définie tant que l'objet est ouvert (c'est-à-dire lorsque IsOpen a pour valeur TRUE).

Défini dans: Classe MQQueue

Type de données: chaîne de 12 caractères

Syntaxe: Pour obtenir: *altuser $ = MQQueue*.**AlternateUserId**

A définir: *MQQueue.***AlternateUserId** = *altuser $*

Propriété de nom BackoutRequeue

Lecture seule. L'attribut MQI BackOutRequeueQName .

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *backoutrequeuename $ = MQQueue*.**BackoutRequeueNom**

Propriété BackoutThreshold

Lecture seule. Attribut BackoutThreshold MQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- Voir BackoutThreshold (MQLONG).

Syntaxe: Pour obtenir: *backoutthreshold & = MQQueue.***BackoutThreshold**

Propriété de nom BaseQueue

Lecture seule. Nom de la file d'attente dans laquelle l'alias est résolu.

Valide uniquement pour les files d'attente alias.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *baseqname $ = MQQueue***.BaseQueueNom**

Propriété CloseOptions

Lecture-Ecriture. Options utilisées pour contrôler ce qui se passe lorsque la file d'attente est fermée.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQCO_AUCUN
- MQCO_DELETE
- MQCO_DELETE_PURGE

MQCO_DELETE et MQCO_DELETE_PURGE sont valides uniquement pour les files d'attente dynamiques.

Syntaxe: Pour obtenir: *closeopt & = MQQueue***.CloseOptions**

A définir: *MQQueue***.CloseOptions** = *closeopt &*

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode & = MQQueue***.CompletionCode**

Propriété ConnectionReference

Lecture-écriture. Définit l'objet gestionnaire de files d'attente auquel appartient un objet file d'attente. La référence de connexion ne peut pas être écrite lorsqu'une file d'attente est ouverte.

Défini dans: Classe MQQueue

Type de données: MQQueueManager

Valeurs :

- Référence à un objet Gestionnaire de files d'attente WebSphere MQ actif

Syntaxe: Pour définir: *set MQQueue***.ConnectionReference** = *ConnectionReference*

Pour obtenir: *set ConnectionReference = MQQueue***.ConnectionReference**

Propriété CreationDateTime

Lecture seule. Date et heure de création de cette file d'attente.

Défini dans: Classe MQQueue

Type de données: Variante de type 7 (date/heure) ou EMPTY

Syntaxe: Pour obtenir: *datetime = MQQueue*.**CreationDateTime**

Propriété CurrentDepth

Lecture seule. Nombre de messages stockés actuellement dans la file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *currentdepth & = MQQueue*.**CurrentDepth**

Propriété DefaultInputOpenOption

Lecture seule. Contrôle la manière dont la file d'attente est ouverte si OpenOptions spécifie MQOO_INPUT_AS_Q_DEF.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

Syntaxe: Pour obtenir: *defaultinop & = MQQueue*.**DefaultInputOpenOption**

Propriété DefaultPersistence

Lecture seule. Persistance par défaut des messages dans une file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *defpersistence & = MQQueue*.**DefaultPersistence**

Propriété DefaultPriority

Lecture seule. Priorité par défaut des messages d'une file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *defpriority & = MQQueue*.**DefaultPriority**

Propriété DefinitionType

Lecture seule. Type de définition de file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQDT_PRÉDÉFINI
- MQQDT_PERMANENT_DYNAMIQUE
- MQQDT_TEMPORARY_DYNAMIQUE

Syntaxe: Pour obtenir: *deftype & = MQQueue*.**DefinitionType**

Propriété d'événement DepthHigh

Lecture seule. Attribut d'événement QDepthHighMQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *depthhighevent & = MQQueue.***DepthHighEvent**

Propriété DepthHighLimit

Lecture seule. Attribut de limite QDepthHighMQI.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *depthhighlimit & = MQQueue.***DepthHighLimite**

Propriété d'événement DepthLow

Lecture seule. Attribut d'événement QDepthLowMQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *depthlowevent & = MQQueue.***DepthLowEvent**

Propriété DepthLowLimit

Lecture seule. Attribut MQI QDepthLowLimit.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *depthlowlimit & = MQQueue.***DepthLowLimit**

Propriété d'événement DepthMaximum

Lecture seule. Attribut d'événement QDepthMaxMQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQEVR_DISABLED
- MQEVR_ENABLED

Syntaxe: Pour obtenir: *depthmaximumevent & = MQQueue.***DepthMaximumEvent**

Propriété Description

Lecture seule. Description de la file d'attente.

Défini dans: Classe MQQueue

Type de données: chaîne de 64 caractères

Syntaxe: Pour obtenir: *description $ = MQQueue*.**Description**

Propriété de nom DynamicQueue

Lecture-écriture, lecture seule lorsque la file d'attente est ouverte.

Contrôle le nom de la file d'attente dynamique utilisée lorsqu'une file d'attente modèle est ouverte. Il peut être défini avec un caractère générique par l'utilisateur en tant qu'ensemble de propriétés (uniquement lorsque la file d'attente est fermée) ou en tant que paramètre de MQQueueManager.AccessQueue().

Le nom réel de la file d'attente dynamique est trouvé en interrogeant QueueName.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Valeurs :

- Tout nom de file d'attente WebSphere MQ valide.

Syntaxe: Pour définir: *MQQueue*.**DynamicQueueNom** = *nom_file_attente_dynamique $*

Pour obtenir: *dynamicqueuename $ = MQQueue*.**DynamicQueueNom**

Propriété d'annulation HardenGet

Lecture seule. Indique s'il convient de conserver un nombre précis de réenregistrements.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQA_BACKOUT_HARDENED
- MQQA_BACKOUT_NOT HARDENED

Syntaxe: Pour obtenir: *hardengetback & = MQQueue*.**HardenGetAnnulation**

Propriété InhibitGet

Lecture-écriture. Attribut InhibitGet de l'interface MQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQA_GET_INHIBÉE
- MQQA_GET_ALLOWED

Syntaxe: Pour obtenir: *getstatus & = MQQueue*.**InhibitGet**

A définir: *MQQueue*.**InhibitGet** = *getstatus &*

Propriété InhibitPut

Lecture-écriture. Attribut InhibitPut de l'interface MQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQA_PUT_INHIBÉ

- MQQA_PUT_ALLOWED

Syntaxe: Pour obtenir: *putstatus & = MQQueue*.**InhibitPut**

A définir: *MQQueue*.**InhibitPut** = *putstatus &*

Propriété de nom InitiationQueue

Lecture seule. Nom de la file d'attente d'initialisation.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *initqname $ = MQQueue*.**InitiationQueueNom**

Propriété IsOpen

Indique si la file d'attente est ouverte.

Lecture seule.

Défini dans: Classe MQQueue

Type de données: Booléen

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *open = MQQueue*.**IsOpen**

Propriété MaximumDepth

Lecture seule. Longueur maximale de la file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *maxdepth & = MQQueue*.**MaximumDepth**

Propriété MaximumMessageLength

Lecture seule. Longueur de message maximale autorisée en octets pour cette file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *maxmlength & = MQQueue*.**MaximumMessageLongueur**

Propriété de séquence MessageDelivery

Lecture seule. Séquence de livraison des messages.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQMDS_PRIORITE
- MQMDS_FIFO

Syntaxe: Pour obtenir: *messdelseq & = MQQueue*.**MessageDeliverySéquence**

propriété Name

Lecture-écriture. Attribut de file d'attente MQI. Cette propriété ne peut pas être écrite une fois que la file d'attente MQQueue est ouverte.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *name $ = MQQueue***.name**

A définir: *MQQueue***.name** = *nom $*

Remarque : Visual Basic réserve la propriété "Name" à utiliser dans l'interface visuelle. Par conséquent, lors de l'utilisation dans Visual Basic, utilisez des minuscules, c'est-à-dire "name".

Propriété ObjectHandle

Lecture seule. Descripteur d'objet de l'objet de file d'attente WebSphere MQ .

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *hobj & = MQQueue.***ObjectHandle**

Propriété OpenInputCount

Lecture seule. Nombre d'ouvertures pour la saisie.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *openincount & = MQQueue***.OpenInputCount**

Propriété OpenOptions

Lecture-écriture. Options à utiliser pour ouvrir la file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- Voir OpenOptions (MQLONG).

Syntaxe: Pour obtenir: *openopt & = MQQueue***.OpenOptions**

A définir: *MQQueue.***OpenOptions** = *openopt &*

Propriété OpenOutputCount

Lecture seule. Nombre d'ouvertures pour la sortie.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *openoutcount & = MQQueue***.OpenOutputCount**

Propriété OpenStatus

Lecture seule. Indique si la file d'attente est ouverte ou non. La valeur initiale est TRUE après la méthode AccessQueue ou FALSE après New.

Défini dans: Classe MQQueue

Type de données: Booléen

Valeurs :

- TRUE (-1)

- FALSE (0)

Syntaxe: Pour obtenir: *status & = MQQueue*.**OpenStatus**

Propriété ProcessName

Lecture seule. Attribut ProcessName MQI.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *procname $ =MQQueue*.**ProcessName**

Propriété de nom QueueManager

Lecture-écriture. Nom du gestionnaire de files d'attente WebSphere MQ .

Défini dans: Classe MQQueue

Type de données : chaîne

Syntaxe: To get: *QueueManagerName$ = MQQueue*.**QueueManagerNom**

A définir: *MQQueue*.**QueueManagerName** = *QueueManagerName$*

Propriété QueueType

Lecture seule. Attribut QType MQI.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQT_ALIAS (alias MQ)
- MQQT_LOCAL
- MQQT_MODELE
- MQQT_REMOTE

Syntaxe: Pour obtenir: *queuetype & = MQQueue*.**QueueType**

Propriété ReasonCode

Lecture seule. Renvoie le code raison défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasoncode & = MQQueue*.**ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Défini dans: Classe MQQueue

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQQueue*.**ReasonName**

Propriété RemoteQueueManagerName

Lecture seule. Nom du gestionnaire de files d'attente éloignées. Valide uniquement pour les files d'attente éloignées.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *remqmanname $ = MQQueue*.**RemoteQueueManagerName**

Propriété de nom RemoteQueue

Lecture seule. Nom de la file d'attente tel qu'il est connu sur le gestionnaire de files d'attente éloignées. Valide uniquement pour les files d'attente éloignées.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *remqname $ = MQQueue*.**RemoteQueueNom**

Propriété ResolvedQueueManagerName

Lecture seule. Nom du gestionnaire de files d'attente de destination finale connu du gestionnaire de files d'attente local.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *resqmanname $ = MQQueue*.**ResolvedQueueManagerName**

Propriété de nom ResolvedQueue

Lecture seule. Nom de la file d'attente de destination finale connue du gestionnaire de files d'attente local.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *resqname $ = MQQueue*.**ResolvedQueueNom**

Propriété RetentionInterval

Lecture seule. Durée pendant laquelle la file d'attente doit être conservée.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *retinterval & = MQQueue*.**RetentionInterval**

Propriété de portée

Lecture seule. Contrôle si une entrée de cette file d'attente existe également dans un répertoire de cellule.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQSCO_Q_DIR
- MQSCO_CELL

Syntaxe: Pour obtenir: *scope & = MQQueue*.**Scope**

Propriété ServiceInterval

Lecture seule. Attribut QServiceInterval de MQI.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *serviceinterval & = MQQueue.***ServiceInterval**

Propriété d'événement ServiceInterval

Lecture seule. Attribut d'événement MQI QServiceInterval.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQSIE_HIGH
- MQQSIE_OK
- MQQSIE_NONE

Syntaxe: Pour obtenir: *serviceintervalevent & = MQQueue.***ServiceIntervalEvent**

Propriété de partageabilité

Lecture seule. Partage de file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

Syntaxe: Pour obtenir: *shareability & = MQQueue*.**Shareability**

Propriété de nom TransmissionQueue

Lecture seule. Nom de la file d'attente de transmission. Valide uniquement pour les files d'attente éloignées.

Défini dans: Classe MQQueue

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *transqname $ = MQQueue*.**TransmissionQueueNom**

Propriété TriggerControl

Lecture-écriture. Contrôle du déclencheur.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQTC_OFF
- MQTC_ON

Syntaxe: Pour obtenir: *trigcontrol & = MQQueue*.**TriggerControl**

A définir: *MQQueue*.**TriggerControl** = *trigcontrol &*

Propriété TriggerData

Lecture-écriture. Données de déclenchement.

Défini dans: Classe MQQueue

Type de données: chaîne de 64 caractères

Syntaxe: Pour obtenir: *trigdata $ = MQQueue*.**TriggerData**

A définir: *MQQueue*.**TriggerData** = *trigdata $*

Propriété TriggerDepth

Lecture-écriture. Nombre de messages devant figurer dans la file d'attente avant qu'un message de déclenchement ne soit écrit.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *trigdepth & = MQQueue*.**TriggerDepth**

A définir: *MQQueue*.**TriggerDepth** = *trigdepth &*

Propriété de priorité TriggerMessage

Lecture-écriture. Priorité de message de seuil pour les déclencheurs.

Défini dans: Classe MQQueue

Type de données: Long

Syntaxe: Pour obtenir: *trigmesspriority & = MQQueue*.**TriggerMessagePriority**

A définir: *MQQueue*.**TriggerMessagePriority** = *trigmesspriority &*

Propriété TriggerType

Lecture-écriture. Type de déclencheur.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQTT_AUCUN
- MQTT_PREMIER
- MQTT_EVERY
- MQTT_DEPTH

Syntaxe: Pour obtenir: *trigtype & = MQQueue*.**TriggerType**

A définir: *MQQueue*.**TriggerType** = *Type de déclencheur &*

propriété Usage

Lecture seule. Indique à quoi sert la file d'attente.

Défini dans: Classe MQQueue

Type de données: Long

Valeurs :

- MQUS_NORMAL
- TRANSMIS_MQUS_TRANSMISSION

Syntaxe: Pour obtenir: *usage & = MQQueue*.**Utilisation**

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQQueue et la classe MQSession.

Défini dans: Classe MQQueue

Syntaxe: Call *MQQueue*.**ClearErrorCodes ()**

méthode Close

Ferme une file d'attente à l'aide des valeurs en cours de CloseOptions.

Défini dans: Classe MQQueue

Syntaxe: Call *MQQueue*.**Fermer ()**

Méthode GET

Extrait un message de la file d'attente.

Cette méthode utilise un objet MQMessage comme paramètre, en utilisant certaines des zones du MQMD de l'objet comme paramètres d'entrée. En particulier, les zones MessageId et CorrelId sont utilisées, il est donc important de s'assurer que ces zones sont définies selon les besoins. Pour plus d'informations sur ces zones, voir MsgId (MQBYTE24) et CorrelId (MQBYTE24) .

Si la méthode échoue, l'objet MQMessage reste inchangé. Si elle aboutit, les parties MQMD et Données de message de l'objet MQMessage sont remplacées par les parties MQMD et Données de message du message entrant. Les propriétés de contrôle MQMessage sont définies comme suit:

- **MessageLength** est défini sur la longueur du message WebSphere MQ
- **DataLength** est défini sur la longueur du message WebSphere MQ
- **DataOffset** est défini sur zéro

Défini dans:
classe MQQueue

Syntaxe: Call *MQQueue*.**Obtenir**(*Message, GetMsgOptions, GetMsgLongueur*)

Paramètres
Message :

Objet MQMessage représentant le message à extraire.

Options de GetMsg:

Objet facultatif MQGetMessageOptions permettant de contrôler l'opération d'obtention. Si ce paramètre n'est pas spécifié, les options MQGetMessagepar défaut sont utilisées.

