MQSeries

Utilisation de Java

SC11-1511-02

MQSeries

Utilisation de Java

SC11-1511-02

Important

Avant d'utiliser le présent manuel et le produit associé, prenez connaissance des informations générales figurant à la section «Annexe F. Remarques» à la page 377.

Septième édition - janvier 2001

REF US: SC34-5456-06

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- http://www.fr.ibm.com (serveur IBM en France)
- http://www.can.ibm.com (serveur IBM au Canada)
- http://www.ibm.com (serveur IBM aux Etats-Unis)

Compagnie IBM France
Direction Qualité
Tour Descartes
92066 Paris-La Défense Cedex 50

Table des matières

Figures

Tableaux

Préface

Le présent manuel décrit les éléments suivants :

- Les classes MQSeries pour Java, qui permettent d'accéder aux systèmes MQSeries.
- Les classes MQSeries pour Java Message Service, qui permettent d'accéder à Java Message Service (JMS) et aux applications MQSeries.

Remarque : Cette documentation est disponible en ligne uniquement (aux formats PDF et HTML) et fournie avec le produit. Elle est également accessible sur le site Web MQSeries à l'adresse suivante :

`http://www.ibm.com/software/mqseries/`

Elle **n'est pas** disponible au format papier.

Abréviations utilisées dans ce manuel

Les abréviations suivantes sont utilisées dans ce manuel :

MQ Java	classes MQSeries pour Java et classes MQSeries pour Java Message Service ensemble
MQ base Java	classes MQSeries pour Java
MQ JMS	classes MQSeries pour Java Message Service

A qui s'adresse ce manuel

Ces informations sont destinées aux programmeurs qui connaissent déjà l'interface de programmation d'application procédurale MQSeries décrite dans le manuel *MQSeries - Guide de programmation d'applications*. Elles illustrent comment exploiter ces connaissances et utiliser de façon efficace l'interface de programmation MQ Java.

Connaissances préalables

Pour tirer parti de ce manuel, vous devez :

- connaître le langage de programmation Java ;
- comprendre le rôle de l'interface MQI (Message Queue Interface), décrite dans le chapitre correspondant du manuel *MQSeries - Guide de programmation d'applications* et dans le chapitre sur les descriptions d'appel du manuel *MQSeries - Référence de programmation d'applications* ;
- avoir l'expérience des programmes MQSeries en général, ou avoir lu d'autres publications consacrées à MQSeries.

Les utilisateurs qui envisagent d'exécuter MQ base Java avec CICS Transaction Server pour OS/390 doivent également connaître :

- les concepts CICS (Customer Information Control System) ;
- l'utilisation de l'interface de programmation d'application CICS Java ;
- l'exécution des programmes Java au sein de CICS.

Les utilisateurs qui envisagent d'exécuter VisualAge pour Java pour développer des applications HPJ (High Performance Java) de services système UNIX OS/390 doivent savoir utiliser Enterprise Toolkit pour OS/390 (fourni avec VisualAge for Java Enterprise Edition pour OS/390, version 2).

Comment utiliser ce manuel

La première partie porte sur l'utilisation de MQ base Java et MQ JMS, la deuxième partie propose des informations destinées aux programmeurs souhaitant utiliser MQ base Java et la troisième partie fournit des informations pour les programmeurs souhaitant utiliser MQ JMS.

Lisez d'abord les chapitres de la partie 1, qui présentent MQ base Java et MQ JMS. Ensuite, reportez-vous aux conseils de programmation des parties 2 ou 3 pour comprendre comment utiliser les classes MQSeries pour Java, et envoyer et recevoir des messages MQSeries dans l'environnement souhaité.

Un glossaire et une bibliographie se trouvent à la fin de ce manuel.

Résumé des modifications

Cette section décrit les modifications contenues dans la présente édition du manuel *MQSeries - Utilisation de Java*. Toute modification apportée depuis l'édition précédente est signalée par des barres verticales dans la marge de gauche.

Modifications apportées à cette édition (SC11-1511-02)

La présente édition comporte des mises à jour relatives à la nouvelle fonction intégrée à MQ Java 5.2. Elle contient les nouveautés suivantes :

- Mises à jour des procédures d'installation. Voir «Chapitre 2. Procédures d'installation» à la page 9.
- Prise en charge du regroupement de connexions, qui permet d'améliorer les performances des applications et des logiciels intermédiaires qui utilisent des connexions multiples vers les gestionnaires de files d'attente MQSeries. Reportez-vous aux sections suivantes :
  - «Définition d'un pool de connexion» à la page 66
  - «MQEnvironment» à la page 93
  - «MQPoolServices» à la page 129
  - «MQPoolServicesEvent» à la page 130
  - «MQPoolToken» à la page 132
  - «MQQueueManager» à la page 147
  - «MQSimpleConnectionManager» à la page 157
  - «MQConnectionManager» à la page 161
  - «MQPoolServicesEventListener» à la page 160
  - «ManagedConnection» à la page 168
  - «ManagedConnectionFactory» à la page 171
  - «ManagedConnectionMetaData» à la page 173
- Nouvelles options de configuration de file d'attente de souscripteur offrant la possibilité conjointe de files d'attente multiples et de files d'attente partagées pour les applications de publication/souscription. Reportez-vous aux sections suivantes :
  - «Propriétés» à la page 40
  - «Configuration de la file d'attente de base du souscripteur» à la page 193
  - «Topic» à la page 328
  - «TopicConnectionFactory» à la page 333
- Nouvel utilitaire de nettoyage d'objets subscriber qui permet d'éviter les incidents résultant de la fermeture inappropriée d'un objet subscriber. Voir «Utilitaire de nettoyage d'objets subscriber» à la page 196.
- Prise en charge des fonctions de serveur d'applications ASF (Application Server Facilities), de traitement concurrent des messages. Reportez-vous aux sections suivantes :
  - «Chapitre 13. Fonctions de serveur d'applications MQ JMS» à la page 217
  - «ConnectionConsumer» à la page 253
  - «QueueConnection» à la page 299
  - «Session» à la page 312

Modifications apportées à la sixième édition (SC11-1511-01)

Prise en charge de Linux.

Modifications apportées à la cinquième édition (SC11-1511-00)

Prise en charge de WebSphere et MQSeries Integrator V2

MQ base Java, version 5.1.2, est désormais disponible en tant qu'extension produit et permet :

- La connexion à MQSeries Integrator pour Windows NT, version 2.0, pour offrir un support de publication/souscription. Pour plus de précisions, voir «Annexe D. Connexion à MQSeries Integrator V2» à la page 369.
- L'utilisation du fournisseur de services CosNaming JNDI de WebSphere. Pour plus de détails, reportez-vous à la section «Configuration» à la page 34.

Partie 1. Présentation aux utilisateurs

Chapitre 1. Initiation

Le présent chapitre présente les classes MQSeries pour Java et les classes MQSeries pour Java Message Service, ainsi que leur utilisation.

Que sont les classes MQSeries pour Java ?

Les classes MQSeries pour Java (MQ base Java) permettent à un programme écrit en langage de programmation Java de :

- se connecter à MQSeries en tant que client MQSeries ;
- se connecter directement à un serveur MQSeries.

Elles permettent aux applets, applications et servlets Java d'émettre des appels et des requêtes vers MQSeries avec accès à des applications grand système ou existantes, en règle générale via Internet. La présence de code MQSeries sur l'ordinateur client n'est pas nécessaire. Avec MQ base Java, l'utilisateur d'un terminal Internet participe réellement aux transactions ; il n'est plus un simple récepteur et émetteur d'informations.

Que sont les classes MQSeries pour Java Message Service ?

Les classes MQSeries pour Java Message Service (MQ JMS) regroupent des classes Java servant à implémenter des interfaces JMS Java (Java Message Service) de Sun pour permettre aux programmes JMS d'accéder aux systèmes MQSeries. Les modèles JMS point à point et de publication/souscription sont tous deux pris en charge.

L'utilisation de MQ JMS comme interface de programmation lors de l'écriture d'applications MQSeries offre un certain nombre d'avantages. Certains d'entre eux sont propres à JMS qui constitue une norme de systèmes ouverts aux multiples implémentations. D'autres sont liés à des fonctions supplémentaires intégrées à MQ JMS, mais pas à MQ base Java.

Les avantages des normes de systèmes ouverts incluent, entre autres :

- la protection des investissements, à la fois en matière d'aptitudes et de code applicatif
- la disponibilité des personnes qualifiées en matière de programmation d'applications JMS
- la possibilité de relier différentes implémentations JMS en fonction des besoins

Pour plus de détails sur les avantages de l'API JMS, accédez au site Web Sun à l'adresse : `http://java.sun.com`.

La fonction supplémentaire fournie avec MQ base Java offre, entre autres :

- la communication des messages en mode asynchrone
- des sélecteurs de message
- un support de messagerie de publication/souscription
- des classes de message structurées

A qui est destiné MQ Java ?

Si votre entreprise répond à l'une des descriptions suivantes, les classes MQSeries pour Java et les classes MQSeries pour Java Message Service présentent pour vous de nombreux avantages :

- Une entreprise moyenne ou grande mettant en place des solutions client-serveur sur intranet. En l'occurrence, la technologie Internet fournit un accès aisé et de faible coût au monde entier, tandis que la connectivité MQSeries apporte l'assurance d'une intégrité des données élevée avec livraison garantie en mode asynchrone.
- Une entreprise moyenne ou grande cherchant à se doter de moyens de communication fiables pour correspondre avec des partenaires. Là encore, la technologie Internet fournit un accès aisé et de faible coût au monde entier, tandis que la connectivité MQSeries apporte l'assurance d'une intégrité des données élevée avec livraison garantie en mode asynchrone.
- Une entreprise moyenne ou grande souhaitant offrir un accès public, par Internet, à quelques-unes de ses applications. Dans ce cas, Internet met ces applications à la disposition du monde entier, tandis que la connectivité MQSeries assure une grande intégrité des données par la mise en oeuvre du principe des files d'attente. Outre le coût réduit, l'entreprise peut bénéficier d'une amélioration du niveau de satisfaction de sa clientèle en offrant un accès 24 heures sur 24, des temps de réponse réduits et une plus grande précision dans les informations fournies.
- Un fournisseur d'accès Internet ou réseau à valeur ajoutée. Ce type d'entreprise peut en effet tirer parti du faible coût et de la simplicité de communication offerte par Internet en y ajoutant la fiabilité offerte par la connectivité MQSeries. Un fournisseur d'accès Internet exploitant MQSeries peut immédiatement accuser réception des données saisies au niveau d'un navigateur Web, garantir la livraison du message, et offrir à l'utilisateur du navigateur Web un moyen simple de contrôler l'état de sa requête.

MQSeries et les classes MQSeries pour Java Message Service offrent une excellente infrastructure d'accès aux applications d'entreprise et de développement d'applications Web complexes. Une demande de service provenant d'un navigateur Web peut être mise en file d'attente et traitée dès que possible, ce qui permet d'envoyer une réponse rapide à l'utilisateur quelle que soit la charge du système. Si l'on place cette file d'attente «près» de l'utilisateur en termes de réseau, la rapidité de cette réponse n'est pas remise en question par la charge du réseau. Par ailleurs, de par la nature transactionnelle de MQSeries, une simple requête provenant d'un navigateur Web peut être décomposée sans difficulté en une séquence de processus dorsaux.

Les classes MQSeries pour Java permettent également aux développeurs d'applications d'exploiter la puissance du langage de programmation Java pour créer des applets et des applications pouvant s'exécuter sur toute plateforme prenant en charge l'environnement d'exécution Java. Ces avantages cumulés réduisent sensiblement la durée de développement des applications MQSeries multi-plateformes et les améliorations ultérieures apportées aux applets sont automatiquement exploitées par les utilisateurs finals lorsque le code de l'applet est téléchargé.

Options de connexion

Des options programmables permettent à MQ Java de se connecter à MQSeries de deux manières :

- comme client MQSeries utilisant TCP/IP ;
- en connexion directe à MQSeries, lorsque le serveur MQSeries et le programme Java sont exécutés sur le même ordinateur (également appelé «mode liens» ou *bindings*).

MQ base Java, sous Windows NT, permet également une connexion à l'aide de VisiBroker pour Java. Le tableau 1 répertorie les modes de connexion utilisables sur chaque plateforme.

Tableau 1. Plateformes et modes de connexion

Plateforme serveur	**Mode de connexion**		
	Client		**Connexion directe**
	Standard	**VisiBroker**	
Windows NT	oui	oui	oui
Windows 2000	oui	non	oui
AIX	oui	non	oui
Sun OS (v4.1.4 ou précédente)	oui	non	non
Sun Solaris (v2.6, v2.8, V7 ou SunOS v5.6, v5.7)	oui	non	oui
OS/2	oui	non	oui
OS/400	oui	non	oui
HP-UX	oui	non	oui
AT&T GIS UNIX	oui	non	non
SINIX et DC/OSx	oui	non	non
OS/390	non	non	oui
Linux	oui	non	non

Remarques :

1. Le support de liaisons Java de HP-UX est disponible uniquement pour les systèmes HP-UXv11 exécutant la version provisoire POSIX de MQSeries. Vous devez également disposer du composant HP-UX Developer's Kit pour Java 1.1.7 (JDK), édition C.01.17.01 ou suivante.
2. Sous HP-UXv10.20, Linux, Windows 95 et Windows 98, seule la connectivité client TCP/IP est prise en charge.

Ces options sont décrites en détail ci-après.

Connexion client

Pour utiliser MQ Java en tant que client MQSeries, vous pouvez l'installer sur l'ordinateur serveur MQSeries (qui peut également héberger un serveur Web) ou sur un autre ordinateur. L'installation sur la même machine que le serveur Web présente l'avantage de vous permettre de télécharger et d'exécuter les applications client MQSeries sur des machines non dotées de MQ Java en local.

Connexions

Quel que soit l'emplacement d'installation choisi pour le client, ce dernier peut être exécuté dans trois modes différents :

A partir de tout navigateur Web compatible Java
Lorsque vous utilisez ce mode, l'accès aux gestionnaires de files d'attente MQSeries peut être limité par les restrictions de sécurité du navigateur utilisé.

A l'aide d'un afficheur d'applets
Pour utiliser ce mode, vous devez disposer du kit JDK (Java Developer's Kit) ou de l'environnement d'exécution Java (JRE) sur l'ordinateur client.

Comme programme Java autonome ou sur un serveur d'applications Web
Pour utiliser ce mode, vous devez disposer du kit JDK (Java Developer's Kit) ou de l'environnement d'exécution Java (JRE) sur l'ordinateur client.

Utilisation de VisiBroker pour Java

Sous Windows, la connexion via VisiBroker est proposée en complément de la connexion par le biais des protocoles client standard MQSeries. Cette option est prise en charge par VisiBroker pour Java conjointement avec Netscape Navigator. Elle nécessite VisiBroker pour Java et un serveur d'objets MQSeries sur l'ordinateur serveur MQSeries. Un serveur d'objets approprié est fourni avec MQ base Java.

Connexion directe

En connexion directe (mode liens), MQ Java utilise l'interface JNI pour appeler directement l'API du gestionnaire de files d'attente existant plutôt que de communiquer via un réseau. Dans ce mode, les applications MQSeries bénéficient de meilleures performances que lorsqu'elles utilisent des connexions de réseau. Cependant, contrairement au mode client, les applications écrites pour la connexion directe ne peuvent pas être téléchargées sous forme d'applets.

Pour pouvoir utiliser la connexion directe, MQ Java doit être installé sur le serveur MQSeries.

Logiciels requis

Les logiciels suivants sont requis pour pouvoir exécuter MQ base Java :

- MQSeries pour la plateforme serveur que vous souhaitez utiliser.
- Java Developer's Kit (JDK) pour la plateforme serveur.
- Java Developer's Kit, ou Java Runtime Environment (JRE), ou encore un navigateur Web compatible Java pour les plateformes client. (Voir «Connexion client» à la page 5.)

  Remarque : Pour exécuter des applets MQ base Java (par exemple le programme de vérification d'installation) depuis un navigateur Web, vous devez disposer d'un navigateur pouvant exécuter les applets Java 1.1.6. HotJava de Sun System, Netscape Navigator 4 et Microsoft Internet Explorer 4 sont parmi les navigateurs qui satisfont à cette exigence.
- VisiBroker pour Java (uniquement si vous utilisez Windows avec une connexion VisiBroker).
- Pour OS/390, OS/390 version 2.5 avec les services système UNIX.
- Pour OS/400, le kit de développement AS/400 pourJava, 5769-JV1, et l'interpréteur Qshell, OS/400 (5769-SS1) Option 30.

Logiciels requis

Les logiciels supplémentaires suivants sont requis pour pouvoir utiliser l'outil d'administration MQ JMS (voir «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33) :

- Au moins un des modules de fournisseur de services suivants :
  - LDAP (Lightweight Directory Access Protocol) - `ldap.jar`, `providerutil.jar`.
  - Système de fichiers - `fscontext.jar`, `providerutil.jar`.
- Un fournisseur de services JNDI (Java Naming and Directory Service). Il s'agit de la ressource qui stocke les représentations physiques des objets gérés par l'administrateur. Les utilisateurs de MQ JMS utiliseront probablement un serveur LDAP à cette fin, mais cet outil prend également en charge l'utilisation du fournisseur de services du contexte de système de fichiers. Si un serveur LDAP est utilisé, il doit être configuré pour stocker les objets JMS. Pour plus d'informations sur cette configuration, reportez-vous à l'«Annexe C. Configuration du serveur LDAP pour les objets Java» à la page 367.

Pour pouvoir utiliser les fonctions XOpen/XA deMQ JMS, vous devez disposer de MQSeries V5.2.

Chapitre 2. Procédures d'installation

Le présent chapitre décrit comment installer les classes MQSeries pour Java et les classes MQSeries pour Java Message Service.

Installation de classes MQSeries pour Java et de classes MQSeries pour Java Message Service

Ce produit est disponible pour AIX, AS/400, HP-UX, Linux, Sun Solaris et Windows. Il se compose des éléments suivants :

- classes MQSeries pour Java (MQ base Java) version 5.2.0
- classes MQSeries pour Java Message Service (MQ JMS) version 5.2 (pas sous AS/400)

Pour connaître les caractéristiques de connectivité disponibles sur chaque plateforme, reportez-vous à la section «Options de connexion» à la page 5.

Le produit est livré sous la forme de fichiers compressés disponibles sur le site Web MQSeries, à l'adresse `http://www.ibm.com/software/mqseries/`.

Remarque : Pour OS/390, MQ base Java est livré en tant que SupportPac MQSeries téléchargeable à partir de l'adresse suivante : `http://www.ibm.com/software/mqseries/`.

Pour les dernières versions des classes MQ base Java uniquement, vous pouvez installer MQ base Java version 5.2.0 seul. Pour utiliser les applications MQ JMS, vous devez installer MQ base Java et MQ JMS (l'ensemble étant appelé MQ Java).

MQ base Java est fourni dans les fichiers Java `.jar` suivants :

com.ibm.mq.jar	Ce module prend en charge toutes les options de connexion.
com.ibm.mq.iiop.jar	Ce module ne prend en charge que la connexion VisiBroker. Il est uniquement livré avec la plateforme Windows.
com.ibm.mqbind.jar	Ce module ne prend en charge que les connexions de liaison. Il n'est pas livré ou pris en charge sur toutes les plateformes. Il est recommandé de ne pas l'utiliser dans de nouvelles applications.

MQ JMS est fourni dans le fichier Java `.jar` suivant :

com.ibm.mqjms.jar

Les bibliothèques Java suivantes de Sun Microsystems sont redistribuées avec MQ JMS :

connector.jar	Version 1.0 Public Draft
fscontext.jar	Early Access 4 Release
jms.jar	Version 1.0.2
jndi.jar	Version 1.1.2

ldap.jar	Version 1.0.3
providerutil.jar	Version 1.0

Remarque : Sous OS/390, seul le fichier **com.ibm.mq.jar** est livré. Ce fichier prend en charge les connexions de liaison à MQSeries à partir des services système UNIX et de CICS Transaction Server pour OS/390.

Pour obtenir des instructions d'installation, reportez-vous à la section adaptée à la plateforme de votre choix :

AIX, HP-UX et Sun Solaris	«Installation sous UNIX»
AS/400	«Installation sur l'AS/400» à la page 11
Linux	«Installation sous Linux» à la page 11
Windows	«Installation sous Windows» à la page 12

Lorsque l'installation est terminée, les fichiers et les exemples sont installés dans les emplacements indiqués dans la section «Répertoires d'installation» à la page 12.

Après l'installation, vous devez mettre à jour les variables d'environnement de votre poste, en suivant les instructions de la section «Variables d'environnement» à la page 13.

Remarque : Installez le produit avec précaution, puis installez ou réinstallez la base MQSeries. Assurez-vous que vous n'installez pas MQ base Java version 5.1, car vous reviendriez alors au niveau antérieur du supportMQSeries Java.

Installation sous UNIX

La présente section décrit comment installer MQ Java sous AIX, HP-UX et Sun Solaris. Pour plus de détails sur l'installation de MQ base Java sous Linux, reportez-vous à la section «Installation sous Linux» à la page 11.

Remarque : S'il s'agit d'une installation client uniquement (c'est-à-dire sans installation de serveur MQSeries), vous devez définir le groupe et l'ID utilisateur mqm. Pour plus de détails, reportez-vous au manuel MQSeries - Mise en route associé à la plateforme que vous utilisez.

1. Connectez-vous en tant qu'utilisateur root.
2. Copiez le fichier `ma88_xxx.tar.Z` au format binaire, puis placez-le dans le répertoire `/tmp`, où *xxx* correspond à l'identificateur de la plateforme appropriée :
   - `aix` AIX
   - `hp10` HP-UXv10
   - `hp11` HP-UXv11
   - `sol` Sun Solaris
3. Entrez les commandes suivantes (où *xxx* correspond à l'identificateur de la plateforme appropriée) :

   ```
   uncompress -fv /tmp/ma88_xxx.tar.Z
   tar -xvf /tmp/ma88_xxx.tar
   rm /tmp/ma88_xxx.tar
   ```

   Ces commandes permettent de créer les fichiers et répertoires requis.

4. Utilisez l'outil d'installation adapté à chaque plateforme :
   - Sous AIX, utilisez `smitty` et :
     a. Désinstallez tous les composants dont le nom commence par mqm.java.
     b. Installez les composants à partir du répertoire /tmp.
   - Pour HP-UX, utilisez `sam` et procédez à l'installation à partir du fichier `ma88_hp10` ou `ma88_hp11`, selon le cas.

     Remarque : Java ne prend pas en charge la page de codes 1051 (page de codes par défaut avec HP-UX). Pour exécuter le courtier Publish/Subscribe sous HP-UX, vous devez peut-être remplacer le CCSID du gestionnaire de file d'attente du courtier par une autre valeur telle que 819, par exemple.
   - Pour Sun Solaris, entrez la commande suivante et sélectionnez les options nécessaires :

     ```
     pkgadd -d /tmp mqjava
     ```

     Entrez ensuite la commande suivante :

     ```
     rm -R /tmp/mqjava
     ```

Installation sur l'AS/400

La présente section décrit comment installer MQ base Java sur l'AS/400.

1. Copiez le fichier `ma88_400.zip` dans un répertoire de votre poste.
2. Décompressez le fichier à l'aide de l'utilitaire InfoZip Unzip.
   Cette opération permet d'obtenir le fichier `ma88_400.sav`.
3. Créez un fichier de sauvegarde nommé MA88 dans une bibliothèque adaptée de l'AS/400, par exemple la bibliothèque QGPL :

   ```
   CRTSAVF FILE(QGPL/MA88)
   ```
4. Transférez `ma88_400.sav` dans ce fichier de sauvegarde en tant qu'image au format binaire. Si vous effectuez cette opération via FTP, la commande put doit être proche de la suivante :

   ```
   PUT C:\TEMP\MA88_400.SAV QGPL/MA88
   ```
5. Installez les classes MQSeries pour Java, ID produit 5648C60, à l'aide de la commande RSTLICPGM :

   ```
   RSTLICPGM LICPGM(5648C60) DEV(*SAVF) SAVF(QGPL/MA88)
   ```
6. Supprimez le fichier de sauvegarde créé à l'étape 3 :

   ```
   DLTF FILE(QGPL/MA88)
   ```

Installation sous Linux

La présente section décrit comment installer MQ Java sous Linux.