GetMsgLongueur:

Valeur de longueur de 2 ou 4 octets facultative pour contrôler la longueur maximale du message WebSphere MQ extrait de la file d'attente.

Si l'option MQGMO_ACCEPT_TRUNCATED_MSG est spécifiée, l'opération GET aboutit avec le code achèvement MQCC_WARNING et le code anomalie MQRC_TRUNCATED_MSG_ACCEPTED si la taille du message dépasse la longueur spécifiée.

MessageData contient les premiers GetMsgoctets de données de longueur.

Si MQGMO_ACCEPT_TRUNCATED_MSG ***n'est pas*** spécifié et que la taille du message dépasse la longueur spécifiée, le code achèvement de MQCC_FAILED et le code anomalie MQRC_TRUNCATED_MESSAGE_FAILED sont renvoyés.

Si le contenu de la mémoire tampon de messages n'est pas défini, la longueur totale du message est définie sur la longueur totale du message qui aurait été extrait.

Si le paramètre de longueur de message n'est pas spécifié, la longueur de la mémoire tampon de messages est automatiquement ajustée à la taille du message entrant.

méthode Open

Ouvre une file d'attente à l'aide des valeurs en cours de:

1. QueueName
2. QueueManagerName
3. AlternateUserid
4. DynamicQueueNom

Défini dans:
classe MQQueue

Syntaxe: Call *MQQueue*.**Ouvrir ()**

PUT (méthode)

Place un message dans la file d'attente.

Cette méthode utilise un objet MQMessage comme paramètre. Les propriétés du descripteur de message (MQMD) de cet objet peuvent être modifiées à la suite de cette méthode. Les valeurs qu'ils ont immédiatement après l'exécution de cette méthode sont celles qui ont été placées dans WebSphere MQ.

Les modifications apportées à l'objet MQMessage une fois l'insertion terminée n'affectent pas le message réel dans la file d'attente WebSphere MQ .

Défini dans:
classe MQQueue

Syntaxe: Call *MQQueue*.**Insertion**(*Message, PutMsgOptions)*

Paramètres
Message

Objet MQMessage représentant le message à insérer.

Options de PutMsg

Objet d'options MQPutMessagecontenant des options permettant de contrôler l'opération d'insertion. S'ils ne sont pas spécifiés, les options PutMessagepar défaut sont utilisées.

Classe MQMessage

Cette classe représente un message WebSphere MQ . Il inclut des propriétés permettant d'encapsuler le descripteur de message WebSphere MQ (MQMD) et fournit une mémoire tampon pour stocker les données de message définies par l'application.

La classe inclut des méthodes d'écriture pour copier des données d'une application ActiveX vers un objet MQMessage. De même, la classe inclut des méthodes Read pour copier des données d'un objet MQMessage vers une application ActiveX . La classe gère automatiquement l'allocation et la libération de la mémoire pour la mémoire tampon. L'application n'a pas besoin de déclarer la taille de la mémoire tampon lors de la création d'un objet MQMessage car la mémoire tampon augmente pour contenir les données qui lui sont écrites.

Vous ne pouvez pas placer un message dans une file d'attente WebSphere MQ si la taille de la mémoire tampon dépasse la propriété MaximumMessageLongueur de cette file d'attente.

Une fois construit, un objet MQMessage peut être inséré dans une file d'attente WebSphere MQ à l'aide de la méthode MQQueue.Put . Cette méthode prend une copie des portions de données MQMD et de message de l'objet et place cette copie dans la file d'attente. L'application peut donc modifier ou supprimer un objet MQMessage après l'insertion, sans affecter le message dans la file d'attente

WebSphere MQ . Le gestionnaire de files d'attente peut ajuster certaines zones du MQMD lorsqu'il copie le message dans la file d'attente WebSphere MQ .

Un message entrant peut être lu dans un objet MQMessage à l'aide de la méthode MQQueue.Get . Cette opération remplace les données de MQMD ou de message qui peuvent déjà se trouver dans l'objet MQMessage par des valeurs provenant du message entrant. Il ajuste la taille de la mémoire tampon de données de l'objet MQMessage pour qu'elle corresponde à la taille des données de message entrant.

Confinement

Les messages sont contenus dans la classe MQSession.

Création

Nouveau crée un objet MQMessage. Ses propriétés de descripteur de message sont initialement définies sur les valeurs par défaut et sa mémoire tampon de données de message est vide.

Syntaxe

```
Dim msg As New MQMessage or Set msg = New MQMessage
```

Propriétés

Les propriétés de contrôle sont les suivantes:

- «Propriété CompletionCode», à la page 1105
- «Propriété DataLength», à la page 1105
- «Propriété DataOffset», à la page 1106
- «Propriété MessageLength», à la page 1106
- «Propriété ReasonCode», à la page 1106
- «Propriété ReasonName», à la page 1107

Les propriétés du descripteur de message sont les suivantes:

- «Propriété AccountingToken», à la page 1107
- «Propriété hexadécimale AccountingToken», à la page 1107
- «Propriété de données ApplicationId», à la page 1107
- «Propriété de données ApplicationOrigin», à la page 1108
- «Propriété BackoutCount», à la page 1108
- «Propriété CharacterSet», à la page 1108
- «Propriété CorrelationId», à la page 1108
- «Propriété hexadécimale CorrelationId», à la page 1109
- «Propriété de codage», à la page 1109
- «Propriété d'expiration», à la page 1110
- «Propriété de commentaire en retour», à la page 1110
- «Format, propriété», à la page 1110
- «Propriété GroupId», à la page 1110
- «Propriété hexadécimale GroupId», à la page 1111
- «Propriété MessageData», à la page 1111
- «Propriété MessageFlags», à la page 1111
- «Propriété MessageId», à la page 1112
- «Propriété hexadécimale MessageId», à la page 1112

- «Propriété de numéro MessageSequence», à la page 1112
- «Propriété MessageType», à la page 1112
- «Propriété de décalage», à la page 1113
- «Propriété OriginalLength», à la page 1113
- «Propriété de persistance», à la page 1113
- «Priority, propriété», à la page 1113
- «Propriété de nom PutApplication», à la page 1114
- «Propriété de type PutApplication», à la page 1114
- «Propriété PutDateTime», à la page 1114
- «ReplyToQueueManagerPropriété Name», à la page 1114
- «ReplyToPropriétéQueueName», à la page 1114
- «Propriétés du rapport», à la page 1115
- «Propriété TotalMessageLength», à la page 1115
- «Propriété UserID», à la page 1115

Des méthodes

- «Méthode des codes ClearError», à la page 1115
- «Méthode ClearMessage», à la page 1115
- «Méthode de lecture», à la page 1116
- «Méthode ReadBoolean», à la page 1116
- «Méthode ReadByte», à la page 1116
- «Méthode ReadDecimal2», à la page 1116
- «Méthode ReadDecimal4», à la page 1116
- «Méthode ReadDouble», à la page 1116
- «Méthode ReadDouble4», à la page 1117
- «Méthode ReadFloat», à la page 1117
- «Méthode ReadInt2», à la page 1117
- «Méthode ReadInt4», à la page 1117
- «Méthode ReadLong», à la page 1117
- «Méthode ReadNullTerminatedString», à la page 1118
- «Méthode ReadShort», à la page 1118
- «Méthode ReadString», à la page 1118
- «Méthode ReadUInt2», à la page 1119
- «Méthode d'octet ReadUnsigned», à la page 1119
- «Méthode ReadUTF», à la page 1119
- «Méthode ResizeBuffer», à la page 1119
- «Méthode d'écriture», à la page 1120
- «Méthode WriteBoolean», à la page 1120
- «Méthode WriteByte», à la page 1120
- «Méthode WriteDecimal2», à la page 1120
- «Méthode WriteDecimal4», à la page 1121
- «Méthode WriteDouble», à la page 1121
- «Méthode WriteDouble4», à la page 1121

- «Méthode WriteFloat», à la page 1121
- «Méthode WriteInt2», à la page 1122
- «Méthode WriteInt4», à la page 1122
- «Méthode WriteLong», à la page 1122
- «Méthode WriteNullTerminatedString», à la page 1122
- «Méthode WriteShort», à la page 1122
- «Méthode WriteString», à la page 1123
- «Méthode WriteUInt2», à la page 1123
- «Méthode d'octet WriteUnsigned», à la page 1123
- «Méthode WriteUTF», à la page 1124

Accès aux propriétés

Toutes les propriétés peuvent être lues à tout moment.

Les propriétés de contrôle sont en lecture seule, à l'exception de DataOffset qui est en lecture-écriture. Les propriétés du descripteur de message sont toutes en lecture-écriture, à l'exception de BackoutCount et de TotalMessageLength, qui sont toutes deux en lecture seule.

Notez toutefois que certaines propriétés MQMD peuvent être modifiées par le gestionnaire de files d'attente lorsque le message est placé dans une file d'attente WebSphere MQ . Voir les zones dans MQMD pour plus de détails sur la façon dont elles peuvent être modifiées.

Conversion de données

Vous pouvez transmettre des données binaires à un message WebSphere MQ en définissant la propriété CharacterSet sur l'identificateur de jeu de caractères codés du gestionnaire de files d'attente (MQCCSI_Q_MGR) et en lui transmettant une chaîne. Vous pouvez utiliser la fonction chr $ pour définir des données non-caractère dans la chaîne.

Les méthodes de lecture et d'écriture effectuent la conversion des données. Ils sont convertis entre les formats internes ActiveX et les formats de message WebSphere MQ tels que définis par les propriétés Encoding et CharacterSet à partir du descripteur de message. Lors de l'écriture d'un message, définissez des valeurs dans Encoding et CharacterSet qui correspondent aux caractéristiques du destinataire du message avant d'émettre une méthode Write. Lors de la lecture d'un message, cette étape n'est normalement pas requise car ces valeurs ont été définies à partir de celles du MQMD entrant.

Il s'agit d'une étape de conversion de données supplémentaire qui se produit après toute conversion effectuée par la méthode MQQueue.Get .

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement WebSphere MQ défini par la méthode ou la propriété la plus récente émise pour cet objet.

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode & = MQMessage*.**CompletionCode**

Propriété DataLength

Lecture seule. Cette propriété renvoie la valeur:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Il peut être utilisé avant une méthode Read, pour vérifier que le nombre de caractères attendu est bien présent dans la mémoire tampon.

La valeur initiale est zéro.

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *bytesleft & = MQMessage*.**DataLength**

Propriété DataOffset

Lecture-écriture. Position actuelle dans la partie Données de message de l'objet de message.

La valeur est exprimée sous la forme d'un décalage d'octet à partir du début de la mémoire tampon de données de message ; le premier caractère de la mémoire tampon correspond à une valeur DataOffset de zéro.

Une méthode de lecture ou d'écriture commence son opération au niveau du caractère référencé par DataOffset. Ces méthodes traitent les données dans la mémoire tampon séquentiellement à partir de cette position et mettent à jour DataOffset pour qu'il pointe vers l'octet (le cas échéant) qui suit immédiatement le dernier octet traité.

DataOffset ne peut prendre que des valeurs comprises entre zéro et MessageLength inclus. Lorsque DataOffset = MessageLength , il pointe vers la fin, c'est-à-dire le premier caractère non valide de la mémoire tampon. Les méthodes d'écriture sont autorisées dans cette situation: elles étendent les données dans la mémoire tampon et augmentent MessageLength du nombre d'octets ajoutés. La lecture au-delà de la fin de la mémoire tampon n'est pas valide.

La valeur initiale est zéro.

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *currpos & = MQMessage*.**DataOffset**

A définir: *MQMessage*.**DataOffset** = *currpos &*

Propriété MessageLength

Lecture seule. Renvoie la longueur totale de la partie Données de message de l'objet de message en caractères, quelle que soit la valeur de DataOffset.

La valeur initiale est zéro. Il est défini sur la longueur du message entrant après un appel de méthode Get qui a fait référence à cet objet de message. Elle est incrémentée si l'application utilise une méthode d'écriture pour ajouter des données à l'objet. Il n'est pas affecté par les méthodes de lecture.

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: To get: *msglength & = MQMessage*.**MessageLength**

Propriété ReasonCode

Lecture seule. Renvoie le code anomalie défini par la méthode ou l'accès à la propriété la plus récente émis pour cet objet.

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: To get: *reasoncode & = MQMessage*.**ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE". **Défini dans:** Classe MQMessage

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: To get: *reasonname $ = MQMessage*.**ReasonName**

Propriété AccountingToken

Lecture-écriture. Le MQMD AccountingToken -partie du contexte d'identité du message.

Sa valeur initiale est toutes des valeurs nulles.

Défini dans: Classe MQMessage

Type de données: Chaîne de 32 caractères

Syntaxe: Pour obtenir: *actoken $ = MQMessage*.**AccountingToken**

A définir: *MQMessage.***AccountingToken** = *actoken $*

Pour plus d'informations sur le moment où vous devez utiliser AccountingTokenHex à la place de la propriété AccountingToken , voir «Propriétés du descripteur de message», à la page 1066 .

Propriété hexadécimale AccountingToken

Lecture-écriture. Le MQMD AccountingToken -partie du contexte d'identité du message.

Tous les deux caractères représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représentent le caractère unique "A", la paire de caractères "6" et "2" représentent le caractère unique "B", etc.

Vous devez indiquer 64 caractères hexadécimaux valides.

Sa valeur initiale est "0 ... 0"

Défini dans: Classe MQMessage

Type de données: Chaîne de 64 caractères hexadécimaux représentant 32 caractères ASCII

Syntaxe: Pour obtenir: *actokenh $ = MQMessage*.**AccountingTokenHex**

A définir: *MQMessage.***AccountingTokenHex** = *actokenh $*

Pour plus d'informations sur le moment où vous devez utiliser AccountingTokenHex à la place de la propriété AccountingToken , voir «Propriétés du descripteur de message», à la page 1066 .

Propriété de données ApplicationId

Lecture-écriture. Données MQMD ApplIdentity-partie du contexte d'identité du message.

Sa valeur initiale est tous des blancs.

Défini dans: Classe MQMessage

Type de données: Chaîne de 32 caractères

Syntaxe: Pour obtenir: *applid $ = MQMessage*.**ApplicationIdData**

A définir: *MQMessage*.**ApplicationIdData** = *ID_application $*

Propriété de données ApplicationOrigin

Lecture-écriture. Données MQMD ApplOrigin-partie du contexte d'origine du message.

Sa valeur initiale est tous des blancs.

Défini dans: Classe MQMessage

Type de données: chaîne de 4 caractères

Syntaxe: Pour obtenir: *applor $ = MQMessage*.**ApplicationOriginData**

A définir: *MQMessage*.**ApplicationOriginData** = *applor $*

Propriété BackoutCount

Lecture seule. BackoutCountMQMD.

Sa valeur initiale est 0

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *backoutct & = MQMessage*.**BackoutCount**

Propriété CharacterSet

Lecture-écriture. Le MQMD CodedCharSetId.

Sa valeur initiale est la valeur spéciale **MQCCSI_Q_MGR** .

Si CharacterSet est défini sur **MQCCSI_Q_MGR**, la page de codes de l'environnement local en cours est utilisée pour la conversion de caractères dans la méthode WriteString . Pour les applications serveur, la page de codes utilisée est celle du gestionnaire de files d'attente. Pour les applications client, il s'agit de la page de codes de l'environnement local en cours par défaut.

Exemple :

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

où'n'est supérieur ou égal à zéro et inférieur ou égal à 255, entraîne l'écriture d'un octet unique de valeur de'n'dans le tampon.

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *:30ccid& = MQMessage* **.CharacterSet**

A définir: *MQMessage*.**CharacterSet**= *ccid &*

Exemple

Si vous voulez que la chaîne soit écrite dans la page de codes 437, émettez:

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Définissez la valeur souhaitée dans CharacterSet avant d'émettre des appels WriteString .

Propriété CorrelationId

Lecture-écriture. CorrelationId à inclure dans le MQMD d'un message lorsqu'il est inséré dans une file d'attente. Indique également l'ID à utiliser lors de l'obtention d'un message à partir d'une file d'attente.

Sa valeur initiale est null.

Défini dans: Classe MQMessage

Type de données: chaîne de 24 caractères

Syntaxe: Pour obtenir: *correlid $ = MQMessage*.**CorrelationId** Pour définir: *MQMessage*.**CorrelationId** = *correlid $*

Voir «Propriétés du descripteur de message», à la page 1066 pour plus d'informations sur le moment où vous devez utiliser CorrelationIdHex à la place de la propriété CorrelationId .

Propriété hexadécimale CorrelationId

Lecture-écriture. CorrelationId à inclure dans le MQMD d'un message lorsqu'il est inséré dans une file d'attente. Indique également l'ID de corrélation (CorrelationId) à utiliser lors de l'obtention d'un message à partir d'une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représentent le caractère unique "A", la paire de caractères "6" et "2" représentent le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 ... 0".

Défini dans: Classe MQMessage

Type de données: Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII

Syntaxe: Pour obtenir: *correlidh $ = MQMessage*.**CorrelationIdHex**

A définir: *MQMessage*.**CorrelationIdHex** = *correlidh $*

Voir «Propriétés du descripteur de message», à la page 1066 pour savoir quand vous devez utiliser CorrelationIdHex à la place de la propriété CorrelationId .

Propriété de codage

Lecture-écriture. Zone MQMD qui identifie la représentation utilisée pour les valeurs numériques dans les données de message d'application.

Sa valeur initiale est la valeur spéciale MQENC_NATIVE, qui varie en fonction de la plateforme.

Cette propriété est utilisée par les méthodes suivantes:

- Méthode ReadDecimal2
- Méthode ReadDecimal4
- Méthode ReadDouble
- Méthode ReadDouble4
- Méthode ReadFloat
- Méthode ReadInt2
- Méthode ReadInt4
- Méthode ReadLong
- Méthode ReadShort
- Méthode ReadUInt2
- Méthode WriteDecimal2
- Méthode WriteDecimal4
- Méthode WriteDouble
- Méthode WriteDouble4
- Méthode WriteFloat
- Méthode WriteInt2

- Méthode WriteInt4
- Méthode WriteLong
- Méthode WriteShort
- Méthode WriteUInt2

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *encoding & = MQMessage*.**Codage** A définir: *MQMessage*.**Codage** = *codage &*

Si vous vous préparez à écrire des données dans la mémoire tampon de messages, vous devez définir cette zone pour qu'elle corresponde aux caractéristiques de la plateforme du gestionnaire de files d'attente de réception si le gestionnaire de files d'attente de réception est incapable d'effectuer sa propre conversion de données.

Propriété d'expiration

Lecture-écriture. Zone d'heure d'expiration MQMD, attendue en dixièmes de seconde.

Sa valeur initiale est la valeur spéciale MQEI_UNLIMITED

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *expiration & = MQMessage*.**Expiration**

A définir: *MQMessage*.**Expiration** = *expiration &*

Propriété de commentaire en retour

Lecture-écriture. Zone de retour d'informations MQMD.

Sa valeur initiale est la valeur spéciale MQFB_NONE.

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- Voir Commentaires en retour.

Syntaxe: Pour obtenir: *feedback & = MQMessage*.**Commentaires en retour**

A définir: *MQMessage*.**Feedback** = *feedback &*

Format, propriété

Lecture-écriture. Zone de format MQMD. Indique le nom d'un format intégré ou défini par l'utilisateur qui décrit la nature des données de message.

Sa valeur initiale est la valeur spéciale MQFMT_NONE.

Défini dans: Classe MQMessage

Type de données: chaîne de 8 caractères

Syntaxe: Pour obtenir: *format $ = MQMessage*.**Format**

A définir: *MQMessage*.**Format** = *format $*

Propriété GroupId

Lecture-écriture. GroupId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Indique également l'ID à utiliser lors de l'obtention d'un message à partir d'une file d'attente. Sa valeur initiale est toutes des valeurs nulles.

Défini dans:
Classe MQMessage

Type de données :
Chaîne de 24 caractères

Syntaxe: Pour obtenir: *groupid $ = MQMessage*.**GroupId**

A définir: *MQMessage*.**GroupId** = *groupid $*

Pour plus d'informations sur le moment où vous devez utiliser GroupIdHex à la place de la propriété GroupId , voir «Propriétés du descripteur de message», à la page 1066 .

Propriété hexadécimale GroupId

Lecture-écriture. GroupId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Indique également l'ID à utiliser lors de l'obtention d'un message à partir d'une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représente le caractère unique "A", la paire de caractères "6" et "2" représente le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 ... 0".

Défini dans:
Classe MQMessage

Type de données :
Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII.

Syntaxe: Pour obtenir: *groupidh $ = MQMessage*.**GroupIdHex**

A définir: *MQMessage*.**GroupIdHex** = *groupidh $*

Pour plus d'informations sur le moment où vous devez utiliser GroupIdHex à la place de la propriété GroupId , voir «Propriétés du descripteur de message», à la page 1066 .

Propriété MessageData

Lecture-écriture. Extrait ou définit l'intégralité du contenu d'un message sous forme de chaîne de caractères.

Défini dans: Classe MQMessage

Type de données: Variante

Remarque : Le type de données utilisé par cette propriété est Variant, mais MQAX s'attend à ce qu'il s'agit d'une variante de type String. Si vous transmettez une variante autre que ce type, l'erreur MQRC_OBJECT_TYPE_ERROR sera renvoyée.

Syntaxe: To get: *String$* = *MQMessage*.**MessageData**

A définir: *MQMessage*.**MessageData** = *String$*

Propriété MessageFlags

Lecture-Ecriture. Indicateurs de message spécifiant les informations de contrôle de segmentation. La valeur initiale est 0.

Défini dans:
Classe MQMessage

Type de données :
Long

Valeurs :
Voir MsgFlags (MQLONG).

Syntaxe: Pour obtenir: *messageflags & = MQMessage*.**MessageFlags**

A définir: *MQMessage*.**MessageFlags** = *messageflags &*

Propriété MessageId

Lecture-écriture. MessageId à inclure dans le MQMD d'un message lorsqu'il est inséré dans une file d'attente. Indique également l'ID à utiliser lors de l'obtention d'un message à partir d'une file d'attente.

Sa valeur initiale est toutes des valeurs nulles.

Défini dans: Classe MQMessage

Type de données: chaîne de 24 caractères

Syntaxe: Pour obtenir: *messageid $ = MQMessage***.MessageId**

A définir: *MQMessage***.MessageId** = *messageid $*

Voir «Propriétés du descripteur de message», à la page 1066 pour plus d'informations sur le moment où vous devez utiliser MessageIdHex à la place de la propriété MessageId .

Propriété hexadécimale MessageId

Lecture-écriture. MessageId à inclure dans le MQMD d'un message lorsqu'il est inséré dans une file d'attente. Egalement l'ID MessageId à utiliser lors de l'obtention d'un message à partir d'une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représentent le caractère unique "A", la paire de caractères "6" et "2" représentent le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 ... 0".

Défini dans: Classe MQMessage

Type de données: Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII

Syntaxe: Pour obtenir: *messageidh $ = MQMessage***.MessageIdHex**

A définir: *MQMessage***.MessageIdHex** = *messageidh $*

Voir «Propriétés du descripteur de message», à la page 1066 pour plus d'informations sur le moment où vous devez utiliser MessageIdHex à la place de la propriété MessageId .

Propriété de numéro MessageSequence

Lecture-Ecriture. Informations de séquence identifiant un message au sein d'un groupe. La valeur initiale est 1.

Défini dans:
Classe MQMessage

Type de données :
Long

Valeurs :
Voir MsgSeqNumber (MQLONG).

Syntaxe: Pour obtenir: *sequencenumber & = MQMessage*.**SequenceNumber**

A définir: *MQMessage*.**SequenceNumber** = *sequencenumber &*

Propriété MessageType

Lecture-écriture. Zone MQMD MsgType .

Sa valeur initiale est MQMT_DATAGRAM.

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- Voir MsgType (MQLONG).

Syntaxe: Pour obtenir: *msgtype & = MQMessage*.**MessageType**

A définir: *MQMessage*.**MessageType** = *msgtype &*

Propriété de décalage

Lecture-Ecriture. Décalage dans un message segmenté. La valeur initiale est 0.

Défini dans:
Classe MQMessage

Type de données :
Long

Valeurs :
Voir Offset (MQLONG).

Syntaxe: Pour obtenir: *offset & = MQMessage*.**Décalage**

A définir: *MQMessage*.**Décalage** = *décalage &*

Propriété OriginalLength

Lecture-Ecriture. Longueur d'origine d'un message segmenté. La valeur initiale est MQOL_UNDEFINED.

Défini dans:
Classe MQMessage

Type de données :
Long

Valeurs :
Voir OriginalLength (MQLONG).

Syntaxe: Pour obtenir: *originallength & = MQMessage*.**OriginalLength**

A définir: *MQMessage*.**OriginalLength** = *originallength &*

Propriété de persistance

Lecture-écriture. Paramètre de persistance du message.

Sa valeur initiale est MQPER_PERSISTENCE_AS_Q_DEF.

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *persistent & = MQMessage*.**Persistance**

A définir: *MQMessage*.**Persistence** = *persistance &*

Priority, propriété

Lecture-écriture. Priorité du message.

Sa valeur initiale est la valeur spéciale MQPRI_PRIORITY_AS_Q_DEF

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *priority & = MQMessage*.**Priorité**

A définir: *MQMessage*.**Priority** = *priorité &*

Propriété de nom PutApplication

Lecture-écriture. Le nom MQMD PutAppl-partie du contexte d'origine du message.

Sa valeur initiale est tous des blancs.

Défini dans: Classe MQMessage

Type de données: chaîne de 28 caractères

Syntaxe: Pour obtenir: *putapplnm $ = MQMessage*.**PutApplicationNom**

A définir: *MQMessage*.**PutApplicationName** = *putapplnm $*

Propriété de type PutApplication

Lecture-écriture. Type MQMD PutAppl-partie du contexte d'origine du message.

Sa valeur initiale est MQAT_NO_CONTEXT

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- Voir PutApplType (MQLONG).

Syntaxe: Pour obtenir: *putappltp & = MQMessage*.**PutApplicationType**

A définir: *MQMessage*.**PutApplicationType** = *putappltp &*

Propriété PutDateTime

Lecture/Ecriture. Cette propriété combine les zones MQMD PutDate et PutTime . Il s'agit des parties du contexte d'origine du message qui indiquent quand le message a été inséré.

L'extension ActiveX convertit entre le format de date / heure ActiveX et les formats de date et d'heure utilisés dans un MQMD WebSphere MQ . Si un message est reçu avec une valeur PutDate ou PutTimenon valide, la propriété d'heure PutDateaprès la méthode get est définie sur EMPTY.

Sa valeur initiale est EMPTY.

Défini dans: Classe MQMessage

Type de données: Variante de type 7 (date/heure) ou EMPTY.

Syntaxe: To get: *datetime = MQMessage*.**PutDateTime**

A définir: *MQMessage*.**PutDateTime** = *datetime*

ReplyToQueueManagerPropriété Name

Lecture-écriture. Zone MQMD ReplyTodu gestionnaire de files d'attente.

Sa valeur initiale n'est que des blancs

Défini dans: Classe MQMessage

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *replytoqmgr $ = MQMessage*.**ReplyToQueueManagerNom**

A définir: *MQMessage*.**ReplyToQueueManagerNom** = *replytoqmgr $*

ReplyToPropriétéQueueName

Lecture-écriture. Zone de file d'attente ReplyTode MQMD.

Sa valeur initiale n'est que des blancs

Défini dans: Classe MQMessage

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *replytoq $ = MQMessage*.**ReplyToQueueName**

A définir: *MQMessage*.**ReplyToQueueName** = *replytoq $*

Propriétés du rapport

Lecture-écriture. Options de rapport du message.

Sa valeur initiale est MQRO_NONE.

Défini dans: Classe MQMessage

Type de données: Long

Valeurs :

- Voir Rapport.

Syntaxe: Pour obtenir: *report & = MQMessage*.**Rapport**

A définir: *MQMessage*.**Report** = *rapport &*

Propriété TotalMessageLength

Lecture seule. Extrait la longueur du dernier message reçu par MQGET. Si le message n'a pas été tronqué, cette valeur est égale à la valeur de la propriété MessageLength .

Défini dans: Classe MQMessage

Type de données: Long

Syntaxe: Pour obtenir: *totalmessagelength & = MQMessage*.**TotalMessageLongueur**

Propriété UserID

Lecture-écriture. UserIdentifier MQMD-partie du contexte d'identité du message.

Sa valeur initiale est tous des blancs.

Défini dans: Classe MQMessage

Type de données: chaîne de 12 caractères

Syntaxe: Pour obtenir: *userid $ = MQMessage*.**UserId**

A définir: *MQMessage*.**UserId** = *userid $*

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQMessage et la classe MQSession.

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**ClearErrorCodes ()**

Méthode ClearMessage

Cette méthode efface la partie de mémoire tampon de données de l'objet MQMessage. Toute donnée de message dans la mémoire tampon de données est perdue, car MessageLength, DataLengthet DataOffset sont tous définis sur zéro.

La partie Descripteur de message (MQMD) n'est pas affectée ; une application peut avoir besoin de modifier certaines zones MQMD avant de réutiliser l'objet MQMessage. Pour redéfinir les zones MQMD, utilisez Nouveau pour remplacer l'objet par une nouvelle instance.

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**ClearMessage()**

Méthode de lecture

Lit une séquence d'octets de la mémoire tampon de message dans un tableau d'octets. DataOffset est incrémenté et la longueur des données décrémentée du nombre d'octets lus.

Défini dans:
Classe MQMessage

Syntaxe: Data = MQMessage.**Read**(len &)

Paramètres :
len &: Long. Longueur des données en octets à lire.

Méthode ReadBoolean

Lit une valeur booléenne à 1 octet à partir de la position en cours dans la mémoire tampon du message et renvoie une valeur booléenne à 2 octets TRUE (-1) /FALSE (0). DataOffset est incrémenté de un et la longueur des données est décrémentée de un.

Défini dans:
Classe MQMessage

Syntaxe: *value = MQMessage*.**ReadBoolean**

Méthode ReadByte

Cette méthode lit 1 octet à partir de la mémoire tampon des données de message, en commençant par le caractère désigné par DataOffset et le renvoie sous la forme d'un entier (2 octets signés) compris entre -128 et 127.

La méthode échoue si MQMessage.DataLength est inférieur à 1 lorsqu'elle est émise.

DataOffset est incrémenté de 1 et DataLength est décrémenté de 1 si la méthode aboutit.

L'octet de données de message est supposé être un entier binaire signé.

Défini dans: Classe MQMessage

Syntaxe: *integerv% = MQMessage*.**ReadByte**

Méthode ReadDecimal2

Lit un nombre décimal condensé de 2 octets et le renvoie sous la forme d'un entier signé de 2 octets. DataOffset est incrémenté de deux et la longueur des données est décrémentée de deux.