Pour Linux il existe deux fichiers d'installation qui sont `ma88_linux.tgz` et MQSeriesJava-5.2.0-1.noarch.rpm. Chaque fichier offre une installation identique.

Si vous avez accès au système cible en tant qu'utilisateur root, ou si vous utilisez une base de données RPM (Red Hat Package Manager) pour installer les modules, utilisez le fichier MQSeriesJava-5.2.0-1.noarch.rpm.

Si vous ne disposez pas d'accès au système cible en tant que root, ou si RPM n'est pas installé sur le système cible, utilisez le fichier `ma88_linux.tgz`.

Installation sous Linux

Pour procéder à l'installation à l'aide du fichier ma88_linux.tgz, procédez comme suit :

1. Sélectionnez un répertoire d'installation pour le produit (par exemple, /opt).

   S'il ne fait pas partie de votre répertoire personnel, vous devrez peut-être vous connecter en tant que root.
2. Copiez le fichier `ma88_linux.tgz` dans votre répertoire personnel.
3. Accédez au répertoire d'installation sélectionné, par exemple :

   ```
   cd /opt
   ```
4. Entrez la commande suivante :

   ```
   tar -xpzf ~/ma88_linux.tgz
   ```

   Cette commande permet de créer et de remplir un répertoire nommé mqm dans le répertoire en cours (par exemple, /opt).

Pour procéder à l'installation à l'aide du fichier MQSeriesJava-5.2.0-1.noarch.rpm, procédez comme suit :

1. Connectez-vous en tant qu'utilisateur root.
2. Copiez le fichier MQSeriesJava-5.2.0-1.noarch.rpm dans un répertoire de travail.
3. Entrez la commande suivante :

   ```
   rpm -i MQSeriesJava-5.2.0-1.noarch.rpm
   ```

   Elle permet d'installer le produit dans /opt/mqm/. Vous pouvez l'installer dans un autre répertoire (pour plus de détails, reportez-vous à la documentation de RPM).

Installation sous Windows

La présente section décrit comment installer MQ Java sous Windows.

1. Créez un répertoire vide nommé tmp et accédez à ce répertoire.
2. Copiez le fichier `ma88_win.zip` dans ce répertoire.
3. Décompressez le fichier `ma88_win.zip` à l'aide de l'utilitaire InfoZip Unzip.
4. Exécutez le fichier `setup.exe` à partir de ce répertoire et suivez les indications qui s'affichent.

   Remarque : Pour installer MQ base Java uniquement, sélectionnez les options appropriées à cette étape de la procédure.

Répertoires d'installation

Les fichiers MQ Java V5.2 sont installés dans les répertoires indiqués dans le tableau 2.

Tableau 2. Répertoires d'installation du produit

Plateforme	**Répertoire**
AIX	usr/mqm/java/
AS/400	/QIBM/ProdData/mqm/java/
HP-UX et Sun Solaris	opt/mqm/java/
Linux	*Rép_install*/mqm/java/
Windows 95, 98, 2000 et NT	*Rép_install*\
Remarque : *Rép_install* correspond au répertoire dans lequel vous avez installé le produit. Sous Linux, il s'agit probablement de /opt.	

Variables d'environnement

Après l'installation, vous devez mettre à jour la variable d'environnement CLASSPATH pour y faire figurer les répertoires contenant le code MQ base Java et les exemples. Le tableau 3 présente les instructions CLASSPATH type pour chaque plateforme.

Tableau 3. Exemples d'instructions CLASSPATH pour le produit

Plateforme	Exemple CLASSPATH
AIX	CLASSPATH=*Rép_jdk*/lib/classes.zip: /usr/mqm/java/lib/com.ibm.mq.jar: /usr/mqm/java/lib/connector.jar: /usr/mqm/java/lib: /usr/mqm/java/samples/base:
HP-UX et Sun Solaris	CLASSPATH=*Rép_jdk*/lib/classes.zip: /opt/mqm/java/lib/com.ibm.mq.jar: /opt/mqm/java/lib/connector.jar: /opt/mqm/java/lib: /opt/mqm/java/samples/base:
Windows 95, 98, 2000 et NT	CLASSPATH=C:*Rép_jdk*\lib\classes.zip; *Rép_install*\lib\com.ibm.mq.jar; *Rép_install*\lib\com.ibm.mq.iiop.jar; *Rép_install*\lib\connector.jar; *Rép_install*\lib\; *Rép_install*\samples\base\;
AS/400	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/lib/connector.jar: /QIBM/ProdData/mqm/java/lib: /QIBM/ProdData/mqm/java/samples/base:
Linux	CLASSPATH=*Rép_jdk*/lib/classes.zip: *Rép_install*/mqm/java/lib/com.ibm.mq.jar: *Rép_install*/mqm/java/lib/connector.jar: *Rép_install*/mqm/java/lib: *Rép_install*/mqm/java/samples/base:

Remarques :

1. *Rép_jdk* correspond au répertoire d'installation du JDK.
2. *Rép_install* correspond au répertoire d'installation du produit.

Pour utiliser MQ JMS, vous devez ajouter des fichiers jar à la variable classpath. Ces fichiers sont répertoriés dans la section «Configuration de post-installation» à la page 21.

S'il existe des applications dépendantes du module de liaisons déconseillé `com.ibm.mqbind`, vous devez également ajouter le fichier `com.ibm.mqbind.jar` à la variable classpath.

Sur certaines plateformes, d'autres variables d'environnement doivent être mises à jour, comme le signale le tableau 4.

Tableau 4. Variables d'environnement du produit

Plateforme	Variable d'environnement
AIX	LD_LIBRARY_PATH=/usr/mqm/java/lib
HP_UX	SHLIB_PATH=/opt/mqm/java/lib

Répertoires d'installation

Tableau 4. Variables d'environnement du produit (suite)

Plateforme	Variable d'environnement
Sun Solaris	LD_LIBRARY_PATH=/opt/mqm/java/lib
Windows 95, 98, 2000 et NT	PATH=*Rép_install*\lib
Remarque : *Rép_install* correspond au répertoire d'installation du produit.	

Remarques :

1. Pour utiliser MQSeries Bindings pour Java sous OS/400, assurez-vous que la bibliothèque QMQMJAVA figure dans votre liste de bibliothèques.
2. Assurez-vous que les variables de MQSeries sont ajoutées sans remplacer aucune variable d'environnement existante du système. Si vous remplacez des variables d'environnement existantes, l'application risque d'échouer lors de la compilation ou de l'exécution.

Configuration du serveur Web

Si vous installez MQSeries Java sur un serveur Web, vous pouvez télécharger et exécuter des applications MQSeries Java sur des machines sur lesquelles MQSeries Java n'est pas installé en local. Pour que les fichiers MQSeries Java soient accessibles à votre serveur Web, vous devez configurer ce dernier pour qu'il pointe sur le répertoire d'installation du client. Pour plus de précisions sur cette configuration, consultez la documentation de votre serveur Web.

Remarque : Sous OS/390, les classes installées ne prennent pas en charge les connexions client et ne peuvent pas être téléchargées correctement sur des clients. Toutefois, les fichiers jar d'autres plateformes peuvent être transférés vers OS/390 et distribué à des clients.

Chapitre 3. Utilisation de classes MQSeries pour Java (MQ base Java)

Le présent chapitre décrit les procédures suivantes :

- Configuration du système pour exécuter l'applet et l'application exemples afin de vérifier l'installation de MQ base Java
- Modification des procédures pour exécuter vos propres programmes

La marche à suivre dépend de l'option de connexion que vous souhaitez utiliser. Suivez les instructions de la section correspondant à vos besoins.

Utilisation de l'applet exemple pour vérifier le client TCP/IP

MQ base Java contient une applet de vérification de l'installation, `mqjavac.html`. Elle permet de contrôler le mode client connecté par TCP/IP de MQ base Java. (Voir aussi «Vérification à l'aide de l'application exemple» à la page 17.)

L'applet se connecte à un gestionnaire de files d'attente donné, tente de passer tous les appels MQSeries et émet des messages de diagnostic en cas d'incident.

Vous pouvez exécuter l'applet à partir de l'afficheur d'applets fourni avec le JDK. L'afficheur d'applets peut accéder à un gestionnaire de files d'attente sur n'importe quel hôte.

Dans tous les cas, si l'applet ne s'exécute pas correctement, suivez les recommandations des messages de diagnostic puis essayez d'exécuter de nouveau l'applet.

Utilisation de l'applet exemple sur l'AS/400

Le système d'exploitation OS/400 ne dispose pas d'interface utilisateur graphique (GUI) native. Pour exécuter l'applet exemple, vous devez utiliser AWT (Remote Abstract Window Toolkit) pour Java ou CBJ (Class Broker pour Java) sur un support matériel prenant en charge les graphiques. Vous pouvez également contrôler le client à partir de la ligne de commande (voir «Vérification à l'aide de l'application exemple» à la page 17).

Configuration d'un gestionnaire de files d'attente pour les connexions client

Procédez comme indiqué ci-dessous pour configurer le gestionnaire de files d'attente afin qu'il accepte les demandes de connexion des clients.

Client TCP/IP

1. Définissez un canal de connexion serveur en procédant comme suit :

   Pour les plateformes autres que l'AS/400 :

   a. Démarrez votre gestionnaire de files d'attente à l'aide de la commande strmqm.

   b. Entrez la commande suivante pour démarrer le programme runmqsc :

      ```
      runmqsc
      ```

c. Définissez un canal exemple nommé JAVA.CHANNEL, en entrant :

```
DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Canal exemple pour MQSeries Client pour Java')
```

Pour la plateforme AS/400 :

a. Démarrez votre gestionnaire de files d'attente à l'aide de la commande STRMQM.

b. Définissez un canal exemple nommé JAVA.CHANNEL, en entrant :

```
CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
MCAUSERID(SOMEUSERID) TEXT('Canal exemple pour MQSeries Client pour Java')
```

où QMGRNAME correspond au nom du gestionnaire de files d'attente et SOMEUSERID, à un ID utilisateur AS/400 disposant des droits d'accès appropriés aux ressources MQSeries.

2. Lancez un programme d'écoute à l'aide des commandes suivantes :

Pour les systèmes d'exploitation OS/2 et NT :

Entrez la commande :

```
runmqlsr -t tcp [-m NOMGEST] -p 1414
```

Remarque : Si vous utilisez le gestionnaire de files d'attente par défaut, l'option -m peut être omise.

Utilisation de VisiBroker pour Java sous Windows NT :

Démarrez le serveur IIOP (Internet Inter-ORB Protocol) à l'aide de la commande suivante :

```
java com.ibm.mq.iiop.Server
```

Remarque : Pour arrêter le serveur IIOP, tapez la commande suivante :

```
java com.ibm.mq.iiop.samples.AdministrationApplet shutdown
```

Pour les systèmes d'exploitation UNIX :

Configurez le démon inetd de sorte qu'il ouvre les canaux MQSeries. Pour savoir comment procéder, consultez le manuel *MQSeries - Clients*.

Pour le système d'exploitation OS/400 :

Entrez la commande :

```
STRMQMLSR MQMNAME(QMGRNAME)
```

où QMGRNAME correspond au nom du gestionnaire de files d'attente.

Exécution à partir de l'afficheur d'applets

Pour recourir à cette méthode, votre poste doit être équipé du JDK (Java Developer's Kit).

Procédure d'installation en local

1. Accédez au répertoire d'exemples de votre langue.
2. Entrez :

```
appletviewer mqjavac.html
```

Procédure d'installation sur serveur Web

Entrez la commande :

```
appletviewer http://Web.server.host/MQJavaclient/mqjavac.html
```

Remarques :

1. Sur certaines plateformes, la commande est 'applet' au lieu de 'appletviewer'.

2. Sur certaines plateformes, vous aurez peut-être besoin de sélectionner «Properties» dans le menu «Applet» situé en haut à gauche de l'écran, puis de donner à l'option «Network Access» la valeur «Unrestricted».

Cette procédure devrait permettre la connexion à tout gestionnaire de files d'attente fonctionnant sur un système hôte auquel vous avez accès par TCP/IP.

Personnalisation de l'applet de vérification

Le fichier `mqjavac.html` contient des paramètres en option. Ces paramètres vous permettent de modifier l'applet pour l'adapter à vos besoins. Chaque paramètre est défini par une ligne HTML de format suivant :

```
<!PARAM name="xxx" value="yyy">
```

Pour définir une valeur de paramètre, supprimez le point d'exclamation initial, puis modifiez les valeurs entre guillemets. Vous pouvez définir les paramètres suivants :

hostname Valeur à afficher dans la zone de saisie du nom d'hôte.

port Valeur à afficher dans la zone de saisie du numéro de port.

channel Valeur à afficher dans la zone de saisie du canal.

queueManager
Valeur à afficher dans la zone de saisie du gestionnaire de files d'attente.

userID Indique l'ID utilisateur spécifié pour la connexion au gestionnaire de files d'attente.

mot de passe Indique le mot de passe spécifié pour la connexion au gestionnaire de files d'attente.

trace Demande à MQ base Java de créer un fichier de trace. N'utilisez cette option que sur demande du service après vente IBM.

Vérification à l'aide de l'application exemple

Un programme de vérification d'installation, MQIVP, est fourni avec MQ base Java. Vous pouvez l'utiliser pour tester tous les modes de connexion de MQ base Java. Ce programme vous invite à faire un certain nombre de choix et à fournir certaines informations afin de déterminer le mode de connexion que vous souhaitez tester. Procédez comme suit pour vérifier votre installation :

1. Si vous désirez tester une connexion client :
   a. Configurez votre gestionnaire de files d'attente selon les instructions de la section «Configuration d'un gestionnaire de files d'attente pour les connexions client» à la page 15.
   b. Exécutez les autres étapes de cette procédure sur l'ordinateur client.

   Si vous désirez tester une connexion directe, exécutez la fin de cette procédure sur le serveur MQSeries.
2. Accédez au répertoire des exemples.
3. Entrez :

   ```
   java MQIVP
   ```

Le programme tente :

a. de se connecter au gestionnaires de files d'attente nommé et de s'en déconnecter ;

b. d'ouvrir la file d'attente locale par défaut, d'y placer un message, d'en extraire un, puis de la refermer ;

c. de renvoyer un message si les opérations précédentes aboutissent.

4. A l'invite (1), conservez la valeur par défaut 'MQSeries'.
5. A l'invite (2) :
   - Pour utiliser une connexion TCP/IP, indiquez un nom hôte de serveur MQSeries.
   - Pour utiliser la connexion directe (mode liens), laissez cette zone vide.

Vous trouverez ci-dessous un exemple de dialogue possible avec l'application. Les invites et vos réponses dépendent de votre réseau MQSeries.

```
Please enter the type of connection (MQSeries)              : (MQSeries)(1)
Please enter the IP address of the MQSeries server          : myhost(2)
Please enter the port to connect to                         : (1414)(3)
Please enter the server connection channel name              : JAVA.CHANNEL(3)
Please enter the queue manager name  :
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 transport is functioning correctly.
Press Enter to continue...
```

Remarques :

1. Si vous avez choisi une connexion serveur, vous ne verrez pas s'afficher les invites repérées par le signe (3).
2. Sous OS/390, les invites (1), (2) et (3) ne s'affichent pas.
3. Sous OS/400, la commande `java MQIVP` ne peut être exécutée qu'à partir de l'interface interactive Qshell (Qshell est l'option 30 de l'OS/400, 5769-SS1). Vous pouvez également exécuter l'application à l'aide de la commande CL `RUNJVA CLASS(MQIVP)`.
4. Pour utiliser MQSeries bindings pour Java sous OS/400, assurez-vous que la bibliothèque QMQMJAVA figure dans votre liste de bibliothèques.

Utilisation de la connectivité VisiBroker

Si vous utilisez VisiBroker, les instructions de la section «Configuration d'un gestionnaire de files d'attente pour les connexions client» à la page 15 n'ont pas besoin d'être suivies.

Pour tester une installation exploitant VisiBroker, suivez les instructions de la section «Vérification à l'aide de l'application exemple» à la page 17, mais à l'invite (1), tapez `VisiBroker` en respectant la distinction majuscules-minuscules.

Utilisation de CICS Transaction Server pour OS/390

1. Définissez le programme d'application exemple dans CICS.
2. Définissez une transaction permettant d'exécuter l'application exemple.
3. Insérez le nom du gestionnaire de files d'attente dans le fichier utilisé en entrée standard.

4. Exécutez la transaction.

Le résultat est envoyé dans les fichiers utilisés en sortie standard et d'erreurs.

Pour plus de détails sur l'exécution des programmes Java et sur la définition des fichiers d'entrée et de sortie, reportez-vous à la documentation CICS.

Exécution de programmes MQ base Java personnalisés

Pour exécuter vos propres applets ou applications Java, suivez les instructions fournies pour les programmes de vérification, en remplaçant simplement «mqjavac.html» ou «MQIVP» par le nom de votre application.

Pour plus de détails sur l'écriture d'applications et d'applets MQ base Java, reportez-vous à la «Partie 2. Programmation avec MQ base Java» à la page 47.

Résolution des incidents MQ base Java

Si un programme ne s'exécute pas correctement, lancez l'applet ou le programme de vérification de l'installation, et suivez les conseils fournis par les messages de diagnostic. Ces deux programmes sont décrits dans le «Chapitre 3. Utilisation de classes MQSeries pour Java (MQ base Java)» à la page 15.

Si l'incident persiste et si vous devez prendre contact avec le service d'assistance IBM, vous devrez peut-être activer la fonction de trace. Deux méthodes permettent d'activer celle-ci, en fonction du mode de connexion utilisé (connexion client ou connexion directe). Reportez-vous aux sections suivantes pour connaître les procédures appropriées pour votre système.

Traçage de l'applet exemple

Pour activer la fonction de trace avec l'applet exemple, modifiez le fichier `mqjavac.html`. Repérez la ligne suivante :

```
<!PARAM name="trace" value="1">
```

Supprimez le point d'exclamation, puis remplacez la valeur 1 par un chiffre compris entre 1 et 5, en fonction du niveau de détail voulu (plus ce chiffre est élevé, plus la fonction de trace fournit d'informations). La ligne modifiée a l'apparence suivante :

```
<PARAM name="trace" value="n">
```

où *n* est un chiffre compris entre 1 et 5.

Les résultats du traçage s'inscrivent sur la console Java ou dans le fichier journal de trace Java de votre navigateur Web.

Traçage de l'application exemple

Pour activer la fonction de trace du programme `MQIVP`, entrez la commande suivante :

```
java MQIVP -trace n
```

où *n* est un chiffre compris entre 1 et 5 en fonction du niveau de détail requis (plus ce chiffre est élevé, plus la fonction de trace fournit d'informations).

Activation de la fonction de trace pour MQ base Java

Pour plus d'informations sur l'utilisation de la fonction de trace, reportez-vous à la section «Traçage des programmes MQ base Java» à la page 72.

Traçage avec CICS Transaction Server pour OS/390

Lorsque vous utilisez CICS Transaction Server pour OS/390, vous ne pouvez pas fournir directement des arguments de ligne de commande au programme. Vous devez écrire un petit programme encapsuleur qui appelle MQIVP.main() avec les arguments appropriés.

Messages d'erreur

Voici certains des messages d'erreurs les plus courants :

Unable to identify local host IP address
Le serveur n'est pas connecté au réseau.

Recommandation: Connectez le serveur au réseau, puis renouvelez l'opération.

Unable to load file gatekeeper.ior
Cet incident peut se manifester sur un serveur Web déployant des applets VisiBroker si le fichier `gatekeeper.ior` ne se trouve pas à l'emplacement approprié.

Recommandation : Redémarrez VisiBroker Gatekeeper à partir du répertoire dans lequel l'applet est déployé. Le fichier `gatekeeper` sera écrit dans ce répertoire.

Failure: Missing software, may be MQSeries, or VBROKER_ADM variable
Cet incident survient au niveau du programme exemple MQIVP lorsque votre environnement logiciel Java est incomplet.

Recommandation : Sur le client, assurez-vous que la variable d'environnement VBROKER_ADM contient l'adresse du répertoire d'administration de VisiBroker pour Java (adm), puis réessayez.

Sur le serveur, assurez-vous que la dernière version de MQ base Java est installée, puis réessayez.

NO_IMPLEMENT
Une erreur de communication affecte les agents intelligents de VisiBroker.

Recommandation : Consultez votre documentation VisiBroker.

COMM_FAILURE
Une erreur de communication affecte les agents intelligents de VisiBroker.

Recommandation : Utilisez le même numéro de port pour tous les agents intelligents VisiBroker, puis renouvelez l'opération. Consultez la documentation VisiBroker.

MQRC_ADAPTER_NOT_AVAILABLE
Si vous recevez ce message d'erreur lorsque vous essayez d'utiliser VisiBroker, la cause en est peut-être que la classe Java org.omg.CORBA.ORB n'est pas définie par CLASSPATH.

Recommandation : Assurez-vous que votre instruction CLASSPATH comprend le chemin d'accès aux fichiers VisiBroker `vbjorb.jar` et `vbjapp.jar`.

MQRC_ADAPTER_CONN_LOAD_ERROR
Si vous obtenez ce message d'erreur lors de l'exécution sur OS/390, vérifiez que les jeux de données MQSeries SCSQANLE et SCSQAUTH se trouvent dans votre instruction STEPLIB.

Chapitre 4. Utilisation de classes MQSeries pour Java Message Service (MQ JMS)

Ce chapitre décrit les tâches suivantes :

- Configuration de votre système pour utiliser les programmes de test et d'exemple
- Exécution du programme point à point IVT (Installation Verification Test) pour vérifier l'installation des classes MQSeries pour Java Message Service
- Exécution de l'exemple de programme PSIVT (Publish/Subscribe Installation Verification Test) pour vérifier l'installation de Publish/Subscribe
- Exécution de vos propres programmes

Configuration de post-installation

Pour que toutes les ressources nécessaires soient disponibles pour les programmes MQ JMS, vous devez mettre à jour les variables système suivantes :

Chemin de classe

Le bon déroulement des programmes JMS suppose qu'un certain nombre de modules Java soient disponibles pour JVM. Vous devez indiquer ces derniers dans le chemin de classe après avoir obtenu et installé les modules nécessaires.

Ajoutez les fichiers .jar suivants au chemin de classe :

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar
- jms.jar
- jndi.jar
- jta.jar
- ldap.jar
- providerutil.jar

Variables d'environnement

Le sous-répertoire `bin` du répertoire d'installation de MQ JMS contient un certain nombre de scripts. Ces derniers sont destinés à être utilisés comme des raccourcis commodes pour des actions courantes. De nombreux scripts supposent que la variable d'environnement `MQ_JAVA_INSTALL_PATH` est définie et qu'elle pointe sur le répertoire dans lequel MQ JMS est installé. Il n'est pas obligatoire de définir cette variable, mais si vous ne le faites pas, vous devez éditer les scripts en conséquence dans le répertoire `bin`.

Sous Windows NT, vous pouvez définir le chemin de classe et la nouvelle variable d'environnement à l'aide de l'onglet **Environnement** des **Propriétés du système**. Sous UNIX, ces éléments sont normalement définis à partir des scripts de connexion de chaque utilisateur. Pour chaque plateforme, vous pouvez choisir d'utiliser des scripts pour gérer d'autres chemins de classe et variables d'environnement pour d'autres projets.

Configuration supplémentaire pour le mode Publication/Souscription

Avant de pouvoir utiliser l'implémentation MQ JMS de Publish/Subscribe JMS, vous devez procéder à une configuration plus poussée :

Vérifiez que le courtier est en cours d'exécution

Pour vérifier que le courtier MQSeries Publish/Subscribe est installé et en cours d'exécution, utilisez la commande suivante :

```
dspmqbrk -m MY.QUEUE.MANAGER
```

où MY.QUEUE.MANAGER est le nom du gestionnaire de file d'attente sur lequel s'exécute le courtier. Si le courtier est en cours d'exécution, un message semblable au message suivant s'affiche :

```
MQSeries message broker for queue manager MY.QUEUE.MANAGER running.
```

Si le système d'exploitation indique qu'il ne peut exécuter la commande dspmqbrk, vérifiez que le courtier MQSeries Publish/Subscribe est installé correctement.