Défini dans:
Classe MQMessage

Syntaxe: *value% = MQMessage*.**ReadDecimal2**

Méthode ReadDecimal4

Lit un nombre décimal condensé de 4 octets et le renvoie sous la forme d'un entier signé de 4 octets. DataOffset est incrémenté de quatre et la longueur des données est décrémentée de quatre.

Défini dans:
Classe MQMessage

Syntaxe: Call *value & = MQMessage*.**ReadDecimal4**

Méthode ReadDouble

Cette méthode lit 8 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset et en renvoyant une valeur en virgule flottante double (8 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 8 lorsqu'elle est émise.

DataOffset est incrémenté de 8 et DataLength est décrémenté de 8 si la méthode aboutit.

Les 8 caractères des données de message sont supposés être un nombre en virgule flottante binaire. Le codage est spécifié par la propriété MQMessage.Encoding . Notez que la conversion à partir du format System/360 n'est pas prise en charge.

Défini dans: Classe MQMessage

Syntaxe: *doublev# = MQMessage*.**ReadDouble**

Méthode ReadDouble4

Les méthodes ReadDouble4 et WriteDouble4 sont des alternatives à ReadFloat et WriteFloat. En effet, ils prennent en charge les valeurs de message à virgule flottante System/390 à 4 octets qui sont trop grandes pour être converties au format à virgule flottante IEEE à 4 octets.

Cette méthode lit 4 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset et en renvoyant une valeur en virgule flottante double (8 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 4 lorsqu'elle est émise.

DataOffset est incrémenté de 4 et DataLength est décrémenté de 4 si la méthode aboutit.

Les 4 caractères des données de message sont supposés être un nombre binaire à virgule flottante. Le codage est spécifié par la propriété MQMessage.Encoding . Notez que la conversion à partir du format System/360 n'est pas prise en charge.

Défini dans: Classe MQMessage

Syntaxe: *doublev# = MQMessage.*.**ReadDouble4**

Méthode ReadFloat

Cette méthode lit 4 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset et en renvoyant une valeur en virgule flottante unique (4 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 4 lorsqu'elle est émise.

DataOffset est incrémenté de 4 et DataLength est décrémenté de 4 si la méthode aboutit.

Les 4 caractères des données de message sont supposés être un nombre en virgule flottante. Le codage est spécifié par la propriété MQMessage.Encoding . Notez que la conversion à partir du format System/360 n'est pas prise en charge.

Défini dans: Classe MQMessage

Syntaxe: *singlev ! = MQMessage*.**ReadFloat**

Méthode ReadInt2

La méthode est identique à la méthode ReadShort .

Syntaxe: *integerv% = MQMessage*.**ReadInt2**

Méthode ReadInt4

Cette méthode est identique à la méthode ReadLong .

Syntaxe: *bigint & = MQMessage*.**Lecture4**

Méthode ReadLong

Cette méthode lit 4 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset et en le renvoyant sous la forme d'une valeur entière de type Long (4 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 4 lorsqu'elle est émise.

DataOffset est incrémenté de 4 et DataLength est décrémenté de 4 si la méthode aboutit.

Les 4 caractères des données de message sont supposés être un entier binaire. Le codage est spécifié par la propriété MQMessage.Encoding .

Défini dans: Classe MQMessage

Syntaxe: *bigint & = MQMessage*.**ReadLong**

Méthode ReadNullTerminatedString

Cette méthode doit être utilisée à la place de ReadString si la chaîne peut contenir des caractères nuls imbriqués.

Cette méthode lit le nombre d'octets spécifié dans la mémoire tampon de données de message en commençant par l'octet référencé par DataOffset et la renvoie sous la forme d'une chaîne ActiveX . Si la chaîne contient une valeur null imbriquée avant la fin, la longueur de la chaîne renvoyée est réduite pour refléter uniquement les caractères avant la valeur null.

DataOffset est incrémenté et DataLength est décrémenté de la valeur spécifiée, que la chaîne contienne ou non des caractères nuls imbriqués.

Les caractères des données de message sont supposés être une chaîne dans la page de codes spécifiée par la propriété MQMessage.CharacterSet . La conversion en représentation ActiveX est effectuée pour l'application.

Défini dans:
Classe MQMessage

Syntaxe: *string $ = MQMessage*.**ReadNullTerminatedString(length &)**

Paramètres :
longueur & *Long.* Longueur de la zone de chaîne en octets.

Méthode ReadShort

Cette méthode lit 2 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset , puis la renvoie sous forme d'entier (2 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 2 lorsqu'elle est émise.

DataOffset est incrémenté de 2 et DataLength est décrémenté de 2 si la méthode aboutit.

Les 2 caractères des données de message sont supposés être un entier binaire. Le codage est spécifié par la propriété MQMessage.Encoding .

Défini dans: Classe MQMessage

Syntaxe: *integerv% = MQMessage*.**ReadShort**

Méthode ReadString

Cette méthode lit n octets à partir de la mémoire tampon des données de message en commençant par l'octet désigné par DataOffset et la renvoie sous la forme d'une chaîne ActiveX .

La méthode échoue si MQMessage.DataLength est inférieur à n lorsqu'elle est émise.

DataOffset est incrémenté de n et DataLength est décrémenté de n si la méthode aboutit.

Les n caractères des données de message sont supposés être une chaîne de la page de codes spécifiée par la propriété MQMessage.CharacterSet . La conversion en représentation ActiveX est effectuée pour l'application.

Défini dans: Classe MQMessage

Syntaxe: *stringv $ = MQMessage*.**ReadString**(*length &*)

Paramètre

longueur & Long. Longueur de la zone de chaîne en octets.

Méthode ReadUInt2

Cette méthode lit 2 octets à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset , puis la renvoie sous la forme d'un entier long (4 octets signés).

La méthode échoue si MQMessage.DataLength est inférieur à 2 lorsqu'elle est émise.

DataOffset est incrémenté de 2 et DataLength est décrémenté de 2 si la méthode aboutit.

Les 2 octets de données de message sont supposés être un entier binaire non signé. Le codage est spécifié par la propriété MQMessage.Encoding .

Défini dans: Classe MQMessage

Syntaxe: *bigint & = MQMessage*.**IntURead2**

Méthode d'octet ReadUnsigned

Cette méthode lit 1 octet à partir de la mémoire tampon des données de message, en commençant par l'octet désigné par DataOffset , puis la renvoie sous la forme d'un entier (2 octets signés) compris entre 0 et 255.

La méthode échoue si MQMessage.DataLength est inférieur à 1 lorsqu'elle est émise.

DataOffset est incrémenté de 1 et DataLength est décrémenté de 1 si la méthode aboutit.

Le caractère 1 des données de message est supposé être un entier binaire non signé.

Défini dans: Classe MQMessage

Syntaxe: *integerv% = MQMessage*.**ReadUnsignedByte**

Méthode ReadUTF

Cette méthode lit une chaîne de format UTF à partir du message commençant par l'octet référencé par DataOffset et la renvoie sous la forme d'une chaîne ActiveX . La chaîne du message se compose d'une longueur de 2 octets suivie des données de type caractères.

La méthode échoue si MQMessage.DataLength est inférieur à la longueur de la chaîne lorsqu'elle est émise.

DataOffset est incrémenté de la longueur de la chaîne et DataLength est décrémenté de la longueur de la chaîne si la méthode aboutit.

Défini dans:

Classe MQMessage

Syntaxe: *value $ = MQMessage*.**ReadUTF**

Méthode ResizeBuffer

Cette méthode modifie la quantité de mémoire actuellement allouée en interne pour contenir la mémoire tampon des données de message. Elle donne à l'application un certain contrôle sur la gestion automatique de la mémoire tampon, en ce sens que si l'application sait qu'elle va traiter un message de grande taille, elle peut s'assurer qu'une mémoire tampon suffisamment grande est allouée. L'application n'a pas besoin d'utiliser cet appel. Si ce n'est pas le cas, le code de gestion automatique de la mémoire tampon augmente la taille de la mémoire tampon.

Si vous redimensionnez la mémoire tampon pour qu'elle soit plus petite que la MessageLengthen cours, vous risquez de perdre des données. Si vous perdez des données, la méthode renvoie un code CompletionCode de MQCC_WARNING et un code ReasonCode de MQRC_DATA_TRUNCATED.

Si vous redimensionnez la mémoire tampon pour qu'elle soit inférieure à la valeur de la propriété **DataOffset** , procédez comme suit:

- La propriété **DataOffset** est modifiée pour pointer vers la fin de la nouvelle mémoire tampon
- La propriété **DataLength** est définie sur zéro
- La propriété **MessageLength** est remplacée par la nouvelle taille de mémoire tampon

Défini dans:
Classe MQMessage

Syntaxe: *MQMessage***.ResizeBuffer**(*Length &*)

Paramètre:
Longueur & Long. Taille requise en caractères.

Méthode d'écriture

Ecrit une séquence d'octets dans la mémoire tampon de message à partir d'un tableau d'octets à la position indiquée par le décalage de données. Si nécessaire, la longueur de la mémoire tampon (MQMessage.MQMessageLength) est étendue pour tenir compte de la longueur totale du tableau d'octets. DataOffset est incrémenté du nombre d'octets écrits si la méthode aboutit.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**Write**(valeur)

Paramètres :
data: un tableau d'octets ou une variante de référence à un tableau d'octets

Méthode WriteBoolean

Ecrit une valeur booléenne de 1 octet à la position en cours dans la mémoire tampon de messages à partir d'une valeur booléenne de 2 octets. DataOffset est incrémenté de un.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteBoolean**(valeur)

Paramètre:
value: Booléen (2 octets). Valeur à écrire.

Méthode WriteByte

Cette méthode utilise un entier signé de 2 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre binaire de 1 octet à la position indiquée par DataOffset. Elle remplace toutes les données déjà à l'emplacement dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de un si la méthode aboutit.

La valeur indiquée doit être comprise entre -128 et 127. Si ce n'est pas le cas, la méthode renvoie CompletionCode MQCC_FAILED et ReasonCode MQRC_WRITE_VALEUR_ERROR.

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage***.WriteByte**(*value%*)

Paramètre: *value%* Entier. Valeur à écrire.

Méthode WriteDecimal2

Ecrit un entier signé de 2 octets sous la forme d'un nombre décimal condensé de 2 octets. DataOffset est incrémenté de deux.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteDecimal2**(value%)

Paramètre:
value% entier. Valeur à écrire.

Méthode WriteDecimal4

Ecrit un entier signé de 4 octets sous la forme d'un nombre décimal condensé de 4 octets. DataOffset est incrémenté de quatre.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**WritedDecimal4**(value &)

Paramètre:
Valeur & Long. Valeur à écrire.

Méthode WriteDouble

Cette méthode utilise une valeur à virgule flottante signée de 8 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre à virgule flottante de 8 octets à partir de la position indiquée par DataOffset. Elle remplace les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 8 si la méthode aboutit.

La méthode est convertie en représentation en virgule flottante spécifiée par la propriété MQMessage.Encoding . *La conversion au format System/360 n'est pas prise en charge.*

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteDouble**(*value#*)

Paramètre:
value# Double. Valeur à écrire.

Méthode WriteDouble4

Voir «Méthode ReadDouble4», à la page 1117 pour une description du moment où ReadDouble4 et WriteDouble4 doivent être utilisés à la place de ReadFloat et WriteFloat.

Cette méthode utilise une valeur en virgule flottante signée de 8 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre flottant de 4 octets à partir de la position indiquée par DataOffset.

DataOffset est incrémenté de 4 si la méthode aboutit.

Elle remplace les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

La méthode est convertie en représentation en virgule flottante spécifiée par la propriété MQMessage.Encoding . *La conversion au format System/360 n'est pas prise en charge.*

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteDouble4 (value#)**

Paramètre: *value#* Double. Valeur à écrire.

Méthode WriteFloat

Cette méthode utilise une valeur en virgule flottante signée de 4 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre en virgule flottante de 4 octets commençant au caractère désigné par DataOffset. Elle remplace les données qui se trouvent déjà à ces emplacements

dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 4 si la méthode aboutit.

La méthode est convertie en représentation binaire spécifiée par la propriété MQMessage.Encoding . *La conversion au format System/360 n'est pas prise en charge.*

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteFloat**(*valeur !*)

Valeur du paramètre *!* Variable flottante. Valeur à écrire.

Méthode WriteInt2

Cette méthode est identique à la méthode WriteShort .

Syntaxe: Call *MQMessage*.**WriteInt2**(*valeur%*)

Paramètre *value%* Entier. Valeur à écrire.

Méthode WriteInt4

Cette méthode est identique à la méthode WriteLong .

Syntaxe: Call *MQMessage*.**WriteInt4**(*valeur &*)

Paramètre *valeur &* Long. Valeur à écrire.

Méthode WriteLong

Cette méthode utilise une valeur entière signée de 4 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre binaire de 4 octets à partir de l'octet désigné par DataOffset. Elle remplace les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 4 si la méthode aboutit.

La méthode est convertie en représentation binaire spécifiée par la propriété MQMessage.Encoding .

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteLong**(*valeur &*)

Paramètre *valeur &* Long. Valeur à écrire.

Méthode WriteNullTerminatedString

Cette méthode effectue une opération WriteString normale et remplit tous les octets restants jusqu'à la longueur spécifiée avec la valeur null. Si le nombre d'octets écrits par la chaîne d'écriture initiale est égal à la longueur spécifiée, aucune valeur nulle n'est écrite. Si le nombre d'octets dépasse la longueur spécifiée, une erreur (code anomalie MQRC_WRITE_VALEUR_ERROR) est définie.

DataOffset est incrémenté de la longueur spécifiée si la méthode aboutit.

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteNullTerminatedString***(value$, length &)*

Paramètres :
value $ Chaîne. Valeur à écrire.

longueur & Long. Longueur de la zone de chaîne en octets.

Méthode WriteShort

Cette méthode utilise un entier signé de 2 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre binaire de 2 octets à partir de l'octet référencé par DataOffset. Il

remplace toutes les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 2 si la méthode aboutit.

La méthode est convertie en représentation binaire spécifiée par la propriété MQMessage.Encoding .

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteShort**(*value%*)

Paramètre *value%* Entier. Valeur à écrire.

Méthode WriteString

Cette méthode utilise une chaîne ActiveX et l'écrit dans la mémoire tampon des données de message à partir de l'octet désigné par DataOffset. Il remplace toutes les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de la longueur de la chaîne en octets si la méthode aboutit.

La méthode convertit les caractères dans la page de codes spécifiée par la propriété MQMessage.CharacterSet .

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteString**(*valeur $*)

Paramètre *valeur $* Chaîne. Valeur à écrire.

Méthode WriteUInt2

Cette méthode utilise une valeur entière signée de 4 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre binaire non signé de 2 octets à partir de l'octet désigné par DataOffset. Elle remplace les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 2 si la méthode aboutit.

La méthode est convertie en représentation binaire spécifiée par la propriété MQMessage.Encoding . La valeur indiquée doit être comprise entre 0 et 2 * *16-1. Si ce n'est pas le cas, la méthode renvoie CompletionCode MQCC_FAILED et ReasonCode MQRC_WRITE_VALEUR_ERROR.

Défini dans: Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteUInt2**(*valeur &*)

Paramètre *valeur &* Long. Valeur à écrire.

Méthode d'octet WriteUnsigned

Cette méthode utilise un entier signé de 2 octets et l'écrit dans la mémoire tampon des données de message sous la forme d'un nombre binaire non signé de 1 octet à partir du caractère désigné par DataOffset. Elle remplace les données qui se trouvent déjà à ces emplacements dans la mémoire tampon et étend la longueur de la mémoire tampon (MQMessage.MessageLength) si nécessaire.

DataOffset est incrémenté de 1 si la méthode aboutit.

La valeur indiquée doit être comprise entre 0 et 255. Si ce n'est pas le cas, la méthode renvoie CompletionCode MQCC_FAILED et ReasonCode MQRC_WRITE_VALEUR_ERROR.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteUnsignedByte**(*value%*)

Paramètre *value%* Entier. Valeur à écrire.

Méthode WriteUTF

Cette méthode utilise une chaîne ActiveX et l'écrit dans la mémoire tampon de données de message à la position actuelle au format UTF. Les données écrites se composent d'une longueur de 2 octets suivie des données de type caractères. DataOffset est incrémenté de la longueur de la chaîne si la méthode aboutit.

Défini dans:
Classe MQMessage

Syntaxe: Call *MQMessage*.**WriteUTF**(value$)

Paramètre:
valeur $ Chaîne. Valeur à écrire.

Classe MQPutMessageOptions

Cette classe encapsule les différentes options qui contrôlent l'action d'insertion d'un message dans une file d'attente WebSphere MQ .

Confinement

La classe MQPutMessageOptions est contenue dans la classe MQSession.

Création

Nouveau crée un nouvel objet d'options MQPutMessageet définit toutes ses propriétés sur des valeurs initiales.

Vous pouvez également utiliser la méthode AccessPutMessageOptions de la classe MQSession.

Syntaxe

Dim *pmo* **En tant que Nouveau MQPutMessageOptions** ou

Set *pmo* = **Nouveau MQPutMessageOptions**

Propriétés

- «Propriété CompletionCode», à la page 1124.
- «Propriété Options», à la page 1125.
- «Propriété ReasonCode», à la page 1125.
- «Propriété ReasonName», à la page 1125.
- «Propriété RecordFields», à la page 1125.
- «Propriété ResolvedQueueManagerName», à la page 1126.
- «Propriété de nom ResolvedQueue», à la page 1126.

Des méthodes

- «Méthode des codes ClearError», à la page 1126.

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: MQPutMessageClasse Options

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode & = PutOpts***.CompletionCode**

Propriété Options

Lecture-écriture. Zone Options MQPMO. La valeur initiale de cette zone est MQPMO_NONE. Pour plus d'informations, voir Options MQPMO.

Définie dans: MQPutMessageClasse d'options.

Type de données: Long

Syntaxe: Pour obtenir: *options & = PutOpts***.Options**

A définir: *PutOpts***.Options** = *options &*

Les options MQPMO_PASS_IDENTITY_CONTEXT et MQPMO_PASS_ALL_CONTEXT ne sont pas prises en charge.

Propriété ReasonCode

Lecture seule. Renvoie le code raison défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: MQPutMessageClasse Options

Type de données: Long

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasoncode & = PutOpts***.ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Défini dans: MQPutMessageClasse Options

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: To get: *reasonname $ = PutOpts***.ReasonName**

Propriété RecordFields

Lecture-écriture. Indicateurs indiquant les zones à personnaliser par file d'attente lors de l'insertion d'un message dans une liste de distribution. La valeur initiale est zéro.

Cette propriété correspond aux indicateurs PutMsgRecFields dans la structure MQI MQPMO. Dans l'interface MQI, ces indicateurs contrôlent les zones (dans la structure MQPMR) qui sont présentes et utilisées par MQPUT. Dans un objet MQPutMessageOptions, ces zones sont toujours présentes et les indicateurs n'affectent donc que les zones utilisées par l'insertion. Pour plus d'informations, voir *WebSphere MQ Application Programming Reference* .

Défini dans:
Classe MQPutMessageOptions

Type de données :
Long

Syntaxe: Pour obtenir: *recordfields & = PutOpts*.**RecordFields**

A définir: *PutOpts*.**RecordFields** = *recordfields &*

Propriété ResolvedQueueManagerName

Lecture seule. Zone de nom ResolvedQMgrdu MQPMO. Pour plus d'informations, voir ResolvedQMgrName (MQCHAR48) . La valeur initiale est vide.

Défini dans: MQPutMessageClasse Options

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *qmgr $ = PutOpts*.**ResolvedQueueManagerName**

Propriété de nom ResolvedQueue

Lecture seule. Zone ResolvedQName du MQPMO. Pour plus d'informations, voir ResolvedQName (MQCHAR48) . La valeur initiale est vide.

Défini dans: MQPutMessageClasse Options

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *qname $ = PutOpts*.**ResolvedQueueNom**

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQPutMessageOptions et la classe MQSession.

Défini dans: MQPutMessageClasse Options

Syntaxe: Call *PutOpts*.**ClearErrorCodes ()**

Classe d'options MQGetMessage

Cette classe encapsule les différentes options qui contrôlent l'action d'obtention d'un message à partir d'une file d'attente WebSphere MQ .

Confinement

La classe MQGetMessageOptions est contenue dans la classe MQSession.

Création

Nouveau crée un nouvel objet d'options MQGetMessageet définit toutes ses propriétés sur des valeurs initiales.

Vous pouvez également utiliser la méthode AccessGetMessageOptions de la classe MQSession.

Propriétés

Des méthodes

- «Méthode des codes ClearError», à la page 1128

Syntaxe

Dim *gmo* **As New MQGetMessageOptions** ou

Set *gmo* = **Nouveau MQGetMessageOptions**

Propriété CompletionCode

Lecture seule. Renvoie le code achèvement défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: MQGetMessageClasse d'options.

Type de données: Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode & = GetOpts*.**CompletionCode**

Propriété MatchOptions

Lecture-écriture. Options contrôlant les critères de sélection utilisés pour MQGET. La valeur initiale est MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID.

Défini dans:
Classe d'options MQGetMessage

Type de données :
Long

Valeurs :
Voir MatchOptions (MQLONG).

Syntaxe: Pour obtenir: *matchoptions & = GetOpts*.**MatchOptions**

A définir: *GetOpts*.**MatchOptions** =*matchoptions &*

Propriété Options

Lecture-écriture. Zone Options MQGMO. Pour plus d'informations, voir Options . La valeur initiale est MQGMO_NO_WAIT.

Défini dans: MQGetMessageClasse d'options.

Type de données: Long

Syntaxe: Pour obtenir: *options & = GetOpts*.**Options** A définir: *GetOpts*.**Options** = *options &*

Propriété ReasonCode

Lecture seule. Renvoie le code raison défini par la dernière méthode ou l'accès à la propriété émis pour l'objet.

Défini dans: MQGetMessageClasse Options

Type de données: Long

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasoncode & = GetOpts*.**ReasonCode**

Propriété ReasonName

Lecture seule. Renvoie le nom symbolique du dernier code anomalie. Par exemple, "MQRC_QMGR_NOT_AVAILABLE". **Défini dans:** MQGetMessageClasse Options

Type de données : chaîne

Valeurs :

- Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQGetMessageOptions*.**ReasonName**

Propriété de nom ResolvedQueue

Lecture seule. Zone ResolvedQName de MQGMO. Pour plus d'informations, voir ResolvedQName (MQCHAR48) . La valeur initiale est vide.

Défini dans: MQGetMessageClasse Options

Type de données: chaîne de 48 caractères

Syntaxe: Pour obtenir: *qname $ = GetOpts*.**ResolvedQueueNom**

Propriété WaitInterval

Lecture/Ecriture. Zone WaitInterval MQGMO. Durée maximale, en millisecondes, pendant laquelle l'opération d'obtention attend l'arrivée d'un message approprié-si l'action d'attente a été demandée par la propriété Options. Cette zone a une valeur initiale de 0. Pour plus de détails sur les options MQGMO, voir MQGMO.

Défini dans: MQGetMessageClasse Options

Type de données: Long

Syntaxe: Pour obtenir: *wait & = GetOpts*.**WaitInterval**

A définir: *GetOpts*.**WaitInterval** = wait &

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe d'options MQGetMessageet la classe MQSession.

Défini dans: MQGetMessageClasse Options

Syntaxe: Call *GetOpts*.**ClearErrorCodes ()**

Classe MQDistributionList

Cette classe encapsule une collection de files d'attente-locales, distantes ou alias pour la sortie.

Création

new crée un nouvel objet MQDistributionList .

Vous pouvez également utiliser la méthode AddDistributionList de la classe MQQueueManager

Propriétés

- «Propriété d'ID AlternateUser», à la page 1129
- «Propriété CloseOptions», à la page 1129
- «Propriété CompletionCode», à la page 1129

Méthode

Syntaxe

Dim *distlist.***As New MQDistributionList** ou **Set** *distlist* = **New MQDistributionList**

Propriété d'ID AlternateUser

Lecture-écriture. ID utilisateur de remplacement utilisé pour valider l'accès à la liste des files d'attente lorsqu'elles sont ouvertes.

Défini dans:
Classe MQDistributionList

Type de données :
Chaîne de 12 caractères

Syntaxe: Pour obtenir: *altuser $ = MQDistributionList*.**AlternateUserId**

A définir: *MQDistributionList*.**AlternateUserId** = *altuser $*

Propriété CloseOptions

Lecture-écriture. Options utilisées pour contrôler ce qui se passe lorsque la liste de distribution est fermée. La valeur initiale est MQCO_NONE.

Défini dans:
Classe MQDistributionList

Type de données :
Long

Valeurs :

- MQCO_AUCUN
- MQCO_DELETE
- MQCO_DELETE_PURGE

Syntaxe: Pour obtenir: *closeopt & = MQDistributionList*.**CloseOptions**

A définir: *MQDistributionList*.**CloseOptions** = *closeopt &*

Propriété CompletionCode

Lecture seule. Code achèvement défini par le dernier accès à la méthode ou à la propriété émis sur l'objet.

Défini dans:
Classe MQDistributionList

Type de données :
Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode & = MQDistributionList*.**CompletionCode**

Propriété ConnectionReference

Lecture-écriture. Gestionnaire de files d'attente auquel appartient la liste de distribution.

Défini dans:
Classe MQDistributionList

Type de données :
MQQueueManager

Syntaxe: Pour obtenir: *set queuemanager = MQDistributionList*.**ConnectionReference**

A définir: *set MQDistributionList*. **ConnectionReference** = *gestionnaire de files d'attente*

Propriété FirstDistributionListItem

Lecture seule. Premier objet élément de liste de distribution associé à la liste de distribution.

Défini dans:
Classe MQDistributionList

Type de données :
Elément MQDistributionList

Valeurs :

Syntaxe: Pour obtenir: *set distributionlistitem = MQDistributionList*.**FirstDistributionListItem**

Propriété IsOpen

Lecture seule.

Défini dans:
Classe MQDistributionList

Type de données :
Boolean

Valeurs :

- TRUE (-1)
- FALSE (0)

Syntaxe: Pour obtenir: *IsOpen = MQDistributionList*.**IsOpen**

Propriété OpenOptions

Lecture-écriture. Options à utiliser lors de l'ouverture de la liste de diffusion.

Défini dans:
Classe MQDistributionList

Type de données :
Long

Valeurs :
Voir Options MQPMO.

Syntaxe: Pour obtenir: *openopt & = MQDistributionList*.**OpenOptions**

A définir: *MQDistributionList.***OpenOptions** = openopt &

Propriété ReasonCode

Lecture seule. Code raison défini par la dernière méthode ou le dernier accès à la propriété émis pour l'objet.

Défini dans:
Classe MQDistributionList

Type de données :
Long

Valeurs :
Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasoncode & = MQDistributionList*.**ReasonCode**

Propriété ReasonName

Lecture seule. Nom symbolique de ReasonCode. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Défini dans:
Classe MQDistributionList

Type de données :
String

Valeurs :
Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQDistributionList*.**ReasonName**

Méthode AddDistributionListItem

Crée un objet d'élément MQDistributionListet l'associe à l'objet de liste de distribution. Le paramètre de nom de file d'attente est obligatoire.

La propriété DistributionList de l'élément de liste de distribution est définie sur la liste de distribution propriétaire et la propriété FirstDistributionListItem de la liste de distribution est définie pour référencer ce nouvel élément de liste de distribution.

Pour le nouvel élément de liste de distribution, la propriété PreviousDistributionListItem est définie sur rien et la propriété NextDistributionListItem est définie pour référencer tout élément de liste de distribution qui était auparavant le premier, ou rien s'il n'en existait pas auparavant (c'est-à-dire que le nouvel élément est inséré devant ceux qui existent déjà).

Une erreur est renvoyée si la liste de distribution est ouverte.

Défini dans:
Classe MQDistributionList

Syntaxe: set distributionlistitem = *MQDistributionList*.**AddDistributionListItem** (QName$, QMgrName$)

Paramètres :

QName$ Chaîne. Nom de la file d'attente WebSphere MQ .

QMgrName$ Chaîne. Nom du gestionnaire de files d'attente WebSphere MQ .

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQDistributionList et la classe MQSession.

Défini dans:
Classe MQDistributionList

Syntaxe: Call *MQDistributionList*.**ClearErrorCodes**()

méthode Close

Ferme une liste de distribution à l'aide de la valeur en cours des options de fermeture.

Défini dans:
Classe MQDistributionList

Syntaxe: Call *MQDistributionList*.**Close**()

méthode Open

Ouvre chacune des files d'attente spécifiées par les propriétés QueueName et (le cas échéant) QueueManagerdes éléments de liste de distribution associés à l'objet en cours à l'aide de la valeur en cours de l'ID AlternateUser.

Défini dans:
Classe MQDistributionList

Syntaxe: Call *MQDistributionList*.**Open**()

PUT (méthode)

Place un message dans chacune des files d'attente identifiées par les éléments de liste de distribution associés à la liste de distribution.

Défini dans:

Classe MQDistributionList

Syntaxe

Appelez MQDistributionList.**Put**(Message, PutMsgOptions &)

Paramètres

Message Objet MQMessage représentant le message à insérer.

PutMsg MQPutMessagecontenant des options permettant de contrôler l'opération d'insertion. S'il n'est pas spécifié, les options PutMessagepar défaut sont utilisées.

Cette méthode utilise un objet MQMessage comme paramètre. Les propriétés d'élément de liste de distribution suivantes peuvent être modifiées à la suite de cette méthode:

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdValeur hexadécimale
- CorrelationId
- CorrelationIdValeur hexadécimale
- GroupId
- GroupId-Valeur hexadécimale
- Commentaires
- AccountingToken
- AccountingTokenValeur hexadécimale

Classe d'élément MQDistributionList

Cette classe encapsule les structures MQOR, MQRR et MQPMR et les associe à une liste de distribution propriétaire.

Création

Utilisez la méthode AddDistributionListItem de la classe MQDistributionList

Propriétés

Des méthodes

Propriétés :

- Propriété AccountingToken
- Propriété hexadécimale AccountingToken
- Propriété CompletionCode
- Propriété CorrelationId
- Propriété hexadécimale CorrelationId
- Propriété DistributionList
- Propriété de commentaire en retour
- Propriété GroupId
- Propriété hexadécimale GroupId
- Propriété MessageId
- Propriété hexadécimale MessageId
- Propriété NextDistributionListItem
- Propriété PreviousDistributionListItem
- Propriété de nom QueueManager
- Propriété QueueName

- Propriété ReasonCode
- Propriété ReasonName

Methods:

- Méthode des codes ClearError

Création:

Utilisez la méthode AddDistributionListItem de la classe MQDistributionList

Propriété AccountingToken

Lecture-écriture. AccountingToken à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Sa valeur initiale est toutes des valeurs nulles.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 32 caractères

Syntaxe: Pour obtenir: *accountingtoken $ = MQDistributionListItem*.**AccountingToken**

A définir: *MQDistributionListElément*.**AccountingToken** = *accountingtoken $*

Propriété hexadécimale AccountingToken

Lecture-écriture. AccountingToken à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représente le caractère unique "A", la paire de caractères "6" et "2" représente le caractère unique "B", etc.