Si le système d'exploitation indique que le courtier n'est pas actif, démarrez ce dernier à l'aide de la commande suivante :

```
strmqbrk -m MY.QUEUE.MANAGER
```

Créez les files d'attente système MQ JMS

Pour que l'implémentation de MQ JMS Publish/Subscribe soit effective, diverses files d'attente système doivent être créées. Un script est fourni dans le sous-répertoire bin du répertoire d'installation de MQ JMS pour vous aider dans cette tâche. Pour utiliser le script, entrez la commande suivante :

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

En cas d'erreur, vérifiez que vous avez tapé correctement le nom du gestionnaire de file d'attente, et vérifiez que ce dernier est actif.

Files d'attente supposant une autorisation pour les utilisateurs non privilégiés

Les utilisateurs non privilégiés nécessitent une autorisation pour accéder aux files d'attente utilisées par JMS. Pour obtenir des informations détaillées sur le contrôle d'accès dans MQSeries, voir le chapitre relatif à la protection des objets MQSeries dans le manuel *MQSeries - Administration du système*.

Pour le mode JMS point à point, les questions relatives au contrôle d'accès sont les mêmes que pour les classes MQSeries pour Java :

- Les files d'attente utilisées par QueueSender supposent des droits d'insertion.
- Les files d'attente utilisées par QueueReceivers et QueueBrowsers supposent des droits d'extraction, d'interrogation et de recherche.
- La méthode QueueSession.createTemporaryQueue suppose l'accès à la file d'attente modèle définie dans la zone QueueConnectionFactory temporaryModel (par défaut, SYSTEM.DEFAULT.MODEL.QUEUE).

Pour le mode JMS publication/souscription, les files d'attente suivantes sont utilisées :

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

En outre, toute application qui publie des messages suppose l'accès à la file d'attente STREAM indiquée dans l'objet TopicConnectionFactory en cours d'utilisation. La valeur par défaut est :

SYSTEM.BROKER.DEFAULT.STREAM

Exécution du programme IVT point à point

Cette section décrit le programme IVT (Installation Verification Test) point à point fourni avec MQ JMS.

IVT essaye de vérifier l'installation en établissant une connexion au gestionnaire de files d'attente par défaut sur la machine locale, à l'aide de MQ JMS en mode liens. IVT envoie ensuite un message à la file d'attente `SYSTEM.DEFAULT.LOCAL.QUEUE` et le lit à nouveau.

Vous pouvez exécuter le programme selon deux modes.

Recherche JNDI d'objets gérés par l'administrateur
Le mode JNDI force le programme à obtenir ses objets gérés par l'administrateur depuis un espace annuaire JNDI, ce qui correspond au fonctionnement attendu des applications JMS client. (Pour plus de précisions sur les objets gérés par l'administrateur, voir «Administration des objets JMS» à la page 37.) Cette méthode d'appel présente les mêmes conditions préalables que l'outil d'administration (voir «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33).

Sans recherche JNDI d'objets gérés par l'administrateur
Si vous ne souhaitez pas utiliser JNDI, les objets gérés par l'administrateur peuvent être créés lors de l'exécution en lançant IVT en mode non-JNDI. Du fait qu'un référentiel basé sur JNDI est relativement complexe à configurer, nous vous recommandons de lancer d'abord IVT sans JNDI.

Vérification point à point sans JNDI

Un script, appelé `IVTRun` pour UNIX, ou `IVTRun.bat` pour Windows NT, est fourni pour exécuter IVT. Ce fichier est installé dans le sous-répertoire `bin` du répertoire d'installation.

Pour lancer le test sans JNDI, tapez la commande suivante :

```
IVTRun -nojndi
```

En mode client, pour lancer le test sans JNDI, tapez la commande suivante :

```
IVTRun -nojndi -client -m <gest_file> -host <nom_hôte> [-port <port>]
              [-channel <canal>]
```

où :

gest_file correspond au nom du gestionnaire de files d'attente auquel vous souhaitez vous connecter.

nom_hôte est l'hôte sur lequel s'exécute le gestionnaire de files d'attente

port est le port TCP/IP sur lequel le programme d'écoute du gestionnaire de files d'attente s'exécute (par défaut, 1414)

canal est le canal de connexion client (par défaut, SYSTEM.DEF.SVRCONN)

Si le test aboutit, le résultat doit être semblable à ce qui suit :

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - 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: Message Class:   jms_text          JMSType:          null
JMSDeliveryMode: 2                              JMSExpiration:    0
JMSPriority:     4                              JMSMessageID:     ID:414d5120716
d312020202020202020203000c43713400000
JMSTimestamp:    935592657000                   JMSCorrelationID: null
JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
JMSReplyTo:      null
JMSRedelivered:  false
JMS_IBM_Format:MQSTR            JMS_IBM_PutApplType:11
JMSXGroupSeq:1                  JMSXDeliveryCount:0
JMS_IBM_MsgType:8               JMSXUserID:kingdon
JMSXApplID:D:\jdk1.1.8\bin\java.exe
A simple text message from the MQJMSIVT program
Reply string equals original string
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Vérification point à point avec JNDI

Pour lancer IVT avec JNDI, le serveur LDAP doit être actif et configuré pour accepter les objets Java. Le message suivant indique une connexion avec le serveur LDAP, mais une mauvaise configuration de ce dernier :

```
Unable to bind to object
```

Ce message signifie que le serveur ne stocke pas les objets Java ou que les permissions relatives aux objets ou au suffixe ne sont pas correctes. Voir «Vérification de la configuration du serveur LDAP» à la page 367.

En outre, les objets suivants, gérés par l'administrateur, doivent pouvoir être obtenus à partir d'un espace annuaire JNDI :

- MQQueueConnectionFactory
- MQQueue

Un script, appelé IVTSetup pour UNIX, ou IVTSetup.bat pour Windows NT, est fourni pour créer ces objets automatiquement. Entrez la commande :

```
IVTSetup
```

Le script appelle l'outil d'administration de MQ JMS (voir «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33) et crée les objets dans un espace annuaire JNDI.

L'objet MQQueueConnectionFactory est joint sous le nom ivtQCF (pour LDAP, cn=ivtQCF). Toutes ses propriétés présentent les valeurs par défaut :

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

L'objet MQQueue est joint sous le nom ivtQ (cn=ivtQ). La valeur de la propriété QUEUE devient QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE). Toutes les autres propriétés présentent les valeurs par défaut :

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

Lorsque les objets gérés par l'administrateur sont créés dans l'espace annuaire JNDI, lancez le script IVTRun (IVTRun.bat pour Windows NT) à l'aide de la commande suivante :

```
IVTRun [ -t ] [ -url <"providerURL"> [ -icf <initCtxFact> ] ]
```

où :

-t active la fonction de trace (par défaut, désactivée)

providerURL correspond à l'emplacement JNDI des objets gérés par l'administrateur. Si l'objet InitialContextFactory par défaut est ouvert, il s'agit d'une URL LDAP de la forme suivante :

```
ldap://hostname.company.com/contextName
```

Si un fournisseur de services de système de fichiers est utilisé (voir initCtxFact ci-dessous), l'URL présente la forme suivante :

```
file://directorySpec
```

Remarque : Faites figurer la chaîne *providerURL* entre guillemets (″).

initCtxFact est le nom de classe de InitialContextFactory. La valeur par défaut concerne un fournisseur de service LDAP et présente la valeur :

```
com.sun.jndi.ldap.LdapCtxFactory
```

Si un fournisseur de service de système de fichiers est utilisé, attribuez à ce paramètre la valeur suivante :

```
com.sun.jndi.fscontext.RefFSContextFactory
```

Si le test se termine normalement, le résultat est semblable au résultat non-JNDI, si ce n'est que les lignes 'create' QueueConnectionFactory et Queue indiquent l'extraction de l'objet depuis JNDI. Le fragment de code suivant représente un exemple.

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service - Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
 ...
```

Même si cela n'est pas strictement nécessaire, il est utile de retirer les objets créés par le script IVTSetup de l'espace annuaire JNDI. Un script appelé IVTTidy (IVTTidy.bat pour Windows NT) est fourni à cette fin.

Reprise sur incident IVT

Les remarques suivantes vous seront utiles si le test n'aboutit pas :

- Pour les messages d'erreur concernant le chemin de classe, vérifiez que votre chemin de classe est configuré de manière appropriée, comme décrit à la section «Configuration de post-installation» à la page 21.
- IVT est susceptible d'échouer en renvoyant un message 'failed to create MQQueueManager', et un message supplémentaire comprenant le numéro 2059. Cela indique que MQSeries n'a pas été en mesure de se connecter au gestionnaire de files d'attente local par défaut sur la machine sur laquelle vous avez lancé IVT. Vérifiez que le gestionnaire de files d'attente est actif et qu'il est bien défini comme gestionnaire de files d'attente par défaut.
- Un message 'failed to open MQ queue' indique que MQSeries est bien connecté au gestionnaire de files d'attente par défaut, mais n'a pas pu ouvrir 'SYSTEM.DEFAULT.LOCAL.QUEUE'. Cela peut indiquer que la file d'attente n'existe pas sur votre gestionnaire de files d'attente par défaut ou que la file d'attente n'est pas activée pour PUT et GET. Ajoutez ou activez la file d'attente pour la durée du test.

Le tableau 5 répertorie les classes testées par IVT, ainsi que leur module d'origine :

Tableau 5. Classes testées par IVT

Classe	**fichier Jar**
Classes MQSeries JMS	com.ibm.mqjms.jar
com.ibm.mq.MQMessage	com.ibm.mq.jar
javax.jms.Message	jms.jar
javax.naming.InitialContext	jndi.jar
javax.resource.cci.Connection	connector.jar
javax.transaction.xa.XAException	jta.jar
com/sun/jndi/toolkit/ComponentDirContext	providerutil.jar

Tableau 5. Classes testées par IVT (suite)

Classe	fichier Jar
com.sun.jndi.ldap.LdapCtxFactory	ldap.jar

Test PSIVT

Le programme PSIVT (Publish/Subscribe Installation Verification Test) est fourni uniquement sous forme compilée. Il figure dans le module `com.ibm.mq.jms`.

PSIVT tente de :

1. Créer un diffuseur de publications, `p`, publiant sur la rubrique `MQJMS/PSIVT/Information`
2. Créer un souscripteur, `s`, souscrivant à la rubrique `MQJMS/PSIVT/Information`
3. Utiliser `p` pour publier un simple message de texte
4. Utiliser `s` pour recevoir un message en attente dans sa file d'entrée

Lorsque vous lancez PSIVT, le diffuseur de publications publie le message et le souscripteur reçoit et affiche le message.Le diffuseur de publications publie dans le flux par défaut du courtier. Le souscripteur ne dispose pas d'une souscription durable, n'effectue pas de sélection de message et accepte des messages en provenance des connexions locales. Il effectue une réception synchrone, attendant un maximum de 5 secondes l'arrivée d'un message.

Vous pouvez lancer PSIVT, tout comme IVT, en mode JNDI ou autonome. Le mode JNDI utilise JNDI pour extraire un objet `TopicConnectionFactory` et un objet `Topic` d'un espace annuaire JNDI. Si JNDI n'est pas utilisé, ces objets sont créés à l'exécution.

Vérification Publish/Subscribe sans JNDI

Un script 'PSIVTRun' appelé `PSIVTRun` (`PSIVTRun.bat` sur Windows NT) est fourni pour exécuter PSIVT. Ce fichier se trouve dans le sous-répertoire `bin` du répertoire d'installation.

Pour lancer le test sans JNDI, tapez la commande suivante :

```
PSIVTRun -nojndi [-m <gest_file>] [-t]
```

En mode client, pour lancer le test sans JNDI, tapez la commande suivante :

```
PSIVTRun -nojndi -client -m <gest_file> -host <nom_hôte> [-port <port>]
              [-channel <canal>] [-t]
```

où :

-nojndi signifie sans recherche JNDI d'objets gérés par l'administrateur

gest_file correspond au nom du gestionnaire de files d'attente auquel vous souhaitez vous connecter.

nom_hôte est l'hôte sur lequel s'exécute le gestionnaire de files d'attente

port est le port TCP/IP sur lequel le programme d'écoute du gestionnaire de files d'attente s'exécute (par défaut, 1414)

canal est le canal de connexion client (par défaut, SYSTEM.DEF.SVRCONN)

-t active la fonction de trace (par défaut, désactivée)

PSIVT (Publish/Subscribe Installation Verification Test)

Si le test aboutit, le résultat doit être semblable à ce qui suit :

```
5648-C60 (c) Copyright IBM Corp. 1999. All Rights Reserved.
MQSeries Classes for Java(tm) Message Service
Publish/Subscribe Installation Verification Test
Creating a TopicConnectionFactory
Creating a Topic
Creating a Connection
Creating a Session
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...

Got message:

JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d2e504f4c415249532e4254b7dc3753700000
  JMSTimestamp:    937232048000
  JMSCorrelationID:ID:414d515800000000000000000000000000000000000000000000
  JMSDestination:  topic
  ://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_Format:MQSTR
  UNIQUE_CONNECTION_ID:937232047753
  JMS_IBM_PutApplType:26
  JMSXGroupSeq:1
  JMSXDeliveryCount:0
  JMS_IBM_MsgType:8
  JMSXUserID:hollingl
  JMSXApplID:QM.POLARIS.BROKER
A simple text message from the MQJMSPSIVT program

Reply string equals original string
Closing TopicSubscriber
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT completed OK
PSIVT finished
```

Vérification Publish/Subscribe avec JNDI

Pour lancer PSIVT en mode JNDI, deux objets gérés par l'administrateur doivent pouvoir être obtenus à partir d'un espace annuaire JNDI :

- Un objet TopicConnectionFactory joint sous le nom `ivtTCF`
- Un objet Topic joint sous le nom `ivtT`

Vous pouvez définir ces objets à l'aide de l'outil d'administration de MQ JMS (voir «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33) et en utilisant les commandes suivantes :

```
DEFINE TCF(ivtTCF)
```

Cette commande définit l'objet TopicConnectionFactory.

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

Cette commande définit l'objet Topic.

Ces définitions supposent qu'un gestionnaire de files d'attente par défaut, sur lequel s'exécute le courtier, est disponible. Pour plus de détails sur la configuration de ces objets pour utiliser un gestionnaire de files d'attente autre que celui par défaut, voir «Administration des objets JMS» à la page 37. Ces objets doivent résider dans un contexte sur lequel pointe le paramètre de ligne de commande `-url` décrit ci-dessous.

Pour lancer le test en mode JNDI, entrez la commande suivante :

```
PSIVTRun -url <purl> [-icf <initcf>] [-t]
```

où :

-t active la fonction de trace (par défaut, désactivée)

-url <purl> est l'URL de l'emplacement JNDI dans lequel réside les objets gérés par l'administrateur

-icf <initcf> est l'objet initialContextFactory pour JNDI [`com.sun.jndi.ldap.LdapCtxFactory`]

Si le test se termine normalement, le résultat est semblable au résultat non-JNDI, si ce n'est que les lignes 'create' `QueueConnectionFactory` et `Queue` indiquent l'extraction de l'objet depuis JNDI.

Reprise sur incident PSIVT

Les remarques suivantes vous seront utiles si le test n'aboutit pas :

- Le message suivant :

  ```
  *** The broker is not running! Please start it using 'strmqbrk' ***
  ```

  indique que le courtier est installé sur le gestionnaire de files d'attente cible, mais que sa file d'attente de contrôle contient des messages en attente. Cela indique que le courtier n'est pas en cours d'exécution. Pour l'activer, utilisez la commande `strmqbrk`. (Voir «Configuration supplémentaire pour le mode Publication / Souscription» à la page 22.)

- Si le message suivant s'affiche :

  ```
  Unable to connect to queue manager: <default>
  ```

  vérifiez que votre système MQSeries a bien configuré un gestionnaire de files d'attente par défaut.

- Si le message suivant s'affiche :

  ```
  Unable to connect to queue manager: ...
  ```

  vérifiez que l'objet TopicConnectionFactory géré par l'administrateur, utilisé par PSIVT, est configuré avec un nom de gestionnaire de files d'attente valide. Si vous avez utilisé l'option `-nojndi`, vérifiez que vous avez indiquéun gestionnaire de files d'attente valide (utilisez l'option `-m`).

- Si le message suivant s'affiche :

  ```
  Unable to access broker control queue on queue manager: ...
  vérifiez que le courtier est installé dans le gestionnaire de files d'attente
  ```

  vérifiez que l'objet TopicConnectionFactory géré par l'administrateur, utilisé par PSIVT, est configuré avec le nom du gestionnaire de files d'attente sur lequel le courtier est installé. Si vous avez utilisé l'option `-nojndi`, vérifiez que vous avez fourni un nom de gestionnaire de files d'attente (utilisez l'option `-m`).

Exécution de vos propres programmes MQ JMS

Pour plus d'informations sur l'écriture de vos propres programmes MQ JMS, voir «Chapitre 10. Ecriture de programmes MQ JMS» à la page 177.

MQ JMS comprend un fichier utilitaire, `runjms` (`runjms.bat` pour Windows NT), destiné à vous aider à exécuter les programmes fournis ainsi que les programmes que vous avez écrits.

Cet utilitaire fournit des emplacements par défaut pour les fichiers de trace et les journaux, et vous permet d'ajouter tout paramètre d'exécution dont votre application peut avoir besoin. Le script fourni suppose que la variable d'environnement MQ_JAVA_INSTALL_PATH a pour valeur le répertoire dans lequel MQ JMS est installé. Le script suppose également que les sous-répertoires `trace` et `log` de ce répertoire sont utilisés respectivement pour les sorties de trace et de journaux. Ces emplacements ne constituent que des suggestions et vous pouvez éditer le script pour utiliser le répertoire de votre choix.

Utilisez la commande suivante pour lancer votre application :

```
runjms <nom de classe de l'application> [arguments spécifiques de
l'application]
```

Pour plus d'informations sur l'écriture d'applications et d'applets MQ JMS, consultez la «Partie 3. Programmation avec MQ JMS» à la page 175.

Résolution des incidents

Si un programme ne s'exécute pas correctement, lancez le programme de vérification d'installation décrit dans le «Chapitre 4. Utilisation de classes MQSeries pour Java Message Service (MQ JMS)» à la page 21, puis suivez les conseils fournis par les messages de diagnostic.

Traçage des programmes

La fonction de trace MQ JMS est fournie pour aider le personnel IBM à diagnostiquer les incidents client.

La fonction de trace est désactivée par défaut, car les résultats deviennent rapidement trop importants et présentent peu d'intérêt dans des circonstances normales.

Si les résultats de la fonction de trace vous sont demandés, vous pouvez activer cette dernière en attribuant à la propriété Java MQJMS_TRACE_LEVEL l'une des valeurs suivantes :

on trace uniquement les appels MQ JMS

base trace les appels MQ JMS et les appels MQ base Java sous-jacents

Par exemple :

```
java -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

Pour désactiver la fonction de trace, affectez à MQJMS_TRACE_LEVEL la valeur **off**.

Par défaut, les résultats de la fonction de trace sont générés dans un fichier appelé mqjms.trc dans le répertoire de travail en cours. Vous pouvez réacheminer ces résultats dans un autre répertoire à l'aide de la propriété Java MQJMS_TRACE_DIR.

Par exemple :

```
java -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

Le script utilitaire runjms définit ces propriétés à l'aide des variables d'environnement MQJMS_TRACE_LEVEL et MQ_JAVA_INSTALL_PATH, de la manière suivante :

```
java -DMQJMS_LOG_DIR=%MQ_JAVA_INSTALL_PATH%\log
-DMQJMS_TRACE_DIR=%MQ_JAVA_INSTALL_PATH%\trace
-DMQJMS_TRACE_LEVEL=%MQJMS_TRACE_LEVEL% %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Il s'agit seulement d'une suggestion et vous pouvez y apporter les modifications de votre choix.

Consignation

La fonction de consignation MQ JMS est fournie pour signaler les incidents graves, notamment ceux susceptibles de se rapporter à des erreurs de configuration plutôt que de programmation. Par défaut, la sortie de journal est envoyée au flux System.err qui figure généralement à l'emplacement stderr de la console sur laquelle JVM est exécuté.

Vous pouvez réacheminer la sortie vers un fichier à l'aide d'une propriété Java qui indique le nouvel emplacement, par exemple :

```
java -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

Le script utilitaire runjms, contenu dans le sous-répertoire bin du répertoire d'installation de MQ JMS, définit cette propriété de la manière suivante :

```
<MQ_JAVA_INSTALL_PATH>/log
```

où MQ_JAVA_INSTALL_PATH est le chemin d'accès de votre répertoire d'installation de MQ JMS. Il ne s'agit que d'une suggestion, que vous pouvez modifiez librement.

Lorsque le journal est réacheminé vers un fichier, il est généré sous forme binaire. L'utilitaire formatLog (formatLog.bat pour Windows NT) est fourni pour vous permettre d'afficher le journal. Il convertit le fichier au format texte. Cet utilitaire se trouve dans le répertoire bin de votre répertoire d'installation de MQ JMS. Effectuez la conversion de la manière suivante :

```
formatLog <fichier_d'entrée> <fichier_de_sortie>
```

Consignation

Chapitre 5. Utilisation de l'outil d'administration MQ JMS

L'outil d'administration permet aux administrateurs de définir les propriétés de huit types d'objet MQ JMS et de les stocker dans un espace annuaire JNDI. Les clients JMS peuvent ensuite extraire ces objets gérés par l'administrateur de l'espace annuaire via JNDI et les utiliser.

Les objets JMS que vous pouvez gérer via cet outil sont les suivants :
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory
- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Pour plus de détails sur ces objets, reportez-vous à la section «Administration des objets JMS» à la page 37.

Remarque : JMSWrapXAQueueConnectionFactory et JMSWrapXATopicConnectionFactory sont des classes propres à WebSphere. Elles se trouvent dans le module **com.ibm.ejs.jms.mq**.

Cet outil permet également aux administrateurs de manipuler les sous-contextes associés aux espaces annuaires dans le JNDI. Voir la section «Manipulation des sous-contextes» à la page 37.

Appel de l'outil d'administration

L'outil d'administration possède une interface de ligne de commande. Vous pouvez l'utiliser en mode interactif ou pour lancer un traitement par lots. Le mode interactif fournit une invite de commande permettant de saisir des commandes d'administration. En mode traitement par lots, la commande de démarrage de l'outil contient le nom d'un fichier script de commandes d'administration.

Pour démarrer l'outil en mode interactif, tapez la commande suivante :

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

où :

-t	Active la fonction de trace (elle est désactivée par défaut)
-v	Génère une sortie en mode prolixe (par défaut, la sortie est en mode concis)
-cfg config_filename	Nom d'un fichier de configuration de substitution (voir la section«Configuration» à la page 34)

Une invite de commande s'affiche et indique que l'outil est prêt à accepter les commandes d'administration. Cette invite s'affiche initialement sous la forme suivante :

```
InitCtx>
```

Elle indique que le contexte en cours (c'est-à-dire le contexte JNDI auquel toutes les opérations de répertoire et d'appellation font référence) est le contexte initial défini dans le paramètre de configuration PROVIDER_URL (voir la section «Configuration»).

Lorsque vous parcourez l'espace annuaire, l'invite est modifiée afin de toujours indiquer le contexte en cours.

Pour démarrer l'outil en mode traitement par lots, tapez la commande suivante :

```
JMSAdmin <test.scp
```

où *test.scp* est un fichier script qui contient les commandes d'administration (voir la section «Commandes d'administration» à la page 36). La dernière commande du fichier doit être la commande `END`.

Configuration

Vous devez configurer l'outil d'administration en affectant des valeurs aux trois paramètres suivants :

INITIAL_CONTEXT_FACTORY
Indique le fournisseur de services utilisé par l'outil. Cette propriété peut prendre l'une des trois valeurs suivantes :
- com.sun.jndi.ldap.LdapCtxFactory (pour LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (pour le contexte de système de fichiers)
- com.ibm.ejs.ns.jndi.CNInitialContextFactory (pour utiliser le référentiel CosNaming WebSphere)

PROVIDER_URL
Indique l'URL du contexte initial de la session, la racine de toutes les opérations JNDI effectuées par l'outil. Cette propriété peut prendre trois formes :
- ldap://hostname/contextname (pour LDAP)
- file:[drive:]/pathname (pour le contexte de système de fichiers)
- iiop://hostname[:port] /[?TargetContext=ctx] (pour accéder à l'espace annuaire CosNaming WebSphere de "base")

SECURITY_AUTHENTICATION
Indique si JNDI transmet les données d'identification de sécurité au fournisseur de services. Ce paramètre n'est utilisé que 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'authentification CRAM-MD5)

Si aucune valeur correcte n'est fournie, la propriété prend par défaut la valeur none. Pour plus de détails sur la sécurité via l'outil d'administration, reportez-vous à la section «Sécurité» à la page 35.

Ces paramètres sont définis 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` (voir la section «Appel de l'outil d'administration» à la page 33). 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 ce fichier

dans le répertoire en cours, puis dans le répertoire <MQ_JAVA_INSTALL_PATH>/bin (où <MQ_JAVA_INSTALL_PATH> représente le chemin d'accès au répertoire d'installation de MQ JMS.)

Le fichier de configuration est un fichier texte se composant d'une série de paires de valeurs clé séparées par le signe '='. En voici un exemple :