Vous devez indiquer 64 caractères hexadécimaux valides.

Sa valeur initiale est "0 ... 0".

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 64 caractères hexadécimaux représentant 32 caractères ASCII.

Syntaxe: Pour obtenir: *accountingtokenh $ = MQDistributionListItem*.**AccountingTokenHex**

Pour définir: *MQDistributionListItem*.**AccountingTokenHex** = *accountingtokenh $*

Propriété CompletionCode

Lecture seule. Code achèvement défini par la dernière demande d'ouverture ou d'insertion émise sur l'objet de liste de distribution propriétaire.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Long

Valeurs :

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

Syntaxe: Pour obtenir: *completioncode $ = MQDistributionListItem*.**CompletionCode**

Propriété CorrelationId

Lecture-écriture. CorrelId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Sa valeur initiale est toutes des valeurs nulles.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 24 caractères

Syntaxe: Pour obtenir: *correlid $ = MQDistributionListItem*.**CorrelationId**

A définir: *MQDistributionListItem*.**CorrelationId** = *correlid $*

Propriété hexadécimale CorrelationId

Lecture-écriture. CorrelId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représente le caractère unique "A", la paire de caractères "6" et "2" représente le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 .. 0".

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII.

Syntaxe: Pour obtenir: *correlidh $ = MQDistributionListItem*.**CorrelationIdHex**

A définir: *MQDistributionListItem*.**CorrelationIdHex** = *correlidh $*

Propriété DistributionList

Lecture seule. Liste de distribution à laquelle cet élément de liste de distribution est associé.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
MQDistributionList

Syntaxe: Pour obtenir: *set distributionlist = MQDistributionListElément*.**DistributionList**

Propriété de commentaire en retour

Lecture-écriture. Valeur Feedback à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Long

Valeurs :
Voir Feedback (MQLONG).

Syntaxe: Pour obtenir: *feedback & = MQDistributionListItem*.**Feedback**

A définir: *MQDistributionListItem*.**Feedback** = *feedback &*

Propriété GroupId

Lecture-écriture. GroupId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Sa valeur initiale est toutes des valeurs nulles.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 24 caractères

Syntaxe: Pour obtenir: *groupid $ = MQDistributionListItem*.**GroupId**

A définir: *MQDistributionListItem*.**GroupId** = *groupid $*

Propriété hexadécimale GroupId

Lecture-écriture. GroupId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représente le caractère unique "A", la paire de caractères "6" et "2" représente le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 .. 0".

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII.

Syntaxe: Pour obtenir: *groupidh $ = MQDistributionListItem*.**GroupIdHex**

A définir: *MQDistributionListItem*.**GroupIdHex** = *groupidh $*

Propriété MessageId

Lecture-écriture. MessageId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente. Sa valeur initiale est toutes des valeurs nulles.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 24 caractères

Syntaxe: Pour obtenir: *messageid $ = MQDistributionListItem*.**MessageId**

A définir: *MQDistributionListItem*.**MessageId** = *messageid $*

Propriété hexadécimale MessageId

Lecture-écriture. MessageId à inclure dans le MQPMR d'un message lorsqu'il est inséré dans une file d'attente.

Tous les deux caractères de la chaîne représentent l'équivalent hexadécimal d'un caractère ASCII unique. Par exemple, la paire de caractères "6" et "1" représente le caractère unique "A", la paire de caractères "6" et "2" représente le caractère unique "B", etc.

Vous devez fournir 48 caractères hexadécimaux valides.

Sa valeur initiale est "0 .. 0".

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 48 caractères hexadécimaux représentant 24 caractères ASCII.

Syntaxe: Pour obtenir: *messageidh $ = MQDistributionListItem*.**MessageIdHex**

A définir: *MQDistributionListItem*.**MessageIdHex** = *messageidh $*

Propriété NextDistributionListItem

Lecture seule. Objet d'élément de liste de distribution suivant associé à la même liste de distribution.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Elément MQDistributionList

Syntaxe: Pour obtenir: *set distributionlistitem = MQDistributionListItem*.**NextDistributionListItem**

Propriété PreviousDistributionListItem

Lecture seule. Objet d'élément de liste de distribution précédent associé à la même liste de distribution.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Elément MQDistributionList

Syntaxe: Pour obtenir: *set distributionlistitem = MQDistributionListItem*.**PreviousDistributionListItem**

Propriété de nom QueueManager

Lecture-écriture. Nom du gestionnaire de files d'attente WebSphere MQ .

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 48 caractères.

Syntaxe: Pour obtenir: *qmname $ = MQDistributionListItem*.**QueueManagerName**

A définir: *MQDistributionListElément*.**QueueManagerNom** = *qmname $*

Propriété QueueName

Lecture-écriture. Nom de la file d'attente WebSphere MQ .

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Chaîne de 48 caractères.

Syntaxe: Pour obtenir: *qname $ = MQDistributionListElément*.**QueueName**

A définir: *MQDistributionListItem*.**QueueName** = *qname $*

Propriété ReasonCode

Lecture seule. Code achèvement défini par la dernière ouverture ou insertion émise pour l'objet de liste de distribution propriétaire.

Défini dans:
Classe d'élément MQDistributionList

Type de données :
Long

Valeurs :
Voir Codes anomalie d'API.

- MQCC_OK
- MQCC_WARNING

- MQCC_FAILED

Syntaxe: Pour obtenir: *reasoncode & = MQDistributionListItem*.**ReasonCode**

Propriété ReasonName

Lecture seule. Nom symbolique de ReasonCode. Par exemple, "MQRC_QMGR_NOT_AVAILABLE".

Défini dans:
Classe d'élément MQDistributionList

Type de données :
String

Valeurs :
Voir Codes anomalie d'API.

Syntaxe: Pour obtenir: *reasonname $ = MQDistributionListItem*.**ReasonName**

Méthode des codes ClearError

Réinitialise CompletionCode sur MQCC_OK et ReasonCode sur MQRC_NONE pour la classe MQDistributionListItem et la classe MQSession.

Défini dans:
Classe d'élément MQDistributionList

Syntaxe: Call *MQDistributionListItem*.**ClearErrorCodes**

Identification et résolution des problèmes

Des informations sur la fonction de trace fournie, des pièges communs et de l'aide sur la façon de les éviter.

La section suivante explique la fonction de trace fournie et détaille les pièges courants, avec une aide pour les éviter:

Utilisation de la trace

MQAX inclut une fonction de trace pour aider l'organisation de service à identifier ce qui se passe lorsque vous rencontrez un problème. Il affiche les chemins d'accès empruntés lors de l'exécution de votre script MQAX. A moins que vous ne soyez confronté à un problème, exécutez la fonction de trace désactivée pour éviter toute utilisation inutile des ressources système.

Vous définissez trois variables d'environnement pour contrôler le traçage :

- OMQ_TRACE
- CHEMIN_TRACE_MQ
- OMQ_NIV_TRACE

Notez que la spécification de la valeur *any* pour OMQ_TRACE active la fonction de trace. Même si vous définissez OMQ_TRACE sur OFF, la trace est toujours active.

Pour désactiver la trace, n'indiquez pas de valeur pour OMQ_TRACE.

1. Cliquez sur **Démarrer**
2. Cliquez sur **Panneau de configuration**
3. Cliquez deux fois sur **Système**

4. Cliquez sur **AVANCEE**
5. Cliquez sur **Environnement**.
6. Dans la section intitulée "User variables for (username)", cliquez sur **New**
7. Entrez le nom de la variable et une valeur valide dans les zones appropriées et cliquez sur **OK** .
8. Cliquez sur **OK** pour fermer la fenêtre Variables d'environnement.
9. Cliquez sur **OK** pour fermer la fenêtre Propriétés système.
10. Fermer la fenêtre Panneau de configuration

Lorsque vous décidez où vous souhaitez que les fichiers de trace soient écrits, vérifiez que vous disposez des droits suffisants pour écrire sur le disque, et pas seulement pour le lire.

Lorsque la fonction de trace est activée, elle ralentit l'exécution de MQAX, mais n'affecte pas les performances de vos environnements ActiveX ou WebSphere MQ . Lorsque vous n'avez plus besoin d'un fichier de trace, vous pouvez le supprimer.

Vous devez arrêter l'exécution de MQAX pour modifier le statut de la variable OMQ_TRACE.

Nom et répertoire du fichier de trace

Le nom du fichier de trace est au format OMQnnnnn.trc, où nnnnn est l'ID du processus ActiveX en cours d'exécution.

Tableau 157. Commandes et leurs effets

Commande	Effet
SET OMQ_TRACE_PATH = unité: \répertoire	Définit le répertoire d'écriture du fichier de trace.
SET OMQ_TRACE_PATH =	Supprime la variable d'environnement OMQ_PATH dans laquelle le répertoire de travail en cours (lorsque ActiveX est démarré) est utilisé.
ECHO %OMQ_TRACE_PATH%	Affiche le paramètre en cours du répertoire de trace sous Windows.
SET OMQ_TRACE = xxxxxxxx	La fonction de trace est activée. Vous activez la fonction de trace en plaçant un ou plusieurs caractères après le signe'='. Par exemple: SET OMQ_TRACE=yes SET OMQ_TRACE = no. Dans ces deux exemples, la fonction de trace est activée. Cela ne s'applique qu'à une seule fenêtre/session
SET OMQ_TRACE=	Désactive la fonction de trace
ECHO %OMQ_TRACE%	Affiche le contenu de la variable d'environnement sous Windows.
SET	Affiche le contenu de toutes les variables d'environnement sous Windows.
SET OMQ_TRACE_LEVEL = 9	Définit le niveau de trace sur 9. Les valeurs supérieures à 9 ne produisent pas d'informations supplémentaires dans le fichier de trace.

Lorsque votre script WebSphere MQ Automation Classes for ActiveX échoue

Si votre script WebSphere MQ Automation Classes for ActiveX échoue, il existe un certain nombre de sources d'informations.

Premier rapport de symptôme d'échec

Indépendamment de la fonction de trace, pour les erreurs inattendues et internes, un rapport de symptôme de premier échec peut être généré.

Ce rapport se trouve dans un fichier nommé OMQnnnnn.fdc, où nnnnn est le numéro du processus ActiveX en cours d'exécution. Ce fichier se trouve dans le répertoire de travail à partir duquel vous avez démarré ActiveX ou dans le chemin indiqué dans la variable d'environnement OMQ_PATH.

Autres sources d'informations

WebSphere MQ fournit divers journaux d'erreurs et informations de trace, en fonction de la plateforme impliquée. Consultez le journal des événements d'application Windows NT.

Codes raison

Les codes anomalie suivants peuvent être générés en plus de ceux documentés pour l'interface MQI WebSphere MQ . Pour les autres codes, consultez le journal des événements d'application WebSphere MQ .

Tableau 158. Codes raison et signification

Code raison	Explication
MQRC_LIBRARY_LOAD_ERROR (6000)	Une ou plusieurs des bibliothèques WebSphere MQ n'ont pas pu être chargées. Vérifiez que toutes les bibliothèques WebSphere MQ se trouvent dans le chemin de recherche correct sur le système que vous utilisez. Par exemple, assurez-vous que les répertoires contenant les bibliothèques WebSphere MQ se trouvent dans PATH.
MQRC_CLASS_LIBRARY_ERROR (6001)	L'un des appels de la bibliothèque de classes WebSphere MQ a renvoyé un code ReasonCode/CompletionCodeinattendu. Pour plus de détails, consultez le rapport sur les symptômes de la première défaillance. Prenez note de la dernière méthode / propriété et de la dernière classe utilisées et informez le support IBM du problème.
MQRC_STRING_LENGTH_TOO_BIG (6002)	Une tentative d'écriture d'une chaîne de format UTF d'une longueur supérieure à 65 535 octets a été effectuée dans la mémoire tampon de messages.
MQRC_WRITE_VALEUR_ERREUR (6003)	Une valeur hors plage est utilisée ; par exemple, msg.WriteByte (240).
MQRC_PACKED_DECIMAL_ERROR (6004)	Une tentative de lecture d'un nombre décimal condensé à partir de la mémoire tampon du message a été effectuée, mais les données au niveau du pointeur de données ne sont pas dans un format de données condensées valide.
MQRC_FLOAT_CONVERSION_ERROR (6005)	Une tentative de lecture d'un nombre à virgule flottante simple ou double a été effectuée dans la mémoire tampon du message, mais les données du pointeur de données ne sont pas dans un format à virgule flottante approprié.
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	Un objet ouvert ne possède pas les **OpenOptions** correctes et requiert une ou plusieurs options supplémentaires. Une réouverture implicite est requise, mais la fermeture a été empêchée. Définissez **OpenOptions**de manière explicite pour couvrir toutes les éventualités de sorte qu'une réouverture implicite ne soit pas requise. La fermeture a été empêchée car la file d'attente est ouverte pour une entrée exclusive et la fermeture permettrait à d'autres personnes d'accéder à la file d'attente.
MQRC_REOPEN_INQUIRE_ERROR (6101)	Un objet ouvert ne possède pas les OpenOptions correctes et requiert une ou plusieurs options supplémentaires. Une réouverture implicite est requise, mais la fermeture a été empêchée. Définissez OpenOptions de manière explicite pour inclure MQOO_INQUIRE. La fermeture a été empêchée car une ou plusieurs caractéristiques de l'objet doivent être vérifiées dynamiquement avant la fermeture et les OpenOptions n'incluent pas déjà MQOO_INQUIRE.

Tableau 158. Codes raison et signification (suite)

Code raison	Explication
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Un objet ouvert ne possède pas les OpenOptions correctes et requiert une ou plusieurs options supplémentaires. Une réouverture implicite est requise, mais la fermeture a été empêchée. Définissez explicitement OpenOptions pour couvrir toutes les éventualités de sorte que la réouverture implicite ne soit pas requise. La fermeture a été empêchée car la file d'attente est ouverte avec MQOO_SAVE_ALL_CONTEXT et une opération Get destructive a été effectuée précédemment. Cela a entraîné l'association des informations d'état conservées à la file d'attente ouverte et ces informations seront détruites par la fermeture.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Un objet ouvert ne possède pas les **OpenOptions** correctes et requiert une ou plusieurs options supplémentaires. Une réouverture implicite est requise, mais la fermeture a été empêchée. Définissez **OpenOptions**de manière explicite pour couvrir toutes les éventualités de sorte qu'une réouverture implicite ne soit pas requise. La fermeture a été empêchée car la file d'attente est une file d'attente locale du type de définition MQQDT_TEMPORARY_DYNAMIC, qui serait détruite par la fermeture.
MQRC_ATTRIBUTE_LOCKED (6104)	Une tentative de modification de la valeur ou de l'attribut d'un objet a été effectuée alors que cet objet est ouvert. Certains attributs, tels que **AlternateUserId**, ne peuvent pas être modifiés lorsqu'un objet est ouvert.
MQRC_CURSOR_NOT_VALID (6105)	Le curseur de navigation d'une file d'attente ouverte a été invalidé depuis sa dernière utilisation par une réouverture implicite. Définissez explicitement OpenOptions pour couvrir toutes les éventualités de sorte que la réouverture implicite ne soit pas requise.
MQRC_ENCODING_ERROR (6106)	Le codage de l'élément de message suivant doit être MQENC_NATIVE pour la lecture.
MQRC_STRUCID_ERROR (6107)	La structure de l'ID de l'élément de message suivant, qui est dérivée des 4 caractères commençant par le pointeur de données, est manquante ou incohérente avec le type de variable dans lequel l'élément est lu.
MQRC_NULL_POINTER (6108)	Un pointeur null a été fourni alors qu'un pointeur non null est obligatoire ou implicite. Cela peut être dû à l'utilisation de déclarations explicites pour les objets WebSphere MQ utilisés à partir de VBA en tant que paramètres pour les appels (par exemple, dim msg as Object is ok, dim msg as MqMessage peut entraîner des problèmes). Par exemple, dans Excel, avec q défini et dim msg défini comme MqMessageq.put msg donne reasonCode MQRC_NULL_POINTER. Il fonctionne correctement à partir de VisualBasic.
MQRC_NO_CONNECTION_REFERENCE (6109)	L'objet **MQQueue** a perdu sa connexion à **MQQueueManager**. Cela se produit si **MQQueueManager**est déconnecté. Supprimez l'objet **MQQueue** .
MQRC_NO_BUFFER (6110)	Aucune mémoire tampon n'est disponible. Pour un objet **MQMessage** , il est impossible d'en attribuer un, ce qui indique une incohérence interne dans l'état de l'objet qui ne doit pas se produire.
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	La longueur des données binaires est incohérente avec la longueur de l'attribut cible. La valeur zéro est une longueur correcte pour tous les attributs. 24 est une longueur correcte pour un **CorrelationId** et pour un **MessageId** 32 est une longueur correcte pour un **AccountingToken**
MQRC_BUFFER_NOT_AUTOMATIC (6112)	Une mémoire tampon définie par l'utilisateur et gérée ne peut pas être redimensionnée. Etant donné que les mémoires tampon de messages sont gérées par le système, cela indique une incohérence interne.
MQRC_INSUFFISENT_BUFFER (6113)	L'espace de mémoire tampon disponible après le pointeur de données est insuffisant pour répondre à la demande. Cela peut être dû au fait que la mémoire tampon ne peut pas être redimensionnée.

Tableau 158. Codes raison et signification (suite)

Code raison	Explication
MQRC_INSUFFISENT_DATA (6114)	Les données après le pointeur de données sont insuffisantes pour répondre à la demande de lecture. Réduisez la taille de la mémoire tampon et lisez à nouveau les données.
MQRC_DATA_TRUNCATED (6115)	Les données ont été tronquées lors de la copie d'une mémoire tampon à une autre. Cela peut être dû au fait que la mémoire tampon cible ne peut pas être redimensionnée, ou qu'il y a un problème d'adressage de l'une ou l'autre mémoire tampon, ou qu'une mémoire tampon est redimensionnée avec un remplacement plus petit.
MQRC_ZERO_LENGTH (6116)	Une longueur nulle a été fournie lorsqu'une longueur positive est requise ou implicite.
MQRC_NEGATIVE_LENGTH (6117)	Une longueur négative a été fournie lorsqu'une longueur nulle ou positive est requise.
MQRC_NEGATIVE_OFFSET (6118)	Un décalage négatif a été fourni lorsqu'un décalage nul ou positif est requis.
MQRC_INCONSISTENT_FORMAT (6119)	Le format de l'élément de message suivant est incompatible avec le type de variable dans lequel l'élément est lu.
MQRC_INCONSISTENT_OBJECT_STATE (6120)	Il existe une incohérence entre cet objet, qui est ouvert, et l'objet MQQueueManager référencé, qui n'est pas connecté.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	La référence de contexte MQPutMessageOptions ne fait pas référence à un objet MQQueue valide. L'objet a été détruit.
MQRC_CONTEXT_OPEN_ERROR (6122)	La référence de contexte MQPutMessageOptions fait référence à un objet MQQueue qui n'a pas pu être ouvert pour établir un contexte. Cela peut être dû au fait que l'objet MQQueue comporte des options d'ouverture inappropriées. Examinez le code raison de l'objet référencé pour en déterminer la cause.
MQRC_STRUC_LENGTH_ERROR (6123)	La longueur d'une structure de données interne est incohérente avec son contenu. Pour un MQRMH, la longueur est insuffisante pour contenir les zones fixes et toutes les données de décalage.
MQRC_NOT_CONNECTED (6124)	Une méthode a échoué car une connexion requise à un gestionnaire de files d'attente n'était pas disponible et une connexion ne peut pas être établie implicitement.
MQRC_NOT_OPEN (6125)	Une méthode a échoué car un objet WebSphere MQ n'était pas ouvert et l'ouverture ne peut pas être effectuée implicitement.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	MQDistributionList n'a pas pu s'ouvrir car il n'y a pas d'objets d'élément MQDistributionList dans la liste de distribution. Action corrective: Ajoutez au moins un objet d'élément MQDistributionListà la liste de distribution.
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	Une méthode a échoué car l'objet est ouvert et les options d'ouverture sont incohérentes avec l'opération requise. Action corrective: Ouvrez l'objet avec les options d'ouverture appropriées et réessayez.
MQRC_WRONG_VERSION (6128)	Une méthode a échoué car un numéro de version spécifié ou rencontré est incorrect ou n'est pas pris en charge.

Outil de niveau de code

L'équipe de maintenance IBM peut vous demander quel niveau de code vous avez installé.

Pour ce faire, exécutez le programme utilitaire'MQAXLEV'.

A partir de l'invite de commande, accédez au répertoire contenant MQAX200.dll ou ajoutez le chemin d'accès complet et entrez:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

où MQAXLEV.OUT est le nom du fichier de sortie.

Si vous ne spécifiez pas de fichier de sortie, les détails s'affichent à l'écran.

Un exemple de fichier de sortie de l'outil de niveau de code est détaillé dans l'exemple suivant:

Exemple de fichier de sortie de l'outil de niveau de code

```
5639-B43 (C) Copyright IBM Corp. 1996, 2024. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 ***** lib/mqole/mqole.cpp, mqole, p000, p000 L981119 1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212  1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216             1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212               1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212            1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212           1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212           1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212           1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219             1.11 99/02/18 12:12:59
```

Interface ActiveX vers MQAI

Pour une brève présentation des interfaces COM et de leur utilisation dans MQAI, voir «Utilisation de Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX)», à la page 1065.

Le MQAI permet aux applications de générer et d'envoyer des commandes PCF (Programmable Command Format) sans obtenir et formater directement les tampons de longueur variable requis pour PCF. Pour plus d'informations sur MQAI, voir Introduction à WebSphere MQ Administration Interface (MQAI). La classe MQAI ActiveX MQBag encapsule les sacs de données pris en charge par MQAI d'une manière qui peut être utilisée dans n'importe quel langage prenant en charge la création d'objets COM ; par exemple, Visual Basic, C + +, Java et d'autres clients de scriptage ActiveX .

L'interface ActiveX de MQAI est destinée à être utilisée avec les classes MQAX qui fournissent une interface COM à l'interface MQI. Pour plus d'informations sur les classes MQAX, voir «Conception d'applications MQAX qui accèdent à des applications nonActiveX», à la page 1066.

L'interface ActiveX fournit une classe unique appelée MQBag. Cette classe est utilisée pour créer des sacs de données MQAI et ses propriétés et méthodes sont utilisées pour créer et utiliser des éléments de données dans chaque sac. La méthode MQBag Execute envoie les données de sac à un gestionnaire de files d'attente WebSphere MQ en tant que message PCF et collecte les réponses.

Pour plus d'informations sur la classe MQBag, ses propriétés et ses méthodes, voir «Classe MQBag», à la page 1143.

Le message PCF est envoyé à l'objet gestionnaire de files d'attente spécifié, en utilisant éventuellement les files d'attente de demandes et de réponses spécifiées. Les réponses sont renvoyées dans un nouvel objet MQBag. L'ensemble complet de commandes et de réponses est décrit dans la rubrique Définitions des formats de commande programmables. Les commandes peuvent être envoyées à n'importe quel gestionnaire de files d'attente du réseau WebSphere MQ en sélectionnant les files d'attente de demandes et de réponses appropriées.

Classe MQBag

La classe, MQBag, est utilisée pour créer des objets MQBag selon les besoins. Une fois instanciée, la classe MQBag renvoie une nouvelle référence d'objet MQBag.

Créez un objet MQBag dans Visual Basic comme suit:

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

Propriété MQBag

Les propriétés des objets MQBag sont expliquées dans la liste suivante:

- «Propriété d'élément», à la page 1144.
- «Count, propriété», à la page 1145.
- «Propriété Options», à la page 1146.

Méthodes MQBag

Les méthodes des objets MQBag sont expliquées dans la liste suivante:

- «méthode Add», à la page 1147.
- «Méthode AddInquiry», à la page 1147.
- «Méthode Clear», à la page 1147.
- «execute, méthode», à la page 1148.
- «Méthode FromMessage», à la page 1148.
- «Méthode ItemType», à la page 1149.
- «méthode Remove», à la page 1149.
- «Méthode de sélecteur», à la page 1150.
- «Méthode ToMessage», à la page 1151.
- «Méthode truncate», à la page 1151.

Traitement des erreurs

Si une erreur est détectée lors d'une opération sur un objet MQBag, y compris les erreurs renvoyées au sac par un objet MQAX ou MQAI sous-jacent, une exception d'erreur est générée. La classe MQBag prend en charge l'interface COM ISupportErrorInfo, de sorte que les informations suivantes sont disponibles pour votre routine de traitement des erreurs:

- Numéro d'erreur: composé du code anomalie WebSphere MQ de l'erreur détectée et d'un code de fonction COM. Le champ de l'installation, en tant que norme pour COM, indique le domaine de responsabilité de l'erreur. Pour les erreurs détectées par WebSphere MQ , il s'agit toujours de la valeur de la propriété SIFILITY_ITF.
- Source de l'erreur: identifie le type et la version de l'objet qui a détecté l'erreur. Pour les erreurs détectées lors des opérations MQBag, la source de l'erreur est toujours MQBag.MQBag1.
- Description de l'erreur: chaîne indiquant le nom symbolique du code anomalie WebSphere MQ .

La façon dont vous accédez aux informations d'erreur dépend de votre langage de script ; par exemple, dans Visual Basic, les informations sont renvoyées dans l'objet Err et le code anomalie WebSphere MQ est obtenu en soustrayant la constante vbObjectError from Err.Number.

ReasonCode = Err.Number -Erreur vbObject

Si le message MQBag Execute envoie un message PCF et qu'une réponse est reçue, l'opération est considérée comme réussie, même si la commande envoyée a peut-être échoué. Dans ce cas, le sac de réponse lui-même contient les codes raison d'achèvement et d'erreur décrits dans la rubrique Définitions des formats de commande programmables .

Propriété d'élément

Objet

La propriété Article représente un article dans un sac. Il est utilisé pour définir ou interroger la valeur d'un élément. L'utilisation de cette propriété correspond aux appels MQAI suivants:

- "mqSetChaîne"

- "mqSetentier"
- "mqInquireentier"
- "mqInquireChaîne"
- "BagmqInquire"

dans le Référence pour les formats de commande programmables.

Format

Elément (Sélecteur, ItemIndex, Valeur)

Paramètres

Selector **(VARIANT)-entrée**
Sélecteur de l'élément à définir ou demandé.

Lors de l'interrogation d'un élément, MQSEL_ANY_USER_SELECTOR est la valeur par défaut. Lors de la définition d'un élément, MQIA_LIST ou MQCA_LIST est la valeur par défaut.

Si le `Selector` n'est pas de type long, le message MQRC_SELECTOR_TYPE_ERROR est renvoyé.

Ce paramètre est facultatif.

ItemIndex **(LONG)-entrée**
Cette valeur identifie l'occurrence de l'élément du sélecteur spécifié qui doit être défini ou demandé. MQIND_NONE est la valeur par défaut.

Ce paramètre est facultatif.

Value **(VARIANT)-entrée/sortie**
Valeur renvoyée ou valeur à définir. Lors de l'interrogation d'un élément, la valeur de retour peut être de type long, chaîne ou MQBag. Toutefois, lors de la définition d'un élément, la valeur doit être de type long ou chaîne ; si ce n'est pas le cas, MQRC_ITEM_VALEUR_ERROR est renvoyé.

Appel Visual Basic Language

Lors de la recherche d'une valeur d'un article dans un sac:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

Pour les références MQBag:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

Pour définir la valeur d'un article dans un sac:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Count, propriété

Objet

La propriété Nombre représente le nombre d'éléments de données dans un sac. Cette propriété correspond à l'appel MQAI, "mqCountItems", dans Référence pour les formats de commande programmables.

Format

Nombre (*Selector, Value*)

Paramètres

***Selector* (VARIANT)-entrée**
Sélecteur des éléments de données à inclure dans le comptage.

MQSEL_ALL_USER_SELECTORS est la valeur par défaut.

Si `Selector` n'est pas de type long, MQRC_SELECTOR_TYPE_ERROR est renvoyé.

***Value* (LONG)-sortie**
Nombre d'éléments du sac inclus par *Selector*.

Appel Visual Basic Language

Pour renvoyer le nombre d'articles dans un sac:

```
ItemCount = mqbag.Count([Selector])
```

Propriété Options

Objet

La propriété Options définit les options d'utilisation d'un sac. Cette propriété correspond au paramètre `Options` de l'appel MQAI, "mqCreateBag", dans le Référence pour les formats de commande programmables.

Format

Options (*Options*)

Paramètres

***Options* (LONG)-entrée/sortie**
Les options du sac.

Remarque : Les options de sac doivent être définies ***avant que les éléments de données*** soient ajoutés ou définis dans le sac. Si les options sont modifiées lorsque le sac n'est pas vide, MQRC_OPTIONS_ERROR est renvoyé. Cela s'applique même si le sac est ensuite nettoyé.

Appel Visual Basic Language

Lorsque vous vous interrogez sur les options d'un article dans un sac:

```
Options = mqbag.Options
```

Pour définir une option d'un article dans un sac:

```
mqbag.Options = Options
```

Méthodes MQBag

Les méthodes des objets MQBag sont expliquées dans les pages suivantes.

méthode Add

Objet

La méthode Add ajoute un élément de données à un sac. Cette méthode correspond aux appels MQAI, "mqAddInteger" et "mqAddString", dans Référence pour les formats de commande programmables.

Format

Ajouter (*Value, Selector*)

Paramètres

***Value* (VARIANT)-entrée**
Valeur de type entier ou chaîne de l'élément de données.

***Selector* (VARIANT)-entrée**
Sélecteur identifiant l'élément à ajouter.

Selon le type de `Value`, MQIA_LIST ou MQCA_LIST est la valeur par défaut. Si le paramètre `Selector` n'est pas de type long, MQRC_SELECTOR_TYPE_ERROR est renvoyé.

Appel Visual Basic Language

Pour ajouter un article à un sac:

```
mqbag.Add(Value,[Selector])
```

Méthode AddInquiry

Objet

La méthode AddInquiry ajoute un sélecteur spécifiant l'attribut à renvoyer lorsqu'un sac d'administration est envoyé pour exécuter une commande INQUIRE. Cette méthode correspond à l'appel MQAI, "mqAddInquiry", dans Référence pour les formats de commande programmables.

Format

AddInquiry *(Inquiry*)

Paramètres

***Inquiry* (LONG)-entrée**
Sélecteur de l'attribut WebSphere MQ à renvoyer par la commande d'administration INQUIRE.

Appel Visual Basic Language

Pour utiliser la méthode AddInquiry :

```
mqbag.AddInquiry(Inquiry)
```

Méthode Clear

Objet

La méthode Clear supprime tous les éléments de données d'un sac. Cette méthode correspond à l'appel MQAI, "mqClearBag", dans Référence pour les formats de commande programmables.