```
#Définition du fournisseur de services
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Définition du contexte initial
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Définition du type d'authentification
    SECURITY_AUTHENTICATION=none
```

(Le signe '#' dans la première colonne signale un commentaire ou une ligne non utilisée).

Le programme d'installation fournit un exemple de fichier de configuration appelé JMSAdmin.config, situé dans le répertoire <MQ_JAVA_INSTALL_PATH>/bin. Modifiez ce fichier pour l'adapter à la configuration du système.

Configuration pour WebSphere

Pour que l'outil d'administration (ou toute application client devant effectuer des recherches) puisse fonctionner avec le référentiel WebSphere CosNaming, vous devez disposer de la configuration qui suit :

- La variable CLASSPATH doit inclure les fichiers WebSphere .jar associés à JNDI :
  - Pour WebSphere version 3.5 :

    <WSAppserver>\lib\ejs.jar
- La variable PATH pour WebSphere version 3.5 doit inclure :

  <WSAppserver>\jdk\jre\bin

  où <WSAppserver> désigne le chemin d'installation de WebSphere.

Sécurité

Les administrateurs doivent connaître les effets de la propriété SECURITY_AUTHENTICATION décrite dans la section «Configuration» à la page 34.

- Si ce paramètre a pour valeur *none*, JNDI ne transmet aucune donnée d'identification de sécurité au fournisseur de services et une ″authentification anonyme″ est effectuée.
- Si le paramètre a pour valeur simple ou CRAM-MD5, les données d'identification de sécurité sont transmises via JNDI au fournisseur de services sous-jacent. Ces données d'identification se présentent sous la forme d'un nom distinctif d'utilisateur (DN utilisateur) et d'un mot de passe.

Si ces données sont nécessaires, l'utilisateur est invité à les indiquer lors de l'initialisation de l'outil.

Remarque : Le texte saisi par l'utilisateur est répercuté à l'écran, y compris pour le mot de passe. Par conséquent, prenez garde à ce que les mots de passe ne soient pas portés à la connaissance d'utilisateurs non autorisés.

L'outil n'effectue aucune authentification par lui-même. Cette tâche relève du serveur LDAP. Il est de la responsabilité de l'administrateur du serveur LDAP de

définir et de gérer les droits d'accès aux différentes parties du répertoire. En cas d'échec de l'authentification, l'outil affiche un message d'erreur et s'arrête.

Pour plus d'informations sur la sécurité et sur JNDI, reportez-vous à la documentation fournie sur le site Web Java de Sun (`http://java.sun.com`).

Commandes d'administration

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* désigne l'une des instructions (ou verbe) d'administration présentées dans le tableau 6. Toute commande correcte doit comporter au moins une instruction (et une seule), qui apparaît au début de la commande au format standard ou abrégé.

Les paramètres associés à une instruction dépendent de cette dernière. Par exemple, l'instruction `END` ne peut être associée à aucun paramètre, mais l'instruction `DEFINE` peut être associée à de nombreux paramètres (de 1 à 20). Les instructions pouvant être associées à au moins un paramètre sont décrites en détail dans la suite du présent chapitre.

Tableau 6. Instructions d'administration

Instruction		**Description**
Format standard	**Format abrégé**	
ALTER	ALT	Modifie au moins l'une des propriétés d'un objet géré par l'administrateur donné.
DEFINE	DEF	Crée et stocke un objet géré par l'administrateur, ou crée un nouveau sous-contexte.
DISPLAY	DIS	Affiche les propriétés d'un ou plusieurs objets gérés par l'administrateur stockés, ou le contenu du contexte en cours.
DELETE	DEL	Supprime un ou plusieurs objets gérés par l'administrateur de l'espace annuaire, ou supprime un sous-contexte vide.
CHANGE	CHG	Modifie le contexte en cours, ce qui permet à l'utilisateur de parcourir l'espace annuaire sous le contexte initial (autorisation en attente).
COPY	CP	Effectue une copie d'un objet géré par l'administrateur stocké et la stocke sous un autre nom.
MOVE	MV	Modifie le nom sous lequel un objet géré par l'administrateur est stocké.
END		Ferme l'outil d'administration.

Les noms d'instructions ne prennent pas en compte la distinction majuscules / minuscules.

Généralement, pour terminer les commandes, vous devez appuyer sur la touche de retour chariot. Cependant, vous pouvez remplacer cette affectation de touche en tapant le signe '+' directement devant le retour chariot.

Vous pouvez ainsi entrer des commandes sur plusieurs lignes, comme l'indique l'exemple qui suit :

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Les lignes commençant par un caractère *, # ou / sont considérées comme des commentaires ou des lignes à ignorer.

Manipulation des sous-contextes

Pour manipuler les sous-contextes d'espace annuaire, vous pouvez utiliser les instructions `CHANGE`, `DEFINE`, `DISPLAY` et `DELETE`. Leur utilisation est décrite dans le tableau 7.

Tableau 7. Syntaxe et description des commandes utilisées pour manipuler des sous-contextes

Syntaxe de la commande	Description
DEFINE CTX(ctxName)	Tente de créer un nouveau sous-contexte enfant du contexte en cours, sous le nom ctxName. Cette tentative échoue en cas de violation d'accès, si le sous-contexte existe déjà ou si le nom fourni est incorrect.
DISPLAY CTX	Affiche le contenu du contexte en cours. Les objets gérés par l'administrateur sont accompagnés de la lettre `'a'` et les sous-contextes, de la lettre `'[D]'`. Le type Java de chaque objet est également affiché.
DELETE CTX(ctxName)	Permet de supprimer le contexte enfant du contexte en cours, portant le nom ctxName. Cette tentative échoue si le contexte est introuvable ou non vide, ou en cas de violation d'accès.
CHANGE CTX(ctxName)	Modifie le contexte en cours, de telle manière qu'il fasse référence au contexte enfant portant le nom ctxName. Il est possible de fournir l'une des deux valeurs spéciales suivantes pour ctxName : **=UP** Effectue un déplacement vers le parent du contexte en cours. **=INIT** Effectue directement un déplacement vers le contexte initial. Cette tentative échoue si le contexte n'existe pas, ou en cas de violation d'accès.

Administration des objets JMS

Cette section décrit les huit types d'objets pouvant être gérés par l'outil d'administration. Elle inclut une description détaillée de chacune de leurs propriétés configurables et des instructions qui permettent de les manipuler.

Types d'objets

Le tableau 8 à la page 38 présente les huit types d'objets gérés par l'administrateur. La colonne Mot clé contient les chaînes par lesquelles vous pouvez remplacer *TYPE* dans les commandes présentées dans le tableau 9 à la page 39.

Administration des objets JMS

Tableau 8. Types d'objets JMS pouvant être gérés par l'outil d'administration

Type d'objet		**Description**
Java	**Mot clé**	
MQQueueConnectionFactory	QCF	Implémentation MQSeries de l'interface JMS QueueConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine point à point de JMS.
MQTopicConnectionFactory	TCF	Implémentation MQSeries de l'interface JMS TopicConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine publication / souscription de JMS.
MQQueue	Q	Implémentation MQSeries de l'interface JMS Queue. Il s'agit de la destination des messages du domaine point à point de JMS.
MQTopic	T	Implémentation MQSeries de l'interface JMS Topic. Il s'agit de la destination des messages du domaine publication / souscription de JMS.
MQXAQueueConnectionFactory[1]	XAQCF	Implémentation MQSeries de l'interface JMS XAQueueConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine point à point de JMS utilisant les versions XA des classes JMS.
MQXATopicConnectionFactory[1]	XATCF	Implémentation MQSeries de l'interface JMS XATopicConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine publication / souscription de JMS utilisant les versions XA des classes JMS.
JMSWrapXAQueueConnectionFactory[2]	WSQCF	Implémentation MQSeries de l'interface JMS QueueConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine point à point de JMS utilisant les versions XA des classes JMS avec WebSphere.
JMSWrapXATopicConnectionFactory[2]	WSTCF	Implémentation MQSeries de l'interface JMS TopicConnectionFactory. Il s'agit d'un objet fabrique permettant de créer des connexions dans le domaine publication / souscription de JMS utilisant les versions XA des classes JMS avec WebSphere.

Tableau 8. Types d'objets JMS pouvant être gérés par l'outil d'administration (suite)

Type d'objet		Description
Java	**Mot clé**	
1. Ces classes sont fournies afin de pouvoir être utilisées par les fournisseurs de serveurs d'applications. Elles ne sont pas directement utiles aux programmeurs d'applications. 2. Utilisez ce style de ConnectionFactory si vous souhaitez que vos sessions JMS prennent part aux transactions globales coordonnées par WebSphere.		

Instructions utilisées avec les objets JMS

Vous pouvez utiliser les instructions `ALTER`, `DEFINE`, `DISPLAY` , `DELETE`, `COPY` et `MOVE` pour manipuler les objets gérés par l'administrateur dans l'espace annuaire. Le tableau 9 décrit brièvement leur utilisation. Remplacez *TYPE* par le mot clé représentant l'objet géré requis (voir le tableau 8 à la page 38).

Tableau 9. 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 d'un objet géré donné en fonction de celles fournies. Cette tentative échoue en cas de violation d'accès, si l'objet spécifié est introuvable ou si les nouvelles propriétés fournies sont incorrectes.
DEFINE *TYPE*(nom) [propriété]*	Tente de créer un objet géré de type *TYPE* associé aux propriétés fournies, et de le stocker sous le nom `nom` dans le contexte en cours. Cette tentative échoue en cas de violation d'accès, si le nom fourni est incorrect ou existe déjà ou si les propriétés fournies sont incorrectes.
DISPLAY *TYPE*(nom)	Affiche les propriétés de l'objet géré de type *TYPE*, liées sous le nom `nom` dans le contexte en cours. Cette tentative échoue si l'objet n'existe pas, ou en cas de violation d'accès.
DELETE *TYPE*(nom)	Tente de supprimer l'objet géré de type *TYPE* (qui porte le nom `nom`) du contexte en cours. Cette tentative échoue si l'objet n'existe pas, ou en cas de violation d'accès.
COPY *TYPE*(nomA) *TYPE*(nomB)	Permet d'effectuer une copie de l'objet géré de type *TYPE* (qui porte le nom `nomA`) en lui affectant le nom `nomB`. La totalité de l'opération s'effectue dans le cadre du contexte en cours. Cette tentative échoue si l'objet à copier n'existe pas, si un objet portant le nom nomB existe déjà ou en cas de violation d'accès.
MOVE *TYPE*(nomA) *TYPE*(nomB)	Permet de renommer l'objet géré de type *TYPE* portant le nom `nomA` en lui affectant le nom `nomB`. La totalité de l'opération s'effectue dans le cadre du contexte en cours. Cette tentative échoue si l'objet à renommer n'existe pas, si un objet portant le nom nomB existe déjà ou en cas de violation d'accès.

Création d'objets

Les objets sont créés et stockés dans un espace annuaire JNDI à l'aide de la commande suivante :

`DEFINE TYPE(nom) [propriété]*`

Instruction DEFINE, suivie d'une référence à l'objet géré *TYPE*(nom), et de zéro ou plusieurs *propriétés* (voir la section «Propriétés»).

Conventions d'appellation LDAP

Pour que vous puissiez stocker des objets en environnement LDAP, leur nom doit respecter certaines conventions. Les noms de l'objet et du sous-contexte, en particulier, doivent comporter un préfixe, tel que cn= (nom commun) ou ou= (unité d'organisation).

L'outil d'administration simplifie l'utilisation des fournisseurs de services LDAP car il vous permet de faire référence à un nom d'objet ou de contexte sans préfixe. En effet, si vous ne fournissez pas de préfixe, l'outil ajoute automatiquement un préfixe par défaut (cn=) au nom que vous fournissez.

L'exemple suivant illustre cette opération :

```
InitCtx> DEFINE Q(FileAttenteTest)

InitCtx> DISPLAY CTX

    Contenu de InitCtx

      a  cn=FileAttenteTest          com.ibm.mq.jms.MQQueue

     1 Objet(s)
       0 Contexte(s)
       1 Liaison(s), 1 Géré(es)
```

Notez que, bien que le nom d'objet fourni (FileAttenteTest) ne comporte pas de préfixe, l'outil en ajoute automatiquement un pour assurer le respect des conventions d'appellation LDAP. De même, le fait de lancer la commande DISPLAY Q(FileAttenteTest) provoque également l'ajout d'un préfixe.

Il peut s'avérer nécessaire de configurer le serveur LDAP en vue du stockage des objets Java. Pour plus d'informations, reportez-vous à l'«Annexe C. Configuration du serveur LDAP pour les objets Java» à la page 367.

Propriétés

Une propriété se compose d'une paire nom-valeur au format suivant :

```
NOM_PROPRIETE(valeur_propriété)
```

Les noms de propriété ne prennent pas en compte la distinction majuscules/minuscules et sont limités aux noms indiqués dans le tableau 10. Ce tableau indique également les valeurs admises pour chaque propriété.

Tableau 10. Noms de propriété et valeurs admises

Propriété		**Valeurs admises (valeur par défaut en gras)**
Format standard	**Format abrégé**	
DESCRIPTION	DESC	Toute chaîne
TRANSPORT	TRAN	• **BIND** - Les connexions utilisent les liaisons MQSeries. • CLIENT - La connexion client est utilisée.
CLIENTID	CID	Toute chaîne
QMANAGER	QMGR	Toute chaîne
HOSTNAME	HOST	Toute chaîne
PORT		Tout entier positif

Tableau 10. Noms de propriété et valeurs admises (suite)

Propriété		**Valeurs admises (valeur par défaut en gras)**
Format standard	**Format abrégé**	
CHANNEL	CHAN	Toute chaîne
CCSID	CCS	Tout entier positif
RECEXIT	RCX	Toute chaîne
RECEXITINIT	RCXI	Toute chaîne
SECEXIT	SCX	Toute chaîne
SECEXITINIT	SCXI	Toute chaîne
SENDEXIT	SDX	Toute chaîne
SENDXITINIT	SDXI	Toute chaîne
TEMPMODEL	TM	Toute chaîne
MSGRETENTION	MRET	• **Yes** - Les messages inutiles restent dans la file d'attente en entrée. • No - Les messages inutiles sont gérés en fonction de leurs options de destination.
BROKERVER	BVER	**V1** - Seule valeur actuellement admise.
BROKERPUBQ	BPUB	Toute chaîne (la valeur par défaut est **SYSTEM.BROKER.DEFAULT.STREAM**)
BROKERSUBQ	BSUB	Toute chaîne (la valeur par défaut est **SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE**)
BROKERDURSUBQ	BDSUB	Toute chaîne (la valeur par défaut est **SYSTEM.JMS.D.SUBSCRIPTION.QUEUE**)
BROKERCCSUBQ	CCSUB	Toute chaîne (la valeur par défaut est **SYSTEM.JMS.ND.CC.SUBSCRIPTION.QUEUE**)
BROKERCCDSUBQ	CCDSUB	Toute chaîne (la valeur par défaut est **SYSTEM.JMS.D.CC.SUBSCRIPTION.QUEUE**)
BROKERQMGR	BQM	Toute chaîne
BROKERCONQ	BCON	Toute chaîne
EXPIRY	EXP	• **APP** - Le délai d'expiration peut être défini par l'application JMS. • UNLIM - Aucun délai d'expiration. • Tout entier positif représentant un délai d'expiration en millisecondes.
PRIORITY	PRI	• **APP** - La priorité peut être définie par l'application JMS. • QDEF - La priorité prend la valeur de la file d'attente par défaut. • Tout entier compris entre 0 et 9.
PERSISTENCE	PER	• **APP** - La persistance peut être définie par l'application JMS. • QDEF - La persistance prend la valeur de la file d'attente par défaut. • PERS - Les messages sont persistants. • NON - Les messages ne sont pas persistants.
TARGCLIENT	TC	• **JMS** - La cible du message est une application JMS. • MQ - La cible du message est une application MQSeries traditionnelle non JMS.
ENCODING	ENC	Voir la section «Propriété ENCODING» à la page 44

Tableau 10. Noms de propriété et valeurs admises (suite)

Propriété		Valeurs admises (valeur par défaut en gras)
Format standard	**Format abrégé**	
QUEUE	QU	Toute chaîne
TOPIC	TOP	Toute chaîne

Nombreuses sont les propriétés qui ne concernent qu'un sous-ensemble spécifique de types d'objets. Le tableau 11 présente les associations propriété-type d'objet qui sont admises et donne une brève description de chaque propriété.

Tableau 11. Associations propriété-type d'objet admises

Propriété	Types d'objets admis						Description
	QCF	**TCF**	**Q**	**T**	**WSQCF XAQCF**	**WSTCF XATCF**	
DESCRIPTION	O	O	O	O	O	O	Description de l'objet stocké.
TRANSPORT	O	O			O[1]	O[1]	Indique si les connexions utiliseront les liaisons MQ ou une connexion client.
CLIENTID	O	O			O	O	Identificateur de chaîne pour le client.
QMANAGER	O	O	O		O	O	Nom du gestionnaire de files d'attente auquel devra s'effectuer la connexion.
PORT	O	O					Port sur lequel le gestionnaire de files d'attente se met à l'écoute.
HOSTNAME	O	O					Nom de l'hôte hébergeant le gestionnaire de files d'attente.
CHANNEL	O	O					Nom du canal de connexion client utilisé.
CCSID	O	O	O	O			ID de jeu de caractères codés à utiliser au niveau des connexions.
RECEXIT	O	O					Nom complet de la classe de l'exit de réception utilisé.
RECEXITINIT	O	O					Chaîne d'initialisation de l'exit de réception.
SECEXIT	O	O					Nom complet de la classe de l'exit de sécurité utilisé.
SECEXITINIT	O	O					Chaîne d'initialisation de l'exit de sécurité.
SENDEXIT	O	O					Nom complet de la classe de l'exit d'émission utilisé.
SENDEXITINIT	O	O					Chaîne d'initialisation de l'exit d'émission.
TEMPMODEL	O				O		Nom de la file d'attente modèle à partir de laquelle sont créées les files d'attente temporaires.
MSGRETENTION	O				O		Indique si le client de la connexion conserve les messages inutiles dans la file d'attente en entrée.
BROKERVER		O				O	Version du courtier utilisé.
BROKERPUBQ		O				O	Nom de la file d'attente de courtier en entrée à utiliser (file d'attente de flot).

Tableau 11. Associations propriété-type d'objet admises (suite)

Propriété	Types d'objets admis						Description
	QCF	TCF	Q	T	WSQCF XAQCF	WSTCF XATCF	
BROKERSUBQ		O				O	Nom de la file d'attente à partir de laquelle les messages de souscription non persistants sont extraits.
BROKERDURSUBQ				O			Nom de la file d'attente à partir de laquelle les messages de souscription persistants sont extraits.
BROKERCCSUBQ		O				O	Nom de la file d'attente à partir de laquelle les messages de souscription non persistants sont extraits pour un ConnectionConsumer.
BROKERCCDSUBQ				O			Nom de la file d'attente à partir de laquelle les messages de souscription persistants sont extraits pour un ConnectionConsumer.
BROKERQMGR		O				O	Nom du gestionnaire de files d'attente sur lequel le courtier s'exécute.
BROKERCONQ		O				O	Nom de la file d'attente de contrôle du courtier.
EXPIRY			O	O			Délai au-delà duquel les messages à envoyer vers une destination expirent.
PRIORITY			O	O			Priorité des messages envoyés vers une destination.
PERSISTENCE			O	O			Persistance des messages envoyés vers une destination.
TARGCLIENT			O	O			Zone indiquant si le format MQSeries RFH2 est utilisé pour échanger des informations avec les applications cible.
ENCODING			O	O			Algorithme de codage utilisé pour cette destination.
QUEUE			O				Nom sous-jacent de la file d'attente représentant cette destination.
TOPIC				O			Nom sous-jacent de la rubrique représentant cette destination.

Remarques :

1. Pour les objets WSTCF, WSQCF, XATCF et XAQCF, seul le type de transport BIND est autorisé.
2. L'«Annexe A. Correspondance entre les propriétés de l'outil d'administration et les propriétés programmables» à la page 363 décrit les relations entre les propriétés définies par l'outil et les propriétés programmables.
3. La propriété TARGCLIENT indique si le format MQSeries RFH2 est utilisé pour échanger des informations avec les applications cible.

   La constante MQJMS_CLIENT_JMS_COMPLIANT indique que le format RFH2 est utilisé pour l'envoi d'informations. Les applications qui utilisent MQ JMS

comprennent le format RFH2. Vous devez définir la constante MQJMS_CLIENT_JMS_COMPLIANT lorsque vous échangez des informations avec une application cible MQ JMS.

La constante MQJMS_CLIENT_NONJMS_MQ indique que le format RFH2 n'est pas utilisé pour l'envoi d'informations. Généralement, cette valeur est utilisée pour une application MQSeries existante (c'est-à-dire une application qui n'accepte pas le format RFH2).

Dépendances entre propriétés

Certaines propriétés sont dépendantes les unes des autres. Cela signifie qu'il est inutile de fournir certaines propriétés si aucune autre propriété n'est définie. Les deux groupes de propriétés spécifiques au niveau desquels ce problème peut se poser sont les propriétés client et les chaînes d'initialisation d'exit.

Propriétés client

Si la propriété `TRANSPORT(CLIENT)` n'a pas été explicitement définie au niveau d'une fabrique de connexion, le mode de transport utilisé au niveau des connexions et fourni par la fabrique est MQ Bindings (liaisons MQ). Aucune des propriétés client au niveau de cette connexion ne peut donc être configurée. Les propriétés concernées sont les suivantes :

- HOST
- PORT
- CHANNEL
- CCSID
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT

Si vous tentez de définir l'une de ces propriétés sans définir la propriété `TRANSPORT` à `CLIENT`, une erreur sera générée.

Chaînes d'initialisation d'exit

Il n'est pas possible de définir les chaînes d'initialisation d'un exit si le nom de l'exit correspondant n'a pas été fourni. Les propriétés d'initialisation d'exit sont les suivantes :

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

Par exemple, si vous spécifiez `RECEXITINIT(myString)` sans `RECEXIT(nom.classe.exit)`, une erreur est générée.

Propriété ENCODING

Les valeurs admises pour la propriété `ENCODING` sont plus complexes que celles des autres propriétés. La propriété Encoding est construite à partir de trois sous-propriétés :

integer encoding	Normal ou inversé
decimal encoding	Normal ou inversé

floating-point encoding IEEE normal, IEEE inversé ou System/390

La propriété ENCODING s'exprime sous forme d'une chaîne de trois caractères ayant la syntaxe suivante :
{N|R}{N|R}{N|R|3}

Dans cette chaîne :

- N est synonyme de normal
- R est synonyme d'inversé
- 3 représente System/390
- Le premier caractère représente la sous-propriété *integer encoding*
- Le deuxième caractère représente la sous-propriété *decimal encoding*
- Le troisième caractère représente la sous-propriété *floating-point encoding*

Vous obtenez ainsi un ensemble de 12 valeurs possibles pour ENCODING.

Il existe une valeur complémentaire, la chaîne NATIVE, qui définit les valeurs d'encodage adéquates pour la plateforme Java.

Voici quelques exemples de combinaisons admises pour ENCODING :

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

Exemples de conditions d'erreur

Cette section fournit des exemples de conditions d'erreur pouvant survenir lors de la création d'un objet.

Propriété inconnue

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(jambon et champignons)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Propriété inconnue : PIZZA
```

Propriété incorrecte pour un objet

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Propriété incorrecte pour un objet QCF : PRI
```

Type incorrect pour une valeur de propriété

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Valeur incorrecte pour la propriété CCS : English
```

Valeur de propriété hors plage

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Valeur incorrecte pour la propriété PRI : 12
```

Conflit de priorité - client/liaisons

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Propriété incorrecte dans ce contexte : conflit d'attributs client-liaisons
```

Conflit de priorité - Initialisation d'exit

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
Impossible de créer un objet correct. Vérifiez les paramètres indiqués.
Propriété incorrecte dans ce contexte : Chaîne ExitInit fournie
sans chaîne Exit
```

Partie 2. Programmation avec MQ base Java

Chapitre 6. Introduction destinée aux programmeurs

Le présent chapitre contient des informations générales destinées aux programmeurs. Pour plus de détails sur l'écriture de programmes, reportez-vous au «Chapitre 7. Ecriture de programmes MQ base Java» à la page 53.

Pourquoi utiliser l'interface Java ?

L'interface de programmation de classes MQSeries pour Java vous permet de bénéficier des nombreux avantages de Java pour développer des applications MQSeries :

- Le langage de programmation Java est **simple à utiliser**.

  Il n'oblige pas à recourir aux fichiers d'en-tête, aux pointeurs, aux structures, aux unions et aux surcharges d'opérateurs. Les programmes écrits en Java sont plus faciles à développer et à déboguer que leurs homologues écrits en C ou C++.

- Java est **orienté objet**.

  L'orientation objet de Java est comparable à celle de C++, mais il n'existe pas de notion d'héritage multiple. Dans Java, c'est le concept d'interface qui est mis en oeuvre.

- Dans sa nature même, Java est **réparti**.

  Les bibliothèques de classes Java contiennent une bibliothèque de routines pour gérer des protocoles TCP/IP tels que HTTP et FTP. Les programmes Java peuvent accéder aux URL aussi simplement qu'à un système de fichiers.

- Java est **solide**.

  Dans Java, l'accent est mis sur la détection a priori d'incidents éventuels, sur le contrôle dynamique (à l'exécution) et sur l'élimination de situations susceptibles de générer des erreurs. Java recourt au concept de référencement, qui annule le risque d'écraser les données en mémoire ou de les altérer.

- Java est **sécurisé**.

  Java est conçu pour fonctionner en réseau ou dans un environnement réparti et la sécurité est au coeur de sa conception. Les programmes Java ne peuvent pas déborder de leur pile d'exécution ni altérer la mémoire en dehors de l'espace qui leur est alloué. Lorsque les programmes Java sont téléchargés depuis Internet, ils ne parviennent pas à lire ni écrire des fichiers locaux.

- Les programmes Java sont **portables**.

  Il n'existe aucun aspect dépendant de la mise en oeuvre dans la spécification Java. Le compilateur Java génère un format de fichier objet neutre par rapport à l'architecture. Le code compilé est exécutable sur de nombreux processeurs, tant que l'environnement d'exécution Java est présent sur la machine concernée.

Si vous écrivez votre application à l'aide de classes MQSeries pour Java, les utilisateurs peuvent télécharger les codes d'octet Java (appelés *applets*) correspondant à votre programme à partir d'Internet. Ils peuvent ensuite les exécuter sur leur poste. Ainsi, les utilisateurs ayant accès à votre serveur Web peuvent charger et exécuter votre application sans qu'aucune installation particulière n'ait besoin d'être effectuée sur leur ordinateur.

Lorsqu'une mise à jour du programme est nécessaire, il suffit de mettre à niveau la copie de ce programme installée sur le serveur Web. Dès lors, chaque fois que

l'applet est sollicitée par un utilisateur, celui-ci reçoit automatiquement la version mise à jour. Ces principes entraînent une réduction substantielle des coûts d'installation et de mise à jour par rapport aux applications de type client classiques, surtout lorsque le nombre de postes sur le réseau est élevé.

Si vous placez votre applet sur un serveur Web accessible de l'extérieur du coupe-feu de l'entreprise, tout utilisateur d'Internet peut télécharger et utiliser votre application. Votre système MQSeries peut donc recevoir des messages de n'importe où sur Internet. Une telle possibilité permet d'envisager de créer des applications de service, d'assistance technique et de commerce électronique entièrement nouvelles, accessibles à partir d'Internet.

Interface des classes MQSeries pour Java

L'interface de programmation d'application procédurale MQSeries s'articule autour des instructions suivantes :

```
MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
```

Ces instructions prennent toutes comme argument l'indicateur de l'objet MQSeries auquel elles s'appliquent. Comme Java est orienté objet, l'interface de programmation Java inverse ce principe. Votre programme se composera d'un ensemble d'objets MQSeries sur lesquels vous agirez en appelant des méthodes, comme dans l'exemple présenté ci-dessous.

Lorsque vous utilisez l'interface procédurale, vous vous déconnectez d'un gestionnaire de files d'attente en appelant MQDISC(Hconn, CompCode, Reason), *Hconn* étant l'identificateur du gestionnaire de files d'attente concerné.

Dans l'interface Java, le gestionnaire de files d'attente est représenté par un objet de la classe MQQueueManager. Pour vous déconnecter du gestionnaire de files d'attente, il suffit d'appeler la méthode disconnect() de cette classe.

```
// Déclaration d'un objet de type gestionnaire de files d'attente
MQQueueManager queueManager=new MQQueueManager();
...
// Traitement souhaité...
...
// Déconnexion du gestionnaire de files d'attente
queueManager.disconnect();
```

Kit JDK (Java Development Kit)

Pour pouvoir compiler vos applets ou vos applications, vous devez avoir accès au kit JDK (Java Development Kit) adapté à votre plateforme de développement. Le kit JDK contient tous les constructeurs, variables et classes Java standard, ainsi que les interfaces sur lesquelles sont fondées les classes MQSeries pour Java. Il contient également les outils requis pour compiler et exécuter les applets et les programmes sur chaque plateforme prise en charge.

Vous pouvez télécharger les kitsJDK à partir du catalogue mondial de téléchargement IBM, à l'adresse suivante :

```
http://www.ibm.com/software/download
```

Vous pouvez également développer des applications à l'aide du kit JDK inclus avec l'environnement de développement intégré IBM Visual Age for Java.

Kit JDK (Java Development Kit)

Pour compiler des applications Java sur l'AS/400, vous devez d'abord installer les éléments suivants :
- Le kit de développement AS/400 pour Java, 5769-JV1
- L'interpréteur Qshell, OS/400 (5769-SS1) Option 30

Bibliothèque de classes MQSeries pour Java

Les classes MQSeries pour Java sont un ensemble de classes Java qui permettent aux applets et aux applications Java de fonctionner avec MQSeries.

Les classes suivantes sont fournies :
- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Les interfaces Java suivantes sont fournies :
- MQC
- MQPoolServicesEventListener
- MQReceiveExit
- MQSecurityExit
- MQSendExit

La mise en oeuvre des interfaces Java suivantes est également fournie. Toutefois, elles ne doivent pas être utilisées directement par les applications.
- MQConnectionManager
- javax.resource.spi.ManagedConnection
- javax.resource.spi.ManagedConnectionFactory
- javax.resource.spi.ManagedConnectionMetaData

Dans Java, un *module* est un groupe de classes liées entre elles. Les classes et interfaces MQSeries sont livrées sous la forme d'un module Java nommé `com.ibm.mq`. Pour inclure le module des classes MQSeries pour Java dans votre programme, ajoutez la ligne suivante en tête de votre fichier source :

```
import com.ibm.mq.*;
```

Bibliothèque de classes MQ base Java

Chapitre 7. Ecriture de programmes MQ base Java

Pour avoir accès aux files d'attente MQSeries à l'aide de classes MQSeries pour Java, écrivez des programmes Java contenant des appels d'insertion et d'extraction de messages dans les files d'attente MQSeries. Ces programmes peuvent prendre la forme d'*applets* Java, de *servlets* Java ou d'*applications* Java.

Ce chapitre fournit des informations qui vous aideront à écrire des applets, servlets et applications Java permettant de dialoguer avec les systèmes MQSeries. Pour plus de détails sur une classe particulière, reportez-vous au «Chapitre 9. Classes et interfaces MQ base Java» à la page 83.

Ecriture d'une applet ou d'une application ?

C'est du type de connexion utilisé et de l'endroit d'où vous voulez que le programme soit exécutable que dépend le choix entre l'écriture d'une applet, d'une servlet ou d'une application.

Les principales différences entre une applet et une application sont les suivantes :

- Les applets sont exécutées à partir d'un afficheur d'applets ou d'un navigateur Web, les servlets sont exécutées sur un serveur d'applications Web, tandis que les applications sont exécutées de façon autonome.
- Les applets peuvent être téléchargées d'un serveur Web vers un ordinateur utilisant un navigateur Web, mais cette possibilité n'existe pas pour les applications et les servlets.

Pour choisir entre l'écriture d'une applet et l'écriture d'une application, appliquez les règles générales suivantes :

- Si vous voulez que vos programmes puissent être exécutés sur des machines ne disposant pas de classes MQSeries pour Java en local, écrivez des applets.
- En connexion directe, les classes MQSeries pour Java ne prennent pas en charge les applets. Par conséquent, si vous voulez que vos programmes soient utilisables dans tous les modes de connexion, y compris la connexion directe, écrivez des servlets ou des applications.

Différences entre modes de connexion

La façon de programmer avec les classes MQSeries pour Java dépend en partie des modes de connexion que vous souhaitez utiliser.

Connexions client

Lorsque le module classes MQSeries pour Java est utilisé comme client, il est analogue au client MQSeries C, avec cependant les différences suivantes :

- Il ne prend en charge que TCP/IP.
- Il ne prend pas en charge les tables de connexions.
- Il ne lit aucune variable d'environnement MQSeries au démarrage.
- Les informations conservées normalement dans une définition de canal et dans des variables d'environnement sont stockées dans une classe nommée MQEnvironment, ou peuvent être transmises en tant qu'arguments au moment de l'établissement de la connexion.

- Les erreurs et les conditions d'exception sont consignées dans un journal spécifié via la classe MQException. Par défaut, les erreurs sont transmises à la console Java.

Les clients des classes MQSeries pour Java ne prennent en charge ni l'instruction MQBEGIN ni les connexions directes rapides.

Pour plus de détails sur les clients MQSeries, reportez-vous au manuel *MQSeries - Clients*.

Remarque : Lorsque vous utilisez la connexion VisiBroker, les valeurs d'ID utilisateur et de mot de passe de MQEnvironment ne sont pas transmises au serveur MQSeries. L'ID utilisateur effectif est celui qui s'applique au serveur IIOP.

Connexions directes

Le mode connexion directe de classes MQSeries pour Java diffère des modes client par les aspects suivants :

- La plupart des paramètres fournis par la classe MQEnvironment sont ignorés.
- Le mode connexion directe prend en charge l'instruction MQBEGIN et les liens rapides avec le gestionnaire de files d'attente MQSeries.

Remarque : MQSeries for AS/400 ne prend pas en charge l'utilisation de l'instruction MQBEGIN pour démarrer des unités de travail globales coordonnées par le gestionnaire de files d'attente.

Définition de la connexion à utiliser

La connexion à utiliser est définie par l'intermédiaire de variables de la classe MQEnvironment.

MQEnvironment.properties

Cette variable contient les paires clé/valeur suivantes :

- Pour les connexions client et directes :

  ```
  MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES
  ```

- Pour les connexions VisiBroker :

  ```
  MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_VISIBROKER
  MQC.ORB_PROPERTY, orb
  ```

MQEnvironment.hostname

Définissez la valeur de cette variable de la façon suivante :

- Pour les connexions client, affectez-lui le nom d'hôte du serveur MQSeries auquel vous voulez vous connecter.
- Pour les connexions directes, affectez-lui la valeur null.

Exemples partiels de code

La présente section contient deux exemples partiels de code, la figure 1 à la page 55 et la figure 2 à la page 58. Chacun d'entre eux est conçu pour utiliser une connexion donnée et comporte des remarques indiquant les modifications à apporter au code pour utiliser un autre mode de connexion.

Exemple de code d'applet

L'extrait de code présenté ci-après montre comment une applet utilisant une connexion TCP/IP peut :

1. se connecter à un gestionnaire de files d'attente,
2. insérer un message dans la file SYSTEM.DEFAULT.LOCAL.QUEUE,
3. extraire le message en retour.

```
// ============================================================================
//
// Eléments sous licence - Propriété d'IBM
//
// 5639-C34
//
// (c) Copyright IBM Corp. 1995,1999
//
// ============================================================================
// Cet exemple s'exécute sous forme d'applet via l'afficheur d'applets
// et le fichier HTML, à partir de la commande suivante :
//            appletviewer MQSample.html
// Les sorties s'effectuent sur la ligne de commande et NON
// dans la fenêtre de l'afficheur d'applets.
//
// Remarque : Si vous recevez des erreurs MQSeries 'error 2 reason 2059'
// et si vous êtes sûr par ailleurs que votre configuration de MQSeries
// et TCP/IP est bonne,cliquez sur l'option "Applet" de la fenêtre de l'afficheur,
// sélectionnez "Properties", puis affectez à l'option "Network access"
// la valeur "Unrestricted".
import com.ibm.mq.*;            // Inclure le module classes MQSeries pour Java

public class MQSample extends java.applet.Applet
{
  private String hostname = "votre_serveur";      // Définit le nom de
                                                  // l'hôte auquel on se
                                                  // connecte.
  private String channel  = "canal_serveur";      // Définit le nom du canal
                                                  // utilisé par le client.
                                                  // Remarque : On suppose que le serveur
                                                  // écoute sur le port TCP/IP
                                                  // par défaut 1414.
  private String qManager = "votre_gestionnaire"; // Définit le nom de l'objet
                                                  // gestionnaire de files
                                                  // cible.

  private MQQueueManager qMgr;                    // Définit un objet gestionnaire.

  // Lors de l'appel de la classe, cette initialisation est effectuée.

  public void init()
  {
     // Configuration de l'environnement MQSeries
     MQEnvironment.hostname = hostname;           // On aurait pu coder le
                                                  // nom d'hôte et le canal
     MQEnvironment.channel  = channel;            // directement ici !

     MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY,  //Définit la connexion
                                  MQC.TRANSPORT_MQSERIES); //serveur ou TCP/IP.

  } // Fin de Init
```

Figure 1. Exemple d'applet classes MQSeries pour Java (Numéro 1 de 3)

Exemple de code

```
public void start()
{
  try {
    // Création d'une connexion au gestionnaire de files d'attente
    qMgr = new MQQueueManager(qManager);

    // Configuration des options de la file d'attente à ouvrir...
    // Remarque : Toutes les options MQSeries ont le préfixe MQC en Java.
     int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                       MQC.MQOO_OUTPUT ;
    // Spécification de la file à ouvrir et des options d'ouverture...

    MQQueue system_default_local_queue =
            qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                             openOptions);

  // Définition d'un message MQSeries simple et écriture
  // d'un texte au format UTF..

    MQMessage hello_world = new MQMessage();
    hello_world.writeUTF("Bonjour tout le monde");

    // Définition des options de message...

    MQPutMessageOptions pmo = new MQPutMessageOptions();  // Options par défaut,
                                                          // équivalentes à
                                                          // MQPMO_DEFAULT

    // Placement du message en file d'attente

    system_default_local_queue.put(hello_world,pmo);

    // Extraction du message...
    // Définir d'abord une mémoire tampon de message MQSeries pour
    // recevoir les données du message...

    MQMessage retrievedMessage = new MQMessage();
    retrievedMessage.messageId = hello_world.messageId;

    // Définition des options d'extraction de message...

    MQGetMessageOptions gmo = new MQGetMessageOptions();  // Options par défaut
                                                          // équivalentes à
                                                          // MQGMO_DEFAULT
    // Extraction du message de la file d'attente...

    system_default_local_queue.get(retrievedMessage, gmo);

    // Et vérification qu'il s'agit bien de notre message par
    // l'affichage du texte UTF

    String msgText = retrievedMessage.readUTF();
    System.out.println("Texte du message : " + msgText);

    // Fermeture de la file d'attente

    system_default_local_queue.close();

    // Déconnexion du gestionnaire de files d'attente

    qMgr.disconnect();
  }
  // Si une erreur s'est produite au cours du traitement ci-dessus,
  // tenter de l'identifier.
  // S'agit-il d'une erreur MQSeries ?
```

Figure 1. Exemple d'applet classes MQSeries pour Java (Numéro 2 de 3)

```
    catch (MQException ex)
    {
      System.out.println("Erreur MQSeries : Code achèvement " +
                         ex.completionCode +
                         " Code raison " + ex.reasonCode);
    }
    // S'agissait-il d'une erreur de l'espace tampon de Java ?
    catch (java.io.IOException ex)
    {
      System.out.println("Erreur d'écriture dans le
      tampon de message : " + ex);
    }

  } // Fin de Start

} // Fin de l'exemple
```

Figure 1. Exemple d'applet classes MQSeries pour Java (Numéro 3 de 3)

Changement de la connexion afin d'utiliser VisiBroker pour Java

Remplacez la ligne

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);
```

par :

```
MQEnvironment.properties.put (MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_VISIBROKER);
```

et ajoutez les lignes suivantes pour initialiser l'ORB (Object Request Broker) :

```
ORB orb=ORB.init(this,null);
MQEnvironment.properties.put(MQC.ORB_PROPERTY,orb);
```

Vous devez également ajouter l'instruction d'importation suivante en tête du fichier :

```
import org.omg.CORBA.ORB;
```

Lorsque vous utilisez VisiBroker, vous n'avez pas besoin de spécifier de numéro de port ou de canal.

Exemple de code d'application

L'extrait de code ci-après montre comment écrire une application simple utilisant le mode connexion directe pour :

1. se connecter à un gestionnaire de files d'attente,
2. insérer un message dans la file SYSTEM.DEFAULT.LOCAL.QUEUE,
3. extraire le message en retour.

Exemple de code

```
// ====================================================================
// Eléments sous licence - Propriété d'IBM
// 5639-C34
// (c) Copyright IBM Corp. 1995, 1999
// ====================================================================
// Classes MQSeries pour exemple d'application Java
//
// Cet exemple s'exécute sous forme d'application Java via la
// commande :- java MQSample

import com.ibm.mq.*;            // Inclure le module de classes MQSeries pour Java

public class MQSample
{
  private String qManager = "votre_gestionnaire"; // Définit le nom de l'objet
                                                  // gestionnaire cible.
  private MQQueueManager qMgr;                    // Définit un objet
                                                  // gestionnaire.
  public static void main(String args[]) {
     new MQSample();
  }

  public MQSample() {
try {

      // Création d'une connexion au gestionnaire de files d'attente

      qMgr = new MQQueueManager(qManager);

      // Configuration des options de la file d'attente à ouvrir...
      // Remarque : Toutes les options MQSeries ont le préfixe MQC en Java.

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;

      // Indiquer à présent la file d'attente à ouvrir
      // ainsi que les options d'ouverture...

      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);

    // Définition d'un message MQSeries simple et écriture
    // d'un texte au format UTF..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Bonjour tout le monde");

      // Définition des options de message...

      MQPutMessageOptions pmo = new MQPutMessageOptions();  // Options par défaut
                                                           // équivalentes à
                                                            // MQPMO_DEFAULT.
```

Figure 2. Exemple d'application classes MQSeries pour Java (Numéro 1 de 2)

```
     // Placement du message en file d'attente

      system_default_local_queue.put(hello_world,pmo);

      // Extraction du message...
      // Définir d'abord une mémoire tampon de message MQSeries pour
      // recevoir les données du message...

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // Définition des options d'extraction de message

      MQGetMessageOptions gmo = new MQGetMessageOptions();  // Options par défaut
                                                            // équivalentes à
                                                            // MQGMO_DEFAULT.
      // Récupératon du message dans la file d'attente...

      system_default_local_queue.get(retrievedMessage, gmo);

      // Et vérification qu'il s'agit bien de notre message par
      // l'affichage du texte UTF

      String msgText = retrievedMessage.readUTF();
      System.out.println("Texte du message : " + msgText);
      // Fermeture de la file d'attente...
      system_default_local_queue.close();
      // Déconnexion du gestionnaire de files d'attente

      qMgr.disconnect();
    }
      // Si une erreur s'est produite au cours du traitement ci-dessus,
    // tenter de l'identifier.
    // S'agissait-il d'une erreur MQSeries ?
    catch (MQException ex)
    {
      System.out.println("Erreur MQSeries : Code achèvement " +
                         ex.completionCode + " Code raison " + ex.reasonCode);
    }
      // S'agissait-il d'une erreur de l'espace de mémoire tampon de Java ?
    catch (java.io.IOException ex)
    {
      System.out.println("Erreur d'écriture dans le tampon de message : " +ex);
    }
  }
} // Fin de l'exemple
```

Figure 2. Exemple d'application classes MQSeries pour Java (Numéro 2 de 2)

Communication avec les gestionnaires de files d'attente

La présente section explique comment se connecter à un gestionnaire de files d'attente et s'en déconnecter à l'aide de classes MQSeries pour Java.

Configuration de l'environnement MQSeries

Remarque : Cette étape n'est pas nécessaire si vous utilisez les classes MQSeries pour Java en connexion directe. Si c'est votre cas, passez directement à la section «Connexion à un gestionnaire de files d'attente» à la page 60. Pour pouvoir vous connecter à un gestionnaire de files d'attente via une connexion client, vous devez d'abord configurer un objet MQEnvironment.

Communication avec les gestionnaires de files d'attente

Les clients MQSeries écrits en C s'appuient sur des variables d'environnement pour définir les caractéristiques de l'appel MQCONN. Comme les applets Java n'ont pas accès aux variables d'environnement, l'interface de programmation Java comprend une classe MQEnvironment. Cette classe permet de définir les valeurs suivantes, utilisées pour établir la connexion :

- Nom de canal
- Nom d'hôte
- Numéro de port
- ID utilisateur
- Mot de passe

Pour spécifier le nom de canal et le nom d'hôte, utilisez le codesuivant :

```
MQEnvironment.hostname = "hôte.domaine.com";
MQEnvironment.channel  = "canal.client.java";
```

Ces deux lignes sont équivalentes à la définition suivante d'une variable d'environnement MQSERVER :

```
"canal.client.java/TCP/hôte.domaine.com".
```

Par défaut, les clients Java tentent de se connecter à un programme d'écoute MQSeries sur le port 1414. Pour indiquer un autre port, utilisez le codesuivant :

```
MQEnvironment.port = nnnn;
```

Par défaut, l'ID utilisateur et le mot de passe sont à blanc. Pour leur attribuer des valeurs, utilisez le code suivant :

```
MQEnvironment.userID   = "id1";  // Equivalent à la var. d'env. MQ_USER_ID
MQEnvironment.password = "mdp";  // Equivalent à la var. d'env. MQ_USER_PASSWORD
```

Remarque : Si vous configurez une connexion utilisant VisiBroker pour Java, reportez-vous à la section «Changement de la connexion afin d'utiliser VisiBroker pour Java» à la page 57.

Connexion à un gestionnaire de files d'attente

Vous êtes désormais prêt à établir une connexion à un gestionnaire de files d'attente en créant une nouvelle instance de la classe MQQueueManager :

```
MQQueueManager queueManager = new MQQueueManager("NomGest");
```

Pour vous déconnecter d'un gestionnaire de files d'attente, appelez la méthode disconnect() du gestionnaire :

```
queueManager.disconnect();
```

L'appel de la méthode disconnect entraîne la fermeture de toutes les files d'attente et de tous les processus ouverts auxquels vous avez accédé via ce gestionnaire. Cependant, il est recommandé de fermer explicitement ces ressources lorsqu'elles ne sont plus nécessaires. Pour ce faire, utilisez la méthode close().

Les méthodes commit() et backout() correspondent aux appels MQCMIT et MQBACK utilisés avec l'interface procédurale.

Accès aux files d'attente et aux processus

La classe MQQueueManager permet d'accéder aux files d'attente et aux processus. La structure de MQOD (descripteur d'objet) est décomposée en arguments des méthodes de cette classe. Pour ouvrir, par exemple, une file d'attente (FA) d'un gestionnaire nommé «NomGest», utilisez le code suivant :

```
MQQueue queue = queueManager.accessQueue("NomFA",
                                         MQC.MQOO_OUTPUT,
                                         "NomGest",
                                         "NomFAdynamique",
                                         "IDutilSecondaire");
```

L'argument *options* (MQOO_OUTPUT) est le même que celui de l'appel MQOPEN.

La méthode accessQueue renvoie un nouvel objet de la classe MQQueue.

Lorsque vous avez fini d'utiliser la file d'attente, fermez-la à l'aide de la méthode close(), comme suit :

```
queue.close();
```

Avec les classes MQSeries pour Java, vous pouvez également créer une file d'attente à l'aide du constructeur MQQueue. Les arguments sont les mêmes que ceux de la méthode accessQueue, auxquels s'ajoute un argument de gestionnaire de files d'attente. Par exemple :

```
MQQueue queue = new MQQueue(queueManager,
                            "NomFA",
                            MQC.MQOO_OUTPUT,
                            "NomGest",
                            "NomFAdynamique",
                            "IDutilSecondaire");
```

En construisant un objet file d'attente de cette manière, vous avez la possibilité d'écrire vos propres sous-classes de la classe MQQueue.

Pour accéder à un processus, utilisez la méthode accessProcess au lieu de la méthode accessQueue. Cette méthode n'utilise pas d'argument *dynamic queue name* (nom de file d'attente dynamique) puisque cet argument ne s'applique pas aux processus.

La méthode accessProcess renvoie un nouvel objet de la classe MQProcess.

Lorsque vous avez fini d'utiliser l'objet processus, fermez-le à l'aide de la méthode close(), comme suit :

```
process.close();
```

Avec les classes MQSeries pour Java, vous pouvez également créer un processus à l'aide du constructeur MQProcess. Les arguments sont les mêmes que ceux de la méthode accessProcess, auxquels s'ajoute un argument de gestionnaire de files d'attente. En construisant un objet processus de cette manière, vous avez la possibilité d'écrire vos propres sous-classes de la classe MQProcess.

Traitement des messages

Pour insérer des messages dans une file d'attente, vous devez utiliser la méthode put() de la classe MQQueue. Pour extraire des messages qui se trouvent dans une file d'attente, vous devez utiliser la méthode get() de la classe MQQueue. A la différence de l'interface procédurale, dans laquelle MQPUT et MQGET écrivent et

lisent des tableaux d'octets, le langage Java écrit et lit des instances de la classe MQMessage. La classe MQMessage encapsule le tampon de données qui contient les données du message et les arguments MQMD qui décrivent ce message.

Pour construire un nouveau message, créez une nouvelle instance de la classe MQMessage, puis utilisez les méthodes writeXXX pour placer les données dans le tampon de message.

Lorsque la nouvelle instance de message est créée, tous les arguments MQMD prennent automatiquement leurs valeurs par défaut, précisées dans le manuel *MQSeries Application Programming Reference*. La méthode put() de MQQueue prend également comme argument une instance de la classe MQPutMessageOptions. Cette classe représente la structure MQPMO. L'exemple suivant permet de créer un message et de le placer en file d'attente :

```
// Création d'un message contenant mon âge suivi de mon nom
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Jérémie Marchand";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Utilisation des options par défaut d'insertion de message...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// Insertion du 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. Elle aussi prend comme argument une instance de la classe MQGetMessageOptions. Cette classe représente la structure MQGMO.

Il n'est pas nécessaire d'indiquer une taille maximale de message, car la méthode get() adapte automatiquement la taille de son tampon interne à celle du message entrant. Utilisez les méthodes readXXX de la classe MQMessage pour accéder aux données du message renvoyé.

L'exemple suivant illustre comment extraire un message d'une file d'attente :

```
// Extraction d'un message de la file d'attente
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // Valeurs par défaut

// Extraction des données du message
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

Le format de nombres utilisé par les méthodes read et write peut être modifié ; pour cela, définissez la variable membre *encoding*.

Vous pouvez aussi modifier le jeu de caractères utilisé pour lire et écrire les chaînes de caractères en définissant la variable membre *characterSet*.

Pour plus de détails, reportez-vous à la section «MQMessage» à la page 108.

Remarque : Lorsque vous utilisez la méthode writeUTF de MQMessage, la longueur de la chaîne ainsi que les mots Unicode qu'elle contient sont automatiquement définis. Lorsque votre message doit être lu par un autre programme Java (via readUTF()), c'est la manière la plus simple d'envoyer des données de type chaîne.

Traitement des erreurs

Les méthodes de l'interface Java ne renvoient pas de code d'achèvement ni de code raison. En revanche, elles déclenchent une exception lorsque le code achèvement et le code raison résultant d'un appel MQSeries ne sont pas tous deux égaux à zéro. Cela simplifie la logique du programme, puisque vous n'avez pas besoin de contrôler vous-mêmes ces deux codes à la suite de chaque appel à MQSeries. Vous pouvez décider de l'endroit, dans le programme, à partir duquel vous traiterez un échec éventuel. Pour définir cet endroit, placez le code entre des blocs 'try' et 'catch', comme l'illustre l'exemple suivant :

```
try {
myQueue.put(messageA,putMessageOptionsA);
myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
// Ce bloc ne sera exécuté que si l'une des deux méthodes put
// a généré un code achèvement ou un code raison
// différent de zéro.
System.out.println("Erreur lors de l'opération put :" +
                           "Code achèvement = " + ex.completionCode +
                           "Code raison = " + ex.reasonCode);
}
```

Extraction et définition des valeurs d'attributs

Dans le cas de nombreux attributs courants, les classes MQManagedObject, MQQueue, MQProcess et MQQueueManager possèdent des méthodes getXXX() et setXXX() qui vous permettent de lire et de définir la valeur de ces attributs. Notez que pour MQQueue, ces méthodes ne fonctionnent que si vous spécifiez les indicateurs «inquire» (interrogation de valeurs) et «set» (définition de valeurs) appropriés au moment de l'ouverture de la file d'attente.

Dans le cas des attributs moins courants, les classes MQQueueManager, MQQueue et MQProcess héritent toutes d'une classe nommée MQManagedObject. Cette classe définit les interfaces inquire() et set().

Lorsque vous créez un nouvel objet gestionnaire de files d'attente à l'aide de l'opérateur *new*, il est automatiquement ouvert en mode «inquiry». Lorsque vous utilisez la méthode accessProcess() pour accéder à un objet processus, celui-ci est automatiquement ouvert en mode 'inquiry'. En revanche, lorsque vous accédez à un objet file d'attente à l'aide de la méthode accessQueue(), il n'est ouvert automatiquement *ni* pour les opérations «inquire», *ni* pour les opérations «set». En effet, l'ajout de ces options risque de générer des incidents liés à certains types de files d'attente éloignées. Pour utiliser les méthodes inquire, set, getXXX et setXXX sur une file d'attente, vous devez spécifier les indicateurs «inquire» et «set» appropriés dans l'argument openOptions de la méthode accessQueue().

Les méthodes inquire et set prennent trois arguments :
- un tableau de sélecteurs (selectors),
- un tableau d'attributs entiers (intAttrs),
- un tableau d'attributs caractères (charAttrs).

Utilisation des valeurs d'attributs

Les arguments SelectorCount, IntAttrCount et CharAttrLength retrouvés dans MQINQ sont inutiles car la longueur d'un tableau est toujours connue dans Java. Voici un exemple d'interrogation (inquiry) de file d'attente :

```
// Interrogation d'une file d'attente
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("Priorité par défaut = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Programmes à unités d'exécution multiples

Il est difficile de ne pas écrire des programmes à unités d'exécution multiples en Java. Prenons, par exemple, un simple programme se connectant à un gestionnaire de files d'attente et ouvrant une file d'attente au démarrage. Le programme présente un seul bouton à l'utilisateur, et lorsque ce dernier clique sur le bouton, le programme va chercher un message dans la file d'attente.

Par nature, l'environnement d'exécution de Java fonctionne avec des unités d'exécution multiples. Aussi, l'initialisation du programme s'effectue sur une unité et le code exécuté en réponse à l'activation du bouton s'exécute sur une autre unité (l'unité d'exécution de l'interface utilisateur).

Avec le client MQSeries en C, cela poserait un problème, car les identificateurs (*handles*) ne peuvent être partagés entre différentes unités d'exécution. Le module classes MQSeries pour Java réduit cette contrainte en autorisant le partage d'un objet gestionnaire de files d'attente (et de ses objets files d'attente et processus associés) entre plusieurs unités d'exécution.

La mise en oeuvre de classes MQSeries pour Java garantit que, pour une connexion donnée (instance d'objet gestionnaire de files d'attente - MQQueueManager), tous les accès au gestionnaire de files d'attente MQSeries cible sont synchronisés. En d'autres termes, si une unité d'exécution doit envoyer un appel à un gestionnaire de files d'attente, cet appel est bloqué jusqu'à ce que tous les appels en cours pour cette connexion soient terminés. Si vous avez besoin d'accéder au même gestionnaire de files d'attente à partir de différentes unités d'exécution du programme, créez un nouvel objet MQQueueManager pour chaque unité d'exécution requérant un accès simultané. (C'est l'équivalent de l'émission d'un appel MQCONN distinct pour chaque unité d'exécution.)

Remarque : Dans l'environnement CICS Transaction Server pour OS/390, seule l'unité d'exécution principale (la première) peut émettre des appels CICS ou MQSeries. Il est donc impossible de partager des objets MQQueueManager ou MQQueue entre unités d'exécution dans un tel environnement, ni même de créer un objet MQQueueManager sur une unité d'exécution fille.

Exits utilisateur

Le module classes MQSeries pour Java vous permet de créer vos propres exits d'envoi, de réception et de sécurité.

Pour mettre en œuvre un exit, vous devez définir une nouvelle classe Java implémentant l'interface appropriée. Trois interfaces de sortie sont définies dans le module MQSeries :
- MQSendExit
- MQReceiveExit
- MQSecurityExit

L'exemple suivant illustre une classe mettant en œuvre ces trois interfaces :

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // Cette méthode provient de l'interface SendExit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // Placez ici le corps de votre exit d'envoi
  }


  // Cette méthode provient de l'interface ReceiveExit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
     // Placez ici le corps de votre exit de réception
  }


  // Cette méthode provient de l'interface SecurityExit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
     // Placez ici le corps de votre exit de sécurité
  }

}
```

A chaque exit sont passées une instance d'objet MQXChannelExit et une instance d'objet MQChannelDefinition. Ces objets représentent les structures MQCXP et MQCD de l'interface procédurale.

Pour un exit d'envoi, l'argument *agentBuffer* contient les données qui vont être envoyées. Pour un exit de réception ou de sécurité, cet argument contient les données reçues. Il n'y a pas lieu d'inclure un argument de longueur, car l'expression agentBuffer.length vous renvoie la taille du tableau.

Dans le cas des exits d'envoi et de sécurité, votre code doit renvoyer le tableau d'octets à envoyer au serveur. Dans le cas d'un exit de réception, le code doit renvoyer les données modifiées que vous souhaitez voir interprétées par les classes MQSeries pour Java.

L'exit le plus simple est le suivant :

```
{
  return agentBuffer;
}
```

Si votre programme doit être exécuté sous forme d'appletJava téléchargée, notez que les restrictions de sécurité qui lui seront appliquées l'empêcheront de lire ou d'écrire des fichiers en local. Si votre exit a besoin d'un fichier de configuration, vous pouvez placer ce fichier sur le Web et recourir à la classe java.net.URL pour le télécharger et examiner son contenu.

Définition d'un pool de connexion

Le module classes MQSeries pour Java version 5.2 prend en charge les applications pouvant traiter plusieurs connexions à des gestionnaires de files d'attente. Lorsqu'une connexion devient inutile, elle n'est pas détruite. Elle peut être insérée dans un pool pour une utilisation ultérieure. Ainsi, les performances des applications et des logiciels intermédiaires qui se connectent à des gestionnaires de files d'attente arbitraires s'en trouvent améliorées.

MQSeries fournit un pool de connexion par défaut. Les applications peuvent activer ou désactiver ce pool en enregistrant ou en annulant l'enregistrement de jetons via la classe MQEnvironment. Si le pool est actif, lorsque MQ base Java construit un objet MQQueueManager, le pool par défaut est recherché afin d'utiliser une éventuelle connexion adaptée. Lorsqu'un appel MQQueueManager.disconnect() se produit, la connexion sous-jacente est renvoyée au pool.

Les applications peuvent également construire un pool de connexion MQSimpleConnectionManager pour un usage spécifique. Une application peut alors spécifier ce pool pendant la construction d'un objet MQQueueManager, ou transmettre le pool au MQEnvironment pour une utilisation en tant que pool de connexion par défaut.

Enfin, MQ base Java permet une mise en oeuvre partielle de l'architecture de connecteur J2EE (Java 2 Platform Enterprise Edition). Les applications fonctionnant sous Java 2 v1.3 JVM avec JAAS 1.0 (Java Authentication and Authorization Service) peuvent fournir leur propre pool de connexion en mettant en oeuvre l'interface **javax.resource.spi.ConnectionManager**. Cette interface peut être spécifiée sur le constructeur MQQueueManager ou comme pool de connexion par défaut.

Contrôle du pool de connexion par défaut

Soit l'application exemple, MQApp1, suivante :

```
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 extrait une liste de gestionnaires de files d'attente locaux de la ligne de commande, se connecte à chacun d'entre eux et effectue des opérations. Toutefois,

lorsque la ligne de commande répertorie un même gestionnaire de files d'attente plusieurs fois, il est plus efficace de se connecter une seule fois et d'utiliser cette connexion plusieurs fois.

MQ base Java dispose d'un pool de connexion par défaut qui permet d'effectuercette opération. Pour activer ce pool, utilisez une des méthodes MQEnvironment.addConnectionPoolToken(). Pour le désactiver, utilisez MQEnvironment.removeConnectionPoolToken().

L'application exemple suivante, MQApp2, est identique en termes de fonctionnement à MQApp1, mais elle ne se connecte qu'une 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 connexion par défaut en enregistrant un objet MQPoolToken avec MQEnvironment.

Le constructeur MQQueueManager recherche ce pool pour établir une connexion appropriée et il ne crée une connexion au gestionnaire de files d'attente que s'il n'en détecte aucune existante. L'appel qmgr.disconnect() renvoie la connexion au pool pour permettre son utilisation ultérieure. Ces appels d'API sont identiques à ceux de l'application exemple, MQApp1.

La seconde ligne mise en évidence désactive le pool de connexion par défaut, ce qui annule les connexions aux gestionnaires de files d'attente stockées dans le pool. Cette opération est importante car si elle ne se produisait pas, l'application se terminerait avec un trop grand nombre de connexions aux gestionnaires de files d'attente dans le pool. Une telle situation générerait des erreurs consignées dans les journaux d'erreurs des gestionnaires de files d'attente.

Le pool de connexion par défaut n'enregistre pas plus de dix connexions inutilisées, et il les maintient actives pendant cinq minutes, maximum. L'application peut intervenir sur ce comportement (pour plus de détails, reportez-vous à la section «Définition d'un nouveau pool de connexion» à la page 69).

Au lieu d'utiliser MQEnvironment pour fournir un MQPoolToken, l'application peut le construire :

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Certaines applications ou certains fournisseurs de logiciel intermédiaire peuvent fournir des sous-classes de MQPoolToken pour transmettre les données à un pool de connexion personnalisé. Elles peuvent être construites et transmisesà addConnectionPoolToken() de cette manière, de sorte que des informations supplémentaires puissent être transmises au pool de connexion.

Pool de connexion par défaut et composants multiples

MQEnvironment dispose d'un ensemble statique d'objets MQPoolToken enregistrés. Pour ajouter ou supprimer des objets MQPoolTokens de cet ensemble, procédez comme suit :

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

Une application peut être constituée de nombreux composants indépendants et effectuer un travail à l'aide d'un gestionnaire de files d'attente. Dans ce cas, chaque composant doit ajouter un objet MQPoolToken dans l'ensemble MQEnvironment, défini pour sa durée de vie.

Supposons que l'application exemple, MQApp3, crée dix unités d'exécution et qu'elle les démarre toutes. Chaque unité enregistre son objet MQPoolToken, patiente un certain temps, puis se connecte au gestionnaire de files d'attente. Une fois que l'unité d'exécution s'est déconnectée, elle retire son objet MQPoolToken.

Le pool de connexion par défaut reste actif tant que l'ensemble d'objets MQPoolToken contient au moins un jeton, par conséquent, il demeure actif pendant toute la durée de l'application. Cette dernière n'exige pas la conservation d'un objet maître dans le contrôle général 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);
        }
    }
```

Définition d'un nouveau pool de connexion

La présente section décrit comment utiliser la classe **com.ibm.mq.MQSimpleConnectionManager** pour fournir un autre pool de connexion. Cette classe offre les fonctions de base des pools de connexion et les applications peuvent l'utiliser pour personnaliser le comportement d'un pool.

Une fois démarré, un objet MQSimpleConnectionManager peut être défini dans le constructeur MQQueueManager. MQSimpleConnectionManager gère alors la connexion sous-jacente de l'objet MQQueueManager construit. Si MQSimpleConnectionManager dispose d'une connexion appropriée dans un pool, celle-ci sera réutilisée et renvoyée au MQSimpleConnectionManager après un appel MQQueueManager.disconnect().

Le fragment de code suivant illustre un tel 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 définie lors du premier constructeur MQQueueManager est stockée dans l'objet myConnMan après l'appel qmgr.disconnect(). Elle est ensuite réutilisée pendant le second appel émis vers le constructeur MQQueueManager.

La deuxième ligne active l'objet MQSimpleConnectionManager. La dernière ligne désactive MQSimpleConnectionManager, en annulant les connexions contenues dans le pool. Le mode par défaut d'un MQSimpleConnectionManager est MODE_AUTO, décrit plus loin dans cette section.

Le critère d'attribution des connexions par l'objet MQSimpleConnectionManager repose sur la notion de dernière connexion utilisée et MQSimpleConnectionManager annule les connexions selon la notion d'utilisation la plus ancienne. Par défaut, une connexion est annulée si elle n'a pas été utilisée pendant cinq minutes, ou si plus de dix connexions inutilisées sont enregistrées dans le pool. Ces valeurs peuvent être modifiées à l'aide des méthodes suivantes :

- MQSimpleConnectionManager.setTimeout()
- MQSimpleConnectionManager.setHighThreshold()

Vous pouvez également définir un objet MQSimpleConnectionManager à utiliser comme pool de connexion par défaut, lorsqu'aucun gestionnaire de connexion n'est fourni par le constructeur MQQueueManager.

L'application suivante illustre cette opération :

```
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.setHighThreshold(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

Les lignes inscrites en caractères gras permettent de définir un objet MQSimpleConnectionManager avec les caractéristiques suivantes :

- annulation des connexions inutilisées depuis une heure
- limitation du nombre de connexions inutilisées et conservées dans le pool à 50
- MODE_AUTO (valeur par défaut). Cela signifie que le pool n'est actif que s'il s'agit du gestionnaire de connexion par défaut et s'il y a au moins un jeton dans l'ensemble d'objets MQPoolToken contenu dans MQEnvironment.

MQSimpleConnectionManager est alors défini en tant que gestionnaire de connexion par défaut.

Dans la dernière ligne, l'application appelle la méthode MQApp3.main(). Elle permet d'exécuter un certain nombre d'unités d'exécution, chacune utilisant MQSeries indépendemment. Ces unités utilisent myConnMan dès qu'elles établissent des connexions.

Définition d'un ConnectionManager personnalisé

Sous Java 2 v1.3 avec JAAS 1.0 installé, les applications et les fournisseurs de logiciel intermédiaire peuvent fournir différentes mises en oeuvre de pools de connexion. MQ base Java offre une mise en oeuvre partielle de l'architecture de connecteur J2EE. Les mises en oeuvre de **javax.resource.spi.ConnectionManager** peuvent être utilisées comme gestionnaire de connexion par défaut ou définies dans le constructeur MQQueueManager.

MQ base Java est conforme au contrat du gestionnaire de connexion de l'architecture de connecteur J2EE. Prenez connaissance de la présente section et du contrat du gestionnaire de connexion de l'architecture de connecteur J2EE (accédez au site Web de Sun, à l'adresse `http://java.sun.com`).

L'interface ConnectionManager ne définit que la méthode suivante :

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

Le constructeur MQQueueManager appelle allocateConnection sur le gestionnaire ConnectionManager approprié. Il transmet les mises en oeuvre correspondantes de ManagedConnectionFactory et ConnectionRequestInfo sous la forme d'arguments pour décrire la connexion requise.

ConnectionManager recherche dans son pool un objet javax.resource.spi.ManagedConnection créé à l'aide d'objets

ManagedConnectionFactory et ConnectionRequestInfo identiques. Si ConnectionManager localise des objets ManagedConnection adaptés, il crée un ensemble java.util.Set contenant les objets ManagedConnections candidats. Ensuite ConnectionManager appelle les éléments suivants :

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

La mise en oeuvre MQSeries de l'objet ManagedConnectionFactory ignore l'argument objet. Cette méthode sélectionne et renvoie un objet ManagedConnection adapté, extrait de l'ensemble, ou renvoie la valeur null si elle n'en localise aucun. S'il n'existe pas d'objet ManagedConnection approprié dans le pool, ConnectionManager peut en créer un à l'aide de la commande suivante :

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

L'argument objet est de nouveau ignoré. Cette méthode permet d'établir une connexion à un gestionnaire de files d'attente MQSeries et de renvoyer une mise en oeuvre de l'objet javax.resource.spi.ManagedConnection représentant la nouvelle connexion. Une fois que l'objet ConnectionManager a obtenu un objet ManagedConnection (extrait du pool ou créé de toutes pièces), il crée un indicateur de connexion à l'aide de la commande suivante :

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

Cet indicateur de connexion peut être renvoyé par allocateConnection().

ConnectionManager doit enregistrer un intérêt dans ManagedConnection via l'élément suivant :

```
mc.addConnectionEventListener()
```

Si la connexion est soumise à une erreur grave ou lorsque MQQueueManager.disconnect() est appelé, ConnectionEventListener reçoit un avertissement. Lorsque MQQueueManager.disconnect() est appelé, ConnectionEventListener peut agir de l'une des manières suivantes :

- restaurer ManagedConnection à l'aide de l'appel mc.cleanup(), puis renvoyer ManagedConnection dans le pool
- annuler ManagedConnection à l'aide de l'appel mc.destroy()

Si ConnectionManager est défini comme gestionnaire par défaut, il peut également enregistrer un intérêt dans l'ensemble d'objets MQPoolToken géré par MQEnvironment. Pour ce faire, construisez d'abord un objet MQPoolServices, puis enregistrez un objet MQPoolServicesEventListener avec l'objet MQPoolServices :

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

Le programme d'écoute reçoit un avertissement lorsqu'un objet MQPoolToken est ajouté ou supprimé de l'ensemble ou lorsque l'objet ConnectionManager par défaut est modifié. L'objet MQPoolServices fournit également un moyen de demander la taille de l'ensemble d'objets MQPoolToken.

Compilation et test de programmes MQ base Java

Avant de compiler des programmes MQ base Java, assurez-vous que le répertoire d'installation des classes MQSeries pour Java figure bien dans la variable d'environnement CLASSPATH, comme l'explique le «Chapitre 2. Procédures d'installation» à la page 9.

Compilation et test de programmes MQ base Java

Pour compiler la classe nommée ″MaClasse.java″, utilisez la commande suivante :

```
javac MaClasse.java
```

Exécution d'applets MQ base Java

Si vous écrivez une applet (sous-classe de java.applet.Applet), vous devez créer un fichier HTML référençant votre classe pour pouvoir l'exécuter. Voici un exemple de fichier HTML d'applet :

```
<html>
<body>
<applet code="MaClasse.class" width=200 height=400>
</applet>
</body>
</html>
```

Exécutez une applet en chargeant ce fichier HTML dans un navigateur Web compatible Java, ou en utilisant l'afficheur d'applets livré avec le JDK (Java Development Kit).

Pour utiliser l'afficheur d'applets, entrez la commande suivante :

```
appletviewer maclasse.html
```

Exécution des applications MQ base Java

Si vous écrivez une application (une classe contenant une méthode main()) et que vous utilisez une connexion client ou directe, exécutez votre programme via l'interpréteur Java. Utilisez la commande suivante :

```
java MaClasse
```

Remarque : L'extension ″.class″ est omise dans le nom de la classe.

Exécution d'applications MQ base Java sous CICS Transaction Server pour OS/390

Pour exécuter une application Java en tant que transaction CICS, vous devez procéder comme suit :

1. Définissez l'application et la transaction dans CICS à l'aide de la transaction CEDA fournie.
2. Assurez-vous que la carte CICS MQSeries est installée sur votre système CICS. (Pour plus de détails, reportez-vous au manuel *MQSeries for OS/390 System Management Guide*.)
3. Assurez-vous que l'environnement JVM indiqué dans l'argument DHFJVM du JCL (Job Control Language) de démarrage CICS présente les entrées CLASSPATH et LIBPATH appropriées.
4. Lancez la transaction à l'aide d'un processus habituel.

Pour plus de détails sur l'exécution des transactions Java CICS, reportez-vous à la documentation fournie avec votre système CICS.

Traçage des programmes MQ base Java

MQ base Java est doté d'une fonction de trace qui peut être utilisée pour produire des messages de diagnostic si vous pensez que le code risque de provoquer des incidents. (Cette fonction ne doit normalement être utilisée qu'à la demande du service d'assistance IBM.)

Le traçage est commandé par les méthodes enableTracing et disableTracing de la classe MQEnvironment. Par exemple :

```
MQEnvironment.enableTracing(2);   // Traçage de niveau 2.
 ...                             // Ces commandes seront consignées.
MQEnvironment.disableTracing();  // Désactivation du traçage.
```

Les résultats de la fonction de trace sont transmis à la console Java (System.err).

Si votre programme est une application ou si vous l'exécutez sur votre disque local en utilisant l'afficheur d'applets, vous pouvez également réacheminer la sortie de trace vers le fichier de votre choix. L'extrait de code suivant est un exemple de réacheminement de la sortie de trace vers un fichier nommé myapp.trc :

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("monapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
  // Impossible d'ouvrir le fichier
  // Envoi de la sortie de trace vers System.err
  MQEnvironment.enableTracing(2);
}
```

Il existe cinq niveaux de traçage :

1. Traçage des points d'entrée, de sortie et des exceptions
2. En plus du niveau 1, informations sur les arguments
3. En plus du niveau 2, en-têtes et blocs de données MQSeries émis et reçus
4. En plus du niveau 3, données des messages utilisateur émis et reçus
5. En plus du niveau 4, traçage des méthodes de la machine virtuelle Java

Pour tracer les méthodes de la machine virtuelle Java au niveau 5, procédez comme suit :

- Dans le cas d'une application, exécutez-la à l'aide de la commande java_g (plutôt que java).
- Dans le cas d'une applet, exécutez-la à l'aide de la commande appletviewer_g (plutôt que appletviewer).

Remarques :

1. java_g n'est pas pris en charge par les applications HPJ (High Performance Java) sous OS/390.
2. java_g n'est pas pris en charge sous OS/400, mais vous disposez d'une fonction similaire en tapant OPTION(*VERBOSE) dans la commande RUNJVA.

Traçage des programmes MQ base Java

Chapitre 8. Comportement selon l'environnement

Le présent chapitre décrit le comportement des classes Java dans les différents environnements d'utilisation. Les classes MQSeries pour Java permettent de créer des applications qui peuvent être utilisées dans les environnements suivants :

1. MQSeries Client pour Java connecté à un serveur MQSeries V2.x sous UNIX ou Windows
2. MQSeries Client pour Java connecté à un serveur MQSeries V5 sous UNIX ou Windows
3. MQSeries Bindings pour Java exécuté sur un serveur MQSeries V5 sous UNIX ou Windows
4. MQSeries Bindings pour Java exécuté sur un serveur MQSeries pour MVS/ESA
5. MQSeries Bindings pour Java exécuté sur un serveur MQSeries pour MVS/ESA avec CICS Transaction Server pour OS/390 version 1.3

Quel que soit l'environnement, le code classes MQSeries pour Java fait appel aux services fournis par le serveur MQSeries sous-jacent. Il existe des différences au niveau des fonctionnalités fournies (ainsi, MQSeries version 5 offre un sur-ensemble des fonctionnalités de la version 2). Il existe également des différences de comportement pour certains appels et options de l'API. Ces différences de comportement sont en général mineures , elles sont surtout marquées entre les serveurs OS/390 (MQSeries pour MVS/ESA) et les serveurs sur d'autres plateformes.

Le module classes MQSeries pour Java propose un certain nombre de classes de base dont les fonctionnalités et le comportement sont identiques dans tous les environnements. Il offre également des ″extensions V5″, conçues pour être utilisées dans les environnements 2 et 3 uniquement. Les sections suivantes décrivent les classes de base et les extensions.

Détails des classes de base

Le module classes MQSeries pour Java contient les classes de base ci-après, qui peuvent être utilisées dans tous les environnements au prix de quelques variations mineures énumérées au paragraphe «Restrictions et variations relatives aux classes de base» à la page 76.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Sauf :
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  Sauf :
  - inquire()
  - set()
- MQMessage

  Sauf :
  - groupId

  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  Sauf :
  - knownDestCount
  - unknownDestCount
  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  Sauf :
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

Remarques :

1. Certaines constantes ne font pas partie de l'ensemble de base (voir «Restrictions et variations relatives aux classes de base» pour plus de précisions). Vous ne devez donc pas les utiliser pour créer des programmes entièrement portables.
2. Certaines plateformes ne prennent pas en charge tous les modes de connexion. Dans ce cas, vous ne pouvez utiliser que les classes et les options de base associées aux modes pris en charge. (Voir le tableau 1 à la page 5.)

Restrictions et variations relatives aux classes de base

Si, de manière générale, les classes de base ont un comportement cohérent dans tous les environnements, il existe néanmoins un petit nombre de restrictions et de variations énumérées dans le tableau 12.

En dehors de ces variations documentées, les classes de base ont un comportement similaire dans tous les environnements, bien que les classes MQSeries équivalentes présentent des différences selon l'environnement. En règle générale, ce comportement est celui des environnements 2 et 3.

Tableau 12. Restrictions et variations liées aux classes de base

Classe ou élément	Restrictions et variations
MQGMO_LOCK MQGMO_UNLOCK MQGMO_BROWSE_MSG_UNDER_CURSOR	Entraînent une erreur MQRC_OPTIONS_ERROR lorsqu'elles sont utilisées dans les environnements 4 ou 5.
MQPMO_NEW_MSG_ID MQPMO_NEW_CORREL_ID MQPMO_LOGICAL_ORDER	Génèrent des erreurs, sauf dans les environnements 2 et 3. (Voir Extensions V5.)
MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MESSAGE MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE	Génèrent des erreurs, sauf dans les environnements 2 et 3. (Voir Extensions V5.)

Tableau 12. Restrictions et variations liées aux classes de base (suite)

Classe ou élément	Restrictions et variations
MQGMO_SYNCPOINT_IF_PERSISTENT	Génère des erreurs dans l'environnement 1. (Voir Extensions V5.)
MQGMO_MARK_SKIP_BACKOUT	Entraîne une erreur MQRC_OPTIONS_ERROR sauf dans les environnements 4 et 5.
MQCNO_FASTPATH_BINDING	Pris en charge uniquement dans l'environnement 3. (Voir Extensions V5.)
Zones MQPMRF_*	Prises en charge uniquement dans les environnements 2 et 3.
Ecriture d'un message dont la priorité MQQueue.priority est supérieure à la priorité MaxPriority	Refusée avec erreurs MQCC_FAILED et MQRC_PRIORITY_ERROR dans les environnements 4 et 5. Les autres environnements acceptent ce type d'écriture en émettant les avertissements MQCC_WARNING et MQRC_PRIORITY_EXCEEDS_MAXIMUM et traitent le message comme s'il s'agissait d'un message MaxPriority.
BackoutCount	Les environnements 4 et 5 renvoient un nombre maximal d'annulations de 255, même si le message a été annulé plus de 255 fois.
Nom de file d'attente dynamique par défaut	CSQ.* pour les environnements 4 et 5. AMQ.* dans les autres environnements.
Options de MQMessage.report : MQRO_EXCEPTION_WITH_FULL_DATA MQRO_EXPIRATION_WITH_FULL_DATA MQRO_COA_WITH_FULL_DATA MQRO_COD_WITH_FULL_DATA MQRO_DISCARD_MSG	Non prises en charge si un message de rapport est généré par un gestionnaire de files d'attente OS/390, bien qu'elles puissent être définies dans tous lesenvironnements. Ce problème concerne tous les environnements Java, car il est tout à fait possible qu'il existe un gestionnaire de files d'attente OS/390 éloigné par rapport à l'application Java. Par conséquent, évitez d'utiliser ces options s'il y a la moindre probabilité pour qu'un gestionnaire de files d'attente OS/390 soit concerné.
MQQueueManager.commit() et MQQueueManager.backout()	Dans l'environnement 5, ces méthodes renvoient une erreur MQRC_ENVIRONMENT_ERROR. Dans cet environnement, les applications doivent donc utiliser les méthodes de synchronisation des tâches JCICS suivantes : com.ibm.cics.server.Task.commit() et com.ibm.cics.server.Task.rollback().

Restrictions

Tableau 12. Restrictions et variations liées aux classes de base (suite)

Classe ou élément	Restrictions et variations
Constructeur MQQueueManager	Dans les environnements 4 et 5, si les options présentes dans MQEnvironment (et l'argument de propriétés en option) signalent une connexion client, le constructeur échoue et renvoie une erreur MQRC_ENVIRONMENT_ERROR. Dans les environnements 4 et 5, le constructeur peut également renvoyer une erreur MQRC_CHAR_CONVERSION_ERROR. Assurez-vous que le composant des ressources linguistiques OS/390, Language Environment, est installé. Vérifiez surtout que l'outil de conversion de la page de codes IBM-1047 en ISO8859-1 est disponible. Dans les environnements 4 et 5, le constructeur peut également renvoyer l'erreur MQRC_UCS2_CONVERSION_ERROR. Les classes MQSeries pour Java tentent d'effectuer une conversion d'Unicode à la page de codes du gestionnaire de files d'attente. Par défaut, c'est la page de codes IBM-500 qui est retenue si aucune page de codes spécifique n'est disponible. Assurez-vous que vous disposez des tables de conversion appropriées pour Unicode, qui doivent être installées en tant que fonction en option de OS/390 C/C++. Assurez-vous également que le composant Language Environment peut localiser les tables. Pour plus de détails sur l'activation des conversions UCS-2, reportez-vous au manuel *OS/390 C/C++ Programming Guide*, SC09-2362.

Extensions de la version 5 fonctionnant dans d'autres environnements

Les fonctions du module classes MQSeries pour Java énumérées ci-dessous ont été conçues pour tirer parti des extensions de l'API inaugurées dans MQSeries V5. Elles fonctionnent correctement uniquement dans les environnements 2 et 3. La présente section décrit le comportement qu'elles auraient si elles étaient installées dans d'autres environnements.

Option du constructeur MQQueueManager

Le constructeur MQQueueManager comprend un argument facultatif constitué d'un nombre entier. Cet argument correspond à la zone MQCNO.options de MQI et sert à passer d'une connexion normale à une connexion fastpath et vice versa. Cette forme étendue du constructeur est acceptée dans tous les environnements, à condition que les seules options utilisées soient MQCNO_STANDARD_BINDING ou MQCNO_FASTPATH_BINDING. Toute autre option entraîne une erreur MQRC_OPTIONS_ERROR du constructeur. L'option Fastpath MQC.MQCNO_FASTPATH_BINDING n'a d'effet que lorsqu'elle est utilisée avec MQSeries V5 sur connexion directe (environnement 3). En cas d'utilisation dans un autre environnement, l'option est ignorée.

Méthode MQQueueManager.begin()

Prise en charge uniquement dans l'environnement 3. En cas d'utilisation dans un autre environnement, elle échoue et renvoie l'erreur MQRC_ENVIRONMENT_ERROR. MQSeries pour AS/400 ne prend pas en charge la méthode `begin()` pour démarrer des unités de travail globales coordonnées par le gestionnaire de files d'attente.

Options MQPutMessageOptions

Les indicateurs suivants peuvent être définis via les zones d'options de MQPutMessageOptions dans tous les environnements, mais si MQQueue.put() est ensuite utilisée dans un environnement autre que 2 ou 3, elle provoque l'erreur MQRC_OPTIONS_ERROR.

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER

Options MQGetMessageOptions

Les indicateurs suivants peuvent être définis via les zones d'options de MQGetMessageOptions dans tous les environnements, mais si MQQueue.get() est ensuite utilisée dans un environnement autre que 2 ou 3, elle provoque l'erreur MQRC_OPTIONS_ERROR.

- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MESSAGE
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE

L'indicateur suivant peut être défini via les zones d'options de MQGetMessageOptions dans tous les environnements, mais si MQQueue.get() est ensuite utilisée dans l'environnement 1, elle génère l'erreur MQRC_OPTIONS_ERROR.

- MQGMO_SYNCPOINT_IF_PERSISTENT

Extensions V5

Zones de MQGetMessageOptions

Les valeurs peuvent être définies dans les zones ci-après, quel que soit l'environnement. Toutefois, si la classe MQGetMessageOptions utilisée ensuite avec MQQueue.get() contient des valeurs autres que les valeurs par défaut dans un environnement autre que 2 ou 3, get() renvoie l'erreur MQRC_GMO_ERROR. Cela signifie que dans les environnements autres que 2 ou 3, ces zones reprendront toujours leur valeur initiale après un get() réussi.

- MatchOptions
- GroupStatus
- SegmentStatus
- Segmentation

Listes de diffusion

Les classes suivantes servent à créer des listes de diffusion :

- MQDistributionList
- MQDistributionListItem
- MQMessageTracker

Vous pouvez créer et définir MQDistributionList et MQDistributionListItems dans n'importe quel environnement, mais vous ne pouvez créer et ouvrir correctement MQDistributionList que dans les environnements 2 et 3. Toute tentative de création et d'ouverture de liste de diffusion dans un autre environnement est rejetée et génère l'erreur MQRC_OD_ERROR.

Zones de MQPutMessageOptions

Quatre zones de MQPMO sont représentés par les variables membres suivantes de la classe MQPutMessageOptions :

- knownDestCount
- unknownDestCount
- invalidDestCount
- recordFields

Bien que ces variables soient conçues principalement pour servir lors de l'utilisation de listes de diffusion, le serveur MQSeries V5 renseigne aussi les zones DestCount après un MQPUT dans une seule file d'attente. Si, par exemple, la file d'attente se résout en une file locale, knownDestCount reçoit la valeur 1 et les deux autres zones, la valeur 0. Dans les environnements 2 et 3, les valeurs fixées par le serveur V5 sont renvoyées dans la classe MQPutMessageOptions. Dans les autres environnements, les valeurs renvoyées sont simulées comme suit :

- Si le put() réussit, unknownDestCount reçoit la valeur 1 et les autres zones la valeur 0.
- Si le put() échoue, invalidDestCount reçoit la valeur 1 et les autres zones la valeur 0.

La variable recordFields est utilisée avec les listes de diffusion. Elle peut recevoir une valeur à tout moment, quel que soit l'environnement, mais cette valeur est ignorée si les options MQPutMessageOptions sont utilisées avec une opération MQQueue.put() plutôt qu'avec une opération MQDistributionList.put().

Zones de MQMD

Les zones suivantes de MQMD sont principalement liées à la segmentation des messages :

- GroupId

- MsgSeqNumber
- Offset MsgFlags
- OriginalLength

Si une application affecte à ces zones MQDM une valeur autre que la valeur par défaut, puis exécute une commande put() ou get() dans un environnement autre que les environnements 2 ou 3, le put() ou le get() entraîne une exception (MQRC_MD_ERROR). A la suite d'un put() ou un get() réussi dans un autre environnement que les environnements 2 ou 3, les nouvelles zones MQMD retrouvent toujours leur valeur par défaut. En principe, un message groupé ou segmenté ne doit pas être envoyé à une application Java connectée à un gestionnaire de files d'attente MQSeries de version antérieure à la version 5. Si une telle application émet un get() et si le message physique à récupérer fait partie d'un message groupé ou segmenté (ayant donc dans les zones MQMD des valeurs autres que les valeurs par défaut), il est récupéré sans provoquer d'erreur. Cependant, les zones MQMD du MQMessage ne sont pas mises à jour. La propriété de format du MQMessage reçoit la valeur MQFMT_MD_EXTENSION et les données réelles du message sont préfixées avec une structure MQMDE contenant les valeurs des nouvelles zones.

Extensions V5

Chapitre 9. Classes et interfaces MQ base Java

Le présent chapitre décrit toutes les classes et interfaces du module classes MQSeries pour Java. Il fournit des détails sur les variables, les constructeurs et les méthodes de chaque classe et interface.

Sont décrites les classes suivantes :

- MQChannelDefinition
- MQChannelExit
- MQDistributionList
- MQDistributionListItem
- MQEnvironment
- MQException
- MQGetMessageOptions
- MQManagedObject
- MQMessage
- MQMessageTracker
- MQPoolServices
- MQPoolServicesEvent
- MQPoolToken
- MQPutMessageOptions
- MQProcess
- MQQueue
- MQQueueManager
- MQSimpleConnectionManager

Sont décrites les interfaces suivantes :

- MQC
- MQPoolServicesEventListener
- MQConnectionManager
- MQReceiveExit
- MQSecurityExit
- MQSendExit
- ManagedConnection
- ManagedConnectionFactory
- ManagedConnectionMetaData

MQChannelDefinition

```
java.lang.Object
  |
  └ com.ibm.mq.MQChannelDefinition
```

public class **MQChannelDefinition**
extends **Object**

Cette classe sert à passer des informations relatives à la connexion avec le gestionnaire de files d'attente aux routines de sortie (ou «exits») d'envoi, de réception et de sécurité.

Remarque : Cette classe n'est pas applicable lorsque la connexion à MQSeries est directe (mode liens).

Variables

channelName
`public String channelName`

Nom du canal à travers lequel la connexion est établie.

queueManagerName
`public String queueManagerName`

Nom du gestionnaire de files d'attente avec lequel la connexion est établie.

maxMessageLength
`public int maxMessageLength`

Longueur maximale des messages envoyés au gestionnaire de files d'attente.

securityUserData
`public String securityUserData`

Zone de mémoire mise à disposition de l'exit de sécurité. Les données placées ici sont préservées entre deux appels de l'exit de sécurité et sont également mises à disposition des exits d'envoi et de réception.

sendUserData
`public String sendUserData`

Zone de mémoire mise à disposition de l'exit d'envoi. Les données placées ici sont préservées entre deux appels de l'exit d'envoi et sont également mises à disposition des exits de sécurité et de réception.

receiveUserData
`public String receiveUserData`

Zone de mémoire mise à disposition de l'exit de réception. Les données placées ici sont préservées entre deux appels de l'exit de réception et sont également mises à disposition des exits d'envoi et de sécurité.

connectionName
`public String connectionName`

Nom d'hôte TCP/IP de la machine hébergeant le gestionnaire de files d'attente.

remoteUserId

```
public String remoteUserId
```

ID utilisateur utilisé pour établir la connexion.

remotePassword

```
public String remotePassword
```

Mot de passe utilisé pour établir la connexion.

Constructeurs

MQChannelDefinition

```
public MQChannelDefinition()
```

MQChannelExit

```
java.lang.Object
   |
   └─ com.ibm.mq.MQChannelExit
```

public class **MQChannelExit**
extends **Object**

Cette classe définit les informations de contexte passées aux exits d'envoi, de réception et de sécurité lorsqu'ils sont appelés. La variable membre exitResponse doit être définie par l'exit pour indiquer quelle action le client MQSeries Java doit ensuite effectuer.

Remarque : Cette classe n'est pas applicable lorsque la connexion à MQSeries est directe (mode liens).

Variables

MQXT_CHANNEL_SEC_EXIT
```
public final static int MQXT_CHANNEL_SEC_EXIT
```

MQXT_CHANNEL_SEND_EXIT
```
public final static int MQXT_CHANNEL_SEND_EXIT
```

MQXT_CHANNEL_RCV_EXIT
```
public final static int MQXT_CHANNEL_RCV_EXIT
```

MQXR_INIT
```
public final static int MQXR_INIT
```

MQXR_TERM
```
public final static int MQXR_TERM
```

MQXR_XMIT
```
public final static int MQXR_XMIT
```

MQXR_SEC_MSG
```
public final static int MQXR_SEC_MSG
```

MQXR_INIT_SEC
```
public final static int MQXR_INIT_SEC
```

MQXCC_OK
```
public final static int MQXCC_OK
```

MQXCC_SUPPRESS_FUNCTION
```
public final static int MQXCC_SUPPRESS_FUNCTION
```

MQXCC_SEND_AND_REQUEST_SEC_MSG
```
public final static int MQXCC_SEND_AND_REQUEST_SEC_MSG
```

MQXCC_SEND_SEC_MSG
```
public final static int MQXCC_SEND_SEC_MSG
```

MQXCC_SUPPRESS_EXIT
```
public final static int MQXCC_SUPPRESS_EXIT
```

MQXCC_CLOSE_CHANNEL
```
public final static int MQXCC_CLOSE_CHANNEL
```

exitID `public int exitID`

Type de l'exit appelé. Pour un MQSecurityExit, cette variable a toujours la valeur MQXT_CHANNEL_SEC_EXIT. Pour un MQSendExit, sa valeur est toujours MQXT_CHANNEL_SEND_EXIT et pour un MQReceiveExit, elle est toujours MQXT_CHANNEL_RCV_EXIT.

exitReason

`public int exitReason`

Motif de l'appel de l'exit. Les valeurs possibles sont les suivantes :

MQXR_INIT
Initialisation de l'exit ; l'appel se produit après négociation des conditions de connexion au canal mais avant l'envoi de flux de sécurité.

MQXR_TERM
Fin de l'exit ; l'appel se produit après l'envoi des flux de déconnexion, mais avant la suppression de la connexion socket.

MQXR_XMIT
Dans le cas d'un exit d'envoi, indique que des données doivent être transmises au gestionnaire de files d'attente.

Dans le cas d'un exit de réception, indique que des données ont été reçues du gestionnaire de files d'attente.

MQXR_SEC_MSG
Indique à l'exit de sécurité qu'un message de sécurité a été reçu du gestionnaire de files d'attente.

MQXR_INIT_SEC
Indique que l'exit doit entamer le dialogue de sécurité avec le gestionnaire de files d'attente.

exitResponse

`public int exitResponse`

Définie par l'exit pour indiquer quelle action les classes MQSeries pour Java doivent ensuite effectuer. Les valeurs possibles sont les suivantes :

MQXCC_OK
Définie par l'exit de sécurité pour indiquer que les échanges de sécurité sont terminés.

Définie par l'exit d'envoi pour indiquer que les données renvoyées doivent être transmises au gestionnaire de files d'attente.

Définie par l'exit de réception pour indiquer que les données renvoyées peuvent être traitées par le client MQSeries pour Java.

MQXCC_SUPPRESS_FUNCTION
Définie par l'exit de sécurité pour indiquer que les communications avec le gestionnaire de files d'attente doivent être arrêtées.

MQXCC_SEND_AND_REQUEST_SEC_MSG
Définie par l'exit de sécurité pour indiquer que les données renvoyées doivent être transmises au gestionnaire de files d'attente et qu'une réponse du gestionnaire est attendue.

MQXCC_SEND_SEC_MSG
Définie par l'exit de sécurité pour indiquer que les données

renvoyées doivent être transmises au gestionnaire de files d'attente et qu'aucune réponse n'est attendue de ce dernier.

MQXCC_SUPPRESS_EXIT
Définie par n'importe quel exit pour indiquer qu'il ne faut plus l'appeler.

MQXCC_CLOSE_CHANNEL
Définie par n'importe quel exit pour indiquer que la connexion au gestionnaire de files d'attente doit être fermée.

maxSegmentLength
`public int maxSegmentLength`

Longueur maximale d'une transmission vers un gestionnaire de files d'attente.

Si l'exit renvoie des données qui doivent être envoyées au gestionnaire de files d'attente, la longueur des données renvoyées ne doit pas dépasser cette valeur.

exitUserArea
`public byte exitUserArea[]`

Zone de mémoire mise à disposition de l'exit.

Toute donnée placée dans la zone exitUserArea est préservée par MQSeries Client pour Java entre deux appels du même exit. (En d'autres termes, les exits d'envoi, de réception et de sécurité disposent chacun de leur propre zone mémoire utilisateur distincte.)

capabilityFlags
`public static final int capabilityFlags`

Indique les capacités du gestionnaire de files d'attente.

Seul l'indicateur MQC.MQCF_DIST_LISTS est pris en charge.

fapLevel
`public static final int fapLevel`

Niveau négocié de «Format et Protocole» (FAP).

Constructeurs

MQChannelExit
`public MQChannelExit()`

MQDistributionList

```
java.lang.Object
   |
   └─ com.ibm.mq.MQManagedObject
         |
         └─ com.ibm.mq.MQDistributionList
```

public class **MQDistributionList**
extends **MQManagedObject** (voir page 105.)

Remarque : Vous ne pouvez utiliser cette classe que lorsque vous êtes connecté à un gestionnaire de files d'attente MQSeries version 5 (ou ultérieure).

Pour créer une classe MQDistributionList, il faut utiliser le constructeur MQDistributionList ou la méthode accessDistributionList pour MQQueueManager.

Une liste de diffusion représente un ensemble de files d'attente ouvertes auxquelles des messages peuvent être envoyés via un seul appel de la méthode put(). (Voir la section relative aux listes de diffusion dans le manuel *MQSeries - Guide de programmation d'applications*.)

Constructeurs

MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
               MQDistributionListItem[] litems,
               int openOptions,
               String alternateUserId)
                          throws MQException
```

`qMgr` est le gestionnaire de files d'attente auprès duquel la liste doit être ouverte.

`litems` sont les éléments à inclure dans la liste de diffusion.

Pour plus de précisions sur les autres arguments, voir «accessDistributionList» à la page 155.

Méthodes

put

```
public synchronized void put(MQMessage message,
             MQPutMessageOptions putMessageOptions)
                              throws MQException
```

Insère un message dans les files d'attente désignées dans la liste de diffusion.

Arguments

message
Argument d'entrée et de sortie contenant les données du descripteur de message et les données de message renvoyées.

putMessageOptions
Options déterminant le comportement de la méthode MQPUT. (Pour plus de précisions, voir «MQPutMessageOptions» à la page 135.)

Déclenche l'exception MQException en cas d'échec de l'insertion.

getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem()
```

Renvoie le premier élément de la liste de diffusion, ou bien *null* si la liste est vide.

getValidDestinationCount

```
public int getValidDestinationCount()
```

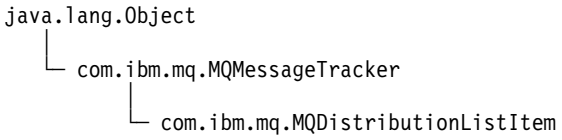
Renvoie le nombre d'éléments de la liste de diffusion dont l'ouverture a réussi.

getInvalidDestinationCount

```
public int getInvalidDestinationCount()
```

Renvoie le nombre d'éléments de la liste de diffusion dont l'ouverture a échoué.

MQDistributionListItem



public class **MQDistributionListItem**
extends **MQMessageTracker** (Voir page 127.)

Remarque : Vous ne pouvez utiliser cette classe que lorsque vous êtes connecté à un gestionnaire de files d'attente MQSeries version 5 (ou ultérieure).

Une instance de MQDistributionListItem représente un élément donné (une file d'attente) d'une liste de diffusion.

Variables

completionCode
`public int completionCode`

Code achèvement de la dernière opération effectuée sur cet élément. S'il s'agissait de la construction d'une liste MQDistributionList, le code achèvement porte sur l'ouverture de la file d'attente. S'il s'agissait d'une opération d'insertion (put), le code achèvement porte sur la tentative d'insertion d'un message dans cette file d'attente.

La valeur initiale est égale à ″0″.

queueName
`public String queueName`

Nom d'une file d'attente que vous voulez utiliser dans le cadre d'une liste de diffusion. Il ne peut pas s'agir du nom d'un modèle de file d'attente.

La valeur initiale est égale à ″″.

queueManagerName
`public String queueManagerName`

Nom du gestionnaire de files d'attente sur lequel la file d'attente est définie.

La valeur initiale est égale à ″″.

reasonCode
`public int reasonCode`

Code anomalie résultant de la dernière opération effectuée sur cet élément. S'il s'agissait de la construction d'une liste MQDistributionList, le code anomalie porte sur l'ouverture de la file d'attente. S'il s'agissait d'une opération d'insertion (put), le code anomalie porte sur la tentative d'insertion d'un message dans cette file d'attente.

La valeur initiale est égale à ″0″.

Constructeurs

MQDistributionListItem
`public MQDistributionListItem()`

MQDistributionListItem

Construit un nouvel objet MQDistributionListItem.

MQEnvironment

```
java.lang.Object
   |
   └─ com.ibm.mq.MQEnvironment
```

public class **MQEnvironment**
extends **Object**

Remarque : Toutes les méthodes et tous les attributs de cette classe s'appliquent aux connexions client des classes MQSeries pour Java, mais seules les méthodes enableTracing et disableTracing et les variables version_notice et properties s'appliquent aux connexions directes.

MQEnvironment contient des variables membres statiques qui permettent de contrôler l'environnement dans lequel un objet MQQueueManager (et sa connexion à MQSeries) est construit.

Les valeurs définies dans la classe MQEnvironment entrent en vigueur lors de l'appel du constructeur MQQueueManager ; vous devez donc définir les valeurs de la classe MQEnvironment avant de construire une instance de MQQueueManager.

Variables

Remarque : Les variables signalées par un astérisque (*) ne sont pas applicables lorsque la connexion à MQSeries est directe (mode liens).

version_notice

```
public final static String version_notice
```

Version actuelle des classes MQSeries pour Java.

securityExit*

```
public static MQSecurityExit securityExit
```

Un exit de sécurité vous permet de personnaliser les flux de sécurité occasionnés par toute tentative de connexion à un gestionnaire de files d'attente.

Pour créer votre propre exit de sécurité, définissez une classe mettant en œuvre l'interface MQSecurityExit, puis affectez securityExit à une instance de cette classe. Vous pouvez aussi conserver la valeur null pour securityExit, auquel cas aucun exit de sécurité n'est appelé.

Voir aussi «MQSecurityExit» à la page 164.

sendExit*

```
public static MQSendExit sendExit
```

Un exit d'envoi vous permet d'examiner et éventuellement de modifier les données envoyées à un gestionnaire de files d'attente. Il fonctionne normalement en tandem avec un exit de réception au niveau du gestionnaire de files d'attente.

Pour créer votre propre exit d'envoi, définissez une classe mettant en œuvre l'interface MQSendExit, puis affectez

sendExit à une instance de cette classe. Vous pouvez aussi conserver la valeur null pour sendExit, auquel cas aucun exit d'envoi n'est appelé.

Voir aussi «MQSendExit» à la page 166.

receiveExit*

```
public static MQReceiveExit receiveExit
```

Un exit de réception vous permet d'examiner et éventuellement de modifier les données reçues d'un gestionnaire de files d'attente. Il fonctionne normalement en tandem avec un exit d'envoi au niveau du gestionnaire de files d'attente.

Pour créer votre propre exit de réception, définissez une classe mettant en œuvre l'interface MQReceiveExit, puis affectez receiveExit à une instance de cette classe. Vous pouvez aussi conserver la valeur null pour receiveExit, auquel cas aucun exit de réception n'est appelé.

Voir aussi «MQReceiveExit» à la page 162.

hostname*

```
public static String hostname
```

Nom d'hôte TCP/IP de la machine hébergeant le serveur MQSeries. Si le nom d'hôte n'est pas défini et si aucune propriété de remplacement n'est définie, c'est le mode connexion directe qui est utilisé pour la connexion au gestionnaire de files d'attente local.

port* `public static int port`

Port sur lequel la connexion doit être établie. Il s'agit du port au niveau duquel le serveur MQSeries reste à l'écoute des demandes de connexion. La valeur par défaut est 1414.

channel*

```
public static String channel
```

Nom du canal de connexion sur le gestionnaire de files d'attente cible. Vous devez *absolument* définir cette variable membre ou la propriété correspondante avant de construire une instance de MQQueueManager qui sera utilisée en mode client.

userID*

```
public static String userID
```

Equivaut à la variable d'environnement MQ_USER_ID.

Si aucun exit de sécurité n'est défini pour ce client, la valeur de userID est envoyée au serveur pour être à disposition de l'exit de sécurité du serveur quand il sera appelé. Cette valeur peut être utilisée pour vérifier l'identité du client MQSeries.

La valeur par défaut est égale à "".

password*

```
public static String password
```

Equivaut à la variable d'environnement MQ_PASSWORD.

Si aucun exit de sécurité n'est défini pour ce client, la valeur de password est envoyée au serveur pour être à disposition de l'exit de sécurité du serveur quand il sera appelé. Cette valeur peut être utilisée pour vérifier l'identité du client MQSeries.

La valeur par défaut est égale à "".

properties

```
public static java.util.Hashtable properties
```

Ensemble de paires clé/valeur définissant l'environnement MQSeries.

Cette table de valeurs calculées vous permet de définir des propriétés d'environnement sous forme de paires clé/valeur et non sous forme de variables individuelles.

Ces propriétés peuvent également être passées sous forme de table de valeurs calculées en argument du constructeur MQQueueManager. Les propriétés passées en argument du constructeur ont la priorité sur les valeurs définies via cette variable `properties`, mais elles sont par ailleurs interchangeables. L'ordre de priorité pour déterminer les propriétés est le suivant :

1. Argument `properties` du constructeur MQQueueManager
2. MQEnvironment.properties
3. Autres variables MQEnvironment
4. Constantes par défaut

Les paires clé/valeur possibles sont les suivantes :

Clé	Valeur
MQC.CCSID_PROPERTY	Integer (remplace MQEnvironment.CCSID)
MQC.CHANNEL_PROPERTY	String (remplace MQEnvironment.channel)
MQC.CONNECT_OPTIONS_PROPERTY	Integer. Valeur par défaut : MQC.MQCNO_NONE
MQC.HOST_NAME_PROPERTY	String (remplace MQEnvironment.hostname)
MQC.ORB_PROPERTY	org.omg.CORBA.ORB (facultatif)
MQC.PASSWORD_PROPERTY	String (remplace MQEnvironment.password)
MQC.PORT_PROPERTY	Integer (remplace MQEnvironment.port)
MQC.RECEIVE_EXIT_PROPERTY	MQReceiveExit (remplace MQEnvironment.receiveExit)
MQC.SECURITY_EXIT_PROPERTY	MQSecurityExit (remplace MQEnvironment.securityExit)
MQC.SEND_EXIT_PROPERTY	MQSendExit (remplace MQEnvironment.sendExit.)

Clé	Valeur
MQC.TRANSPORT_PROPERTY	MQC.TRANSPORT_MQSERIES_BINDINGS ou MQC.TRANSPORT_MQSERIES_CLIENT ou MQC.TRANSPORT_VISIBROKER ou MQC.TRANSPORT_MQSERIES (valeur par défaut, qui sélectionne une connexion directe ou client, en fonction de la valeur de ″hostname″.)
MQC.USER_ID_PROPERTY	String (remplace MQEnvironment.userID.)

CCSID*

```
public static int CCSID
```

CCSID utilisé par le client.

La modification de cette valeur a des répercussions sur la façon dont le gestionnaire de files d'attente auquel vous vous connectez convertit les données des en-têtes MQSeries. Toutes les données des en-têtes MQSeries sont tirées de la partie invariable du code ASCII, sauf les données des zones applicationIdData et putApplicationName de la classe MQMessage. (Voir «MQMessage» à la page 108.)

Si vous veillez à ne pas utiliser de caractères relevant de la partie variable du code ASCII pour ces deux zones, vous pouvez en toute sécurité faire passer le CCSID de 819 à toute autre valeur de jeu de codes ASCII.

Vous bénéficiez d'un gain en performances si vous donnez au CCSID du client la même valeur que celle du gestionnaire de files d'attente auquel vous vous connectez, car ce dernier n'a pas besoin de convertir les en-têtes de messages.

La valeur par défaut est 819.

Constructeurs

MQEnvironment

```
public MQEnvironment()
```

Méthodes

disableTracing

```
public static void disableTracing()
```

Désactive la fonction de trace de MQSeries Client pour Java.

enableTracing

```
public static void enableTracing(int level)
```

Active la fonction de trace de MQSeries Client pour Java.

Arguments

level Niveau de traçage demandé, entre 1 et 5 (5 étant le niveau le plus détaillé).

enableTracing

```
public static void enableTracing(int level,
                                 OutputStream stream)
```

Active la fonction de trace MQSeries Client pour Java.

Arguments

level Niveau de traçage demandé, entre 1 et 5 (5 étant le niveau le plus détaillé).

stream Flot dans lequel les données de trace sont écrites.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager cxManager)
```

Définit le MQConnectionManager fourni comme étant le ConnectionManager par défaut. Le ConnectionManager par défaut est utilisé lorsqu'aucun ConnectionManager n'est spécifié dans le constructeur MQQueueManager. Cette méthode vide également l'ensemble d'objets MQPoolToken.

Arguments

cxManager
MQConnectionManager devant être le ConnectionManager par défaut.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager
                    (javax.resource.spi.ConnectionManager cxManager)
```

Définit le ConnectionManager par défaut et vide l'ensemble d'objets MQPoolToken. Le ConnectionManager par défaut est utilisé lorsqu'aucun ConnectionManager n'est spécifié dans le constructeur MQQueueManager.

Cette méthode nécessite un JVM au niveau Java 2 version 1.3 ou suivante, avec JAAS version 1.0 ou suivante installé.

Arguments

cxManager
ConnectionManager par défaut (qui assure la mise en oeuvre de l'interface javax.resource.spi.ConnectionManager).

getDefaultConnectionManager

```
public static javax.resource.spi.ConnectionManager
                                 getDefaultConnectionManager()
```

Renvoie le ConnectionManager par défaut. Si le ConnectionManager par défaut est un MQConnectionManager, renvoie la valeur NULL.

addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token)
```

Ajoute l'objet MQPoolToken fourni à l'ensemble de jetons. Le ConnectionManager par défaut peut utiliser cet objet MQPoolToken comme indication. Généralement, cette méthode n'est activée que lorsqu'il y a au moins un jeton dans l'ensemble.

Arguments

token Objet MQPoolToken à ajouter à l'ensemble de jetons.

addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken()
```

Construit un objet MQPoolToken et l'ajoute à l'ensemble de jetons. L'objet MQPoolToken est renvoyé à l'application pour être transmis ultérieurement dans removeConnectionPoolToken().

removeConnectionPoolToken

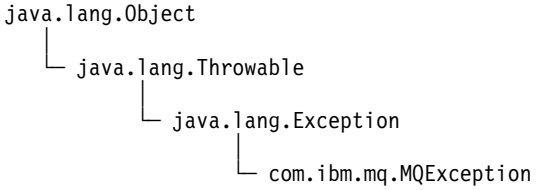
```
public static void removeConnectionPoolToken(MQPoolToken token)
```

Supprime l'objet MQPoolToken spécifié de l'ensemble de jetons. Si cet objet MQPoolToken n'est pas dans l'ensemble de jetons, aucune action n'est effectuée.

Arguments :

token Objet MQPoolToken à supprimer de l'ensemble de jetons.

MQException



public class **MQException**
extends **Exception**

Une exception MQException est déclenchée à chaque fois qu'une erreur MQSeries se produit. Vous pouvez modifier l'emplacement de consignation des exceptions en définissant la valeur MQException.log. Par défaut, cette valeur est égale à System.err. Cette classe contient les définitions des constantes de code achèvement et de code d'erreur. Les constantes dont le nom débute par MQCC_ sont des codes d'achèvement MQSeries et celles dont le nom commence par MQRC_ sont des codes anomalie MQSeries. Le manuel *MQSeries Application Programming Reference* contient une description complète de ces erreurs et de leurs causes probables.

Variables

log `public static java.io.outputStreamWriter log`

Flot dans lequel les exceptions sont consignées. (Par défaut, il s'agit de System.err.) Si vous donnez à cette variable une valeur NULL, aucune consignation n'est effectuée.

completionCode
`public int completionCode`

Code achèvement MQSeries lié à l'erreur. Les valeurs possibles sont les suivantes :
- MQException.MQCC_WARNING (avertissement)
- MQException.MQCC_FAILED (échec)

reasonCode
`public int reasonCode`

Code anomalie MQSeries décrivant l'erreur. Pour obtenir une explication complète des différents codes anomalie, consultez le manuel *MQSeries Application Programming Reference.*

exceptionSource
`public Object exceptionSource`

Instance d'objet ayant déclenché l'exception. Cette valeur peut vous aider à déterminer la cause de l'erreur.

Constructeurs

MQException

```
public MQException(int completionCode,
                   int reasonCode,
                   Object source)
```

Permet de construire un nouvel objet MQException.

Arguments

completionCode
Code achèvement MQSeries.

reasonCode
Code anomalie MQSeries.

source Objet au niveau duquel l'erreur s'est produite.

MQGetMessageOptions

```
java.lang.Object
   |
   └ com.ibm.mq.MQGetMessageOptions
```

public class **MQGetMessageOptions**
extends **Object**

Cette classe contient des options qui déterminent le comportement de MQQueue.get().

Remarque : Le comportement de certaines des options disponibles dans cette classe dépend de l'environnement dans lequel elles sont utilisées. Ces options sont signalées par un astérisque (*****). Pour plus de précisions, reportez-vous au «Chapitre 8. Comportement selon l'environnement» à la page 75.

Variables

options

```
public int options
```

Options déterminant le comportement de la méthode MQQueue.get. Vous pouvez définir, une, plusieurs ou aucune des valeurs indiquées ci-dessous. Pour définir plusieurs options, vous pouvez additionner leurs valeurs ou les combiner à l'aide de l'opérateur bit OR.

MQC.MQGMO_NONE

MQC.MQGMO_WAIT
Attendre l'arrivée d'un message.

MQC.MQGMO_NO_WAIT
Rendre la main immédiatement s'il n'y a pas de message approprié.

MQC.MQGMO_SYNCPOINT
Extraire le message sous le contrôle d'un point de synchronisation ; le message est marqué comme étant indisponible vis-à-vis des autres applications, mais il n'est supprimé de la file d'attente que lorsque l'unité de travail est validée. Le message est de nouveau disponible si l'unité de travail est annulée.

MQC.MQGMO_NO_SYNCPOINT
Extraire le message sans le contrôle d'un point de synchronisation.

MQC.MQGMO_BROWSE_FIRST
Parcourir les messages depuis le début de la file d'attente.

MQC.MQGMO_BROWSE_NEXT
Parcourir les messages à partir de la position en cours dans la file d'attente.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Parcourir le message se trouvant sous le curseur.

MQC.MQGMO_MSG_UNDER_CURSOR
Extraire le message se trouvant sous le curseur.

MQC.MQGMO_LOCK*
Verrouiller le message parcouru.

MQC.MQGMO_UNLOCK*
Déverrouiller un message précédemment verrouillé.

MQC.MQGMO_ACCEPT_TRUNCATED_MSG
Autoriser les données de message tronquées.

MQC.MQGMO_FAIL_IF_QUIESCING
Provoquer une erreur si le gestionnaire de files d'attente est au repos.

MQC.MQGMO_CONVERT
Demander la conversion des données applicatives, afin de respecter le jeu de caractères (characterSet) et les attributs de codage du MQMessage, avant de copier les données dans le tampon de message. Comme la conversion est également appliquée lorsque les données sont extraites du tampon de message, les applications n'ont en général pas à définir cette option.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Extraire un message sous le contrôle d'un point de synchronisation si le message est persistant.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Autoriser l'annulation d'une unité de travail sans réinsertion du message dans la file d'attente.

Segmentation et regroupement Les messages MQSeries peuvent être envoyés et reçus individuellement. Ils peuvent aussi être divisés en plusieurs segments à l'envoi et à la réception ou être liés à d'autres messages dans le cadre d'un groupe.

Chaque élément de données envoyé est considéré comme un message *physique* qui peut être un message *logique* complet ou bien un segment d'un message logique plus long.

Chaque message physique dispose en général de son propre `MsgId`. Tous les segments d'un même message logique ont les mêmes valeurs `groupId` et `MsgSeqNumber`, mais chacun d'eux possède une valeur `Offset` propre. La zone `Offset` précise le décalage des données du message physique par rapport au début du message logique. Les segments ont en général une valeur `MsgId` propre car ils sont physiquement distincts les uns des autres.

Les messages logiques qui font partie d'un groupe ont la même valeur `groupId`, mais chaque message du groupe a sa propre valeur `MsgSeqNumber`. Les messages d'un groupe peuvent également être segmentés.

Les options suivantes permettent d'agir sur les messages segmentés ou groupés.

MQC.MQGMO_LOGICAL_ORDER*
Renvoyer dans leur ordre logique les messages de groupes et les segments de messages logiques.

MQC.MQGMO_COMPLETE_MSG*
N'extraire que les messages logiques complets.

MQC.MQGMO_ALL_MSGS_AVAILABLE*
N'extraire les messages faisant partie d'un groupe que lorsque tous les messages du groupe sont disponibles.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
N'extraire les segments d'un message logique que lorsque tous les segments du message sont disponibles.

waitInterval
`public int waitInterval`

Durée maximale (en millisecondes) d'attente du message approprié lors d'un appel MQQueue.get (utilisée en conjonction avec MQC.MQGMO_WAIT). La valeur MQC.MQWI_UNLIMITED indique que la durée d'attente n'est pas limitée.

resolvedQueueName
`public String resolvedQueueName`

Zone de sortie à laquelle le gestionnaire de files d'attente affecte le nom local de la file d'attente d'où le message a été extrait. Ce nom est différent de celui utilisé pour ouvrir la file d'attente si cette ouverture a été effectuée à l'aide d'un alias ou d'un modèle de file d'attente.

matchOptions*
`public int matchOptions`

Critères de sélection servant à définir le message à extraire. Les options de sélection suivantes peuvent être définies :

MQC.MQMO_MATCH_MSG_ID
ID du message à extraire.

MQC.MQMO_MATCH_CORREL_ID
ID corrélation du message à extraire.

MQC.MQMO_MATCH_GROUP_ID
ID de groupe du message à extraire.

MQC.MQMO_MATCH_MSG_SEQ_NUMBER
Numéro de séquence du message à extraire.

MQC.MQMO_NONE
Pas de sélection particulière.

groupStatus*
`public char groupStatus`

Zone de sortie indiquant si le message extrait fait partie d'un groupe et, dans ce cas, s'il s'agit du dernier message du groupe. Les valeurs possibles sont les suivantes :

MQC.MQGS_NOT_IN_GROUP
Le message ne fait pas partie d'un groupe.

MQC.MQGS_MSG_IN_GROUP
Le message fait partie d'un groupe, mais ce n'est pas le dernier du groupe.

MQC.MQGS_LAST_MSG_IN_GROUP
Le message est le dernier d'un groupe. C'est également la valeur renvoyée si le groupe se compose de ce seul message.

segmentStatus*

```
public char segmentStatus
```

Zone de sortie indiquant si le message extrait est un segment d'un message logique. Si le message est effectivement un segment, cet indicateur précise s'il s'agit du dernier segment du message logique. Les valeurs possibles sont les suivantes :

MQC.MQSS_NOT_A_SEGMENT
Le message n'est pas un segment.

MQC.MQSS_SEGMENT
Le message est bien un segment, mais ce n'est pas le dernier segment du message logique.

MQC.MQSS_LAST_SEGMENT
Le message est le dernier segment du message logique. C'est également la valeur renvoyée si le message logique se compose de ce seul segment.

segmentation*

```
public char segmentation
```

Zone de sortie indiquant si la segmentation est autorisée ou non pour le message extrait. Les valeurs possibles sont les suivantes :

MQC.MQSEG_INHIBITED
La segmentation n'est pas autorisée.

MQC.MQSEG_ALLOWED
La segmentation est autorisée.

Constructeurs

MQGetMessageOptions

```
public MQGetMessageOptions()
```

Permet de construire un nouvel objet MQGetMessageOptions doté de l'option MQC.MQGMO_NO_WAIT, d'une durée d'attente de zéro et d'un nom de file d'attente vide.

MQManagedObject

```
java.lang.Object
   |
   └─ com.ibm.