Format

Effacement

Appel Visual Basic Language

Pour supprimer toutes les données d'un sac:

```
mqbag.Clear
```

execute, méthode

Objet

La méthode Execute envoie un message de commande d'administration au serveur de commandes et attend les éventuels messages de réponse. Cette méthode correspond à l'appel MQAI, "mqExecute", dans Référence pour les formats de commande programmables.

Format

Exécutez *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*)

Paramètres

QueueManager **(MQQueueManager)-entrée**
Gestionnaire de files d'attente auquel l'application est connectée.

Command **(LONG)-entrée**
Commande à exécuter.

OptionsBag **(MQBag)-entrée**
Sac contenant les options qui affectent le traitement de l'appel.

RequestQ **(MQQueue)-entrée**
File d'attente dans laquelle le message de commande d'administration sera placé.

ReplyQ **(MQQueue)-entrée**
File d'attente dans laquelle sont reçus les messages de réponse.

ReplyBag **(MQBag)-sortie**
Référence de sac contenant les données des messages de réponse.

Appel Visual Basic Language

Pour envoyer un message de commande d'administration et attendre les messages de réponse:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

Méthode FromMessage

Objet

La méthode FromMessage charge les données d'un message dans un sac. Cette méthode correspond à l'appel MQAI, "mqBufferToBag", dans Référence pour les formats de commande programmables.

Format

FromMessage *(Message, OptionsBag*)

Paramètres

***Message* (MQMessage)-entrée**
Message contenant les données à convertir.

***OptionsBag* (MQBag)-entrée**
Options permettant de contrôler le traitement de l'appel.

Appel Visual Basic Language

Pour charger les données d'un message dans un sac:

```
mqbag.FromMessage(Message,[OptionsBag])
```

Méthode ItemType

Objet

La méthode ItemType renvoie le type de la valeur d'un élément spécifié dans un sac. Cette méthode correspond à l'appel MQAI, "mqInquireItemInfo", dans le Référence pour les formats de commande programmables.

Format

ItemType *(Selector, ItemIndex, ItemType*)

Paramètres

***Selector* (VARIANT)-entrée**
Sélecteur identifiant l'élément à questionner.

MQSEL_ANY_USER_SELECTOR est la valeur par défaut. Si le paramètre `Selector` n'est pas de type long, MQRC_SELECTOR_TYPE_ERROR est renvoyé.

***ItemIndex* (LONG)-entrée**
Index des éléments à questionner.

MQIND_NONE est la valeur par défaut.

***ItemType* (LONG)-sortie**
Type de données de l'élément spécifié.

Remarque : Le paramètre `Selector` et/ou le paramètre `ItemIndex` doivent être spécifiés. Si aucun paramètre n'est présent, MQRC_PARAMETER_MISSING génère des résultats.

Appel Visual Basic Language

Pour renvoyer le type d'une valeur:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

méthode Remove

Objet

La méthode Remove supprime un élément d'un sac. Cette méthode correspond à l'appel MQAI, "mqDeleteItem", dans Référence pour les formats de commande programmables.

Format

Supprimez *(Selector, ItemIndex)*

Paramètres

***Selector* (VARIANT)-entrée**
Sélecteur identifiant l'élément à supprimer.

MQSEL_ANY_USER_SELECTOR est la valeur par défaut. Si le paramètre `Selector` n'est pas de type long, MQRC_SELECTOR_TYPE_ERROR est renvoyé.

***ItemIndex* (LONG)-entrée**
Index de l'élément à supprimer.

MQIND_NONE est la valeur par défaut.

Remarque : Le paramètre `Selector` et/ou le paramètre `ItemIndex` doivent être spécifiés. Si aucun paramètre n'est présent, MQRC_PARAMETER_MISSING génère des résultats.

Appel Visual Basic Language

Pour supprimer un article d'un sac:

```
mqbag.Remove([Selector],[ItemIndex])
```

Méthode de sélecteur

Objet

La méthode Selector renvoie le sélecteur d'un élément spécifié dans un sac. Cette méthode correspond à l'appel MQAI, "mqInquireItemInfo", dans le fichier Référence pour les formats de commande programmables.

Format

Sélecteur *(Selector, ItemIndex, OutSelector)*

Paramètres

***Selector* (VARIANT)-entrée**
Sélecteur identifiant l'élément à questionner.

MQSEL_ANY_USER_SELECTOR est la valeur par défaut. Si le paramètre `Selector` n'est pas de type long, MQRC_SELECTOR_TYPE_ERROR est renvoyé.

***ItemIndex* (LONG)-entrée**
Index de l'élément à questionner.

MQIND_NONE est la valeur par défaut.

***OutSelector* (VARIANT)-sortie**
Sélecteur de l'élément spécifié.

Remarque : Le paramètre `Selector` et/ou le paramètre `ItemIndex` doivent être spécifiés. Si aucun paramètre n'est présent, MQRC_PARAMETER_MISSING génère des résultats.

Appel Visual Basic Language

Pour renvoyer le sélecteur d'un élément:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

Méthode ToMessage

Objet

La méthode ToMessage renvoie une référence à un objet MQMessage. La référence contient des données provenant d'un sac. Cette méthode correspond à l'appel MQAI, "mqBagToBuffer", dans le Référence pour les formats de commande programmables.

Format

ToMessage *(OptionsBag, Message)*

Paramètres

OptionsBag **(MQBag)-entrée**
Sac contenant des options qui contrôlent le traitement de la méthode.

Message **(MQMessage)-sortie**
Une référence d'objet MQMessage contenant des données du sac.

Appel Visual Basic Language

Pour utiliser la méthode ToMessage :

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Méthode truncate

Objet

La méthode Troncate réduit le nombre d'éléments utilisateur dans un sac. Cette méthode correspond à l'appel MQAI, "mqTruncateBag", dans Référence pour les formats de commande programmables.

Format

Tronquer ***(ItemCount)***

Paramètres

ItemCount **(LONG)-entrée**
Nombre d'éléments utilisateur à conserver dans le sac après la troncature.

Appel Visual Basic Language

Pour réduire le nombre d'éléments utilisateur dans un sac:

```
mqbag.Truncate(ItemCount)
```

A propos des exemples de démarrage WebSphere MQ Automation Classes for ActiveX

Cette annexe décrit les exemples WebSphere MQ Automation Classes for ActiveX Starter et explique comment les utiliser.

WebSphere MQ for Windows fournit les exemples de programmes Visual Basic suivants:

- MQAXTRIV.VBP
- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

Ces exemples s'exécutent sur Visual Basic 4 ou Visual Basic 5. Ils se trouvent dans le répertoire ... \tools\mqax\samples\vb.

Dans le même répertoire, vous trouverez également des exemples pour Microsoft Excel et html. Il s'agit des fonctions suivantes :

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

Remarque : Si vous utilisez Visual Basic 5, vous **devez** sélectionner et installer le composant Visual Basic grid32.ocx.

Ce qui est démontré dans les échantillons

Les exemples montrent comment utiliser WebSphere MQ Automation Classes for ActiveX pour:

- Connexion à un gestionnaire de files d'attente
- Accès à une file d'attente
- Insertion d'un message dans une file d'attente
- Extraction d'un message d'une file d'attente

La partie centrale de l'exemple Visual Basic est présentée dans les pages suivantes.

«Préparation de l'exécution des exemples», à la page 1153et

«Traitement des erreurs dans les exemples», à la page 1153

Exécution des exemples de module de démarrage ActiveX

Avant d'exécuter les exemples WebSphere MQ Automation Classes for ActiveX Starter, vérifiez qu'un gestionnaire de files d'attente par défaut est en cours d'exécution et que vous avez créé les définitions de file d'attente requises. Pour plus de détails sur la création et l'exécution d'un gestionnaire de files d'attente et la création d'une file d'attente, voir Administration. L'exemple utilise la file d'attente SYSTEM.DEFAULT.LOCAL.QUEUE qui doit être définie sur tout serveur WebSphere MQ normalement configuré.

Les différentes façons d'utiliser les sacs de données sont présentées dans la liste suivante:

- Connexion à un gestionnaire de files d'attente
- Accès à une file d'attente
- Insertion d'un message dans une file d'attente
- Extraction d'un message d'une file d'attente

Pour plus d'informations sur les exemples de module de démarrage MQAX pour Microsoft Basic Version 4 ou ultérieure, voir «Exécution de l'exemple MQAXTRIV», à la page 1153

Pour plus d'informations sur un exemple qui vous permet de parcourir les propriétés et les méthodes des gestionnaires de files d'attente et des objets de file d'attente, voir «Démarrage de l'exemple MQAXCLSS», à la page 1155

Pour plus d'informations sur l'exemple MQAXDLST, «Exemple MQAXDLST», à la page 1155

Pour plus d'informations sur l'exécution de l'exemple de module de démarrage MQAX pour Microsoft Excel 95 ou version ultérieure, voir MQAXTRIV.XLS, voir «Exécution de MQAXTRIV.XLS», à la page 1155.

Pour plus d'informations sur l'exécution de la démonstration Bank avec MQAX.XLS, voir «Exécution de la démonstration Bank avec MQAX.XLS)», à la page 1155

Pour plus d'informations sur l'exemple de module de démarrage utilisant un navigateur WWW compatible ActiveX , voir «Exemple de module de démarrage utilisant un navigateur WWW compatible ActiveX», à la page 1156

Préparation de l'exécution des exemples

Pour exécuter l'un des exemples, vous avez besoin de l'un des éléments suivants en fonction des exemples que vous avez l'intention d'exécuter.

- Microsoft Visual Basic version 4 (ou ultérieure)
- Microsoft Excel 95 (ou version ultérieure)
- un navigateur Web

Vous avez également besoin de:

- Gestionnaire de files d'attente WebSphere MQ en cours d'exécution.
- Une file d'attente WebSphere MQ est déjà définie.

Traitement des erreurs dans les exemples

La plupart des exemples fournis dans le package WebSphere MQ Automation Classes for ActiveX présentent peu ou pas de traitement d'erreurs. Pour plus d'informations sur le traitement des erreurs, voir «Traitement des erreurs», à la page 1070.

Exécution de l'exemple MQAXTRIV

1. Démarrez le gestionnaire de files d'attente.
2. Dans Windows Explorer ou File Manager, sélectionnez l'icône de l'exemple, MQAXTRIV.VBP (fichier de projet Visual Basic) et ouvrez le fichier.

   Le programme Visual Basic démarre et ouvre le fichier MQAXTRIV.VBP.
3. Dans Visual Basic, appuyez sur la touche de fonction 5 (F5) pour exécuter l'exemple.
4. Cliquez n'importe où dans le formulaire de la fenêtre, "MQAX trivial tester".

Si tout fonctionne correctement, l'arrière-plan de la fenêtre doit passer au vert. S'il y a un problème avec votre configuration, l'arrière-plan de la fenêtre doit passer en rouge et les informations d'erreur s'affichent.

La figure suivante montre la partie centrale de l'exemple Visual Basic.

```
Option Explicit

Private Sub Form_Click()

'******************************************************************************
'* This simple example illustrates how to put and get a WebSphere MQ message to
'* and from a WebSphere MQ message queue. The data from the message returned by the
'*get is read and compared with that from the original message.
'******************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* get message option
```

```
Dim GetOptions As MQGetMessageOptions   '* put message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*****************************************************************************
'* Handle errors
'*****************************************************************************
On Error GoTo HandleError

'*****************************************************************************
'* Initialize the current position for the form
'*****************************************************************************
CurrentX = 0
CurrentY = 0

'*****************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*****************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                      MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*****************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*****************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*****************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*****************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*****************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*****************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127)  '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*****************************************************************************
'* Handle errors
'*****************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
```

```
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "WebSphere MQ Completion Code = " & MQSess.CompletionCode
    Print "WebSphere MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
Else
    Print "Visual Basic error:  " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

Démarrage de l'exemple MQAXCLSS

Cet exemple permet de parcourir les propriétés et les méthodes des gestionnaires de files d'attente et des objets de file d'attente.

1. Démarrez le gestionnaire de files d'attente.
2. Ouvrez le fichier MQAXCLSS.VBP, en cliquant deux fois sur l'icône du document dans Windows Explorer ou en cliquant sur Fichier-Ouvrir dans le menu Fichier de Visual Basic.
3. Démarrez l'exemple.
4. Entrez le gestionnaire de files d'attente et les noms de file d'attente appropriés, puis cliquez sur les boutons correspondants.

Exemple MQAXDLST

L'exemple Visual Basic MQAXDLST illustre l'utilisation d'une liste de distribution pour envoyer le même message à deux files d'attente avec une seule insertion. Pour exécuter l'exemple, procédez de la même manière que pour l'exemple MQAXCLSS.

Exemple MQAX Starter pour Microsoft Excel 95 ou version ultérieure

Cette section explique comment exécuter l'exemple de module de démarrage MQAX pour Microsoft Excel 95 ou version ultérieure, MQAXTRIV.XLS.

Exécution de MQAXTRIV.XLS

1. Démarrez le gestionnaire de files d'attente.
2. Dans Explorer ou File Manager, sélectionnez l'icône de l'exemple MQAX MQAXTRIV.XLS.
3. Cliquez sur le bouton dans la feuille de calcul.
4. L'écran est mis à jour avec un message de réussite (ou d'échec).

Exécution de la démonstration Bank avec MQAX.XLS)

Procédez comme suit pour exécuter la démonstration de Bank.

1. Démarrez le gestionnaire de files d'attente.
2. Exécutez le fichier de commandes IBM WebSphere MQ MQSC, BANK.TST. Cela permet de configurer les définitions de file d'attente IBM WebSphere MQ nécessaires.

   Pour savoir comment utiliser un fichier de commandes MQSC, voir Commandes de script (MQSC).
3. Exécutez MQAXBSRV.VBP. Cet exemple de programme est le serveur qui simule une application de back end et il doit être exécuté avec Microsoft Excel.
4. Exécutez MQAX.XLS. Cet exemple est la démonstration IBM WebSphere MQ du client.

5. Sélectionnez un client dans la liste.
6. Cliquez sur **Soumettre**.

Après une courte pause (environ 3 secondes), les zones sont renseignées avec des valeurs et un graphique à barres s'affiche.

Exemple de module de démarrage utilisant un navigateur WWW compatible ActiveX

Remarque : Pour exécuter cet exemple, vous devez exécuter un navigateur Web compatible ActiveX . Microsoft Internet Explorer (mais pas Netscape Navigator) est un navigateur Web compatible.

Exécution de l'exemple HTML

Cet exemple explique comment appeler MQAX à partir de VBScript et de JavaScript.

1. Démarrez le gestionnaire de files d'attente.
2. Ouvrez le fichier "MQAXTRIV.HTM", dans votre navigateur Web compatible ActiveX .

   Vous pouvez effectuer cette opération en cliquant deux fois sur l'icône de fichier dans l'explorateur Windows ou en sélectionnant Fichier-Ouvrir dans le menu Fichier de votre navigateur Web compatible ActiveX .
3. Suivez les instructions à l'écran.

Remarques

:NONE.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service IBM puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. 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 EMEA Director of Licensing
IBM Corporation
Tour Descartes
Armonk, NY 10504-1785
U.S.A.

Pour toute demande d'informations relatives au jeu de caractères codé sur deux octets, contactez le service de propriété intellectuelle IBM ou envoyez vos questions par courrier à l'adresse suivante :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japon

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 logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. 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 de IBM WebSphere 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 d'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 (http://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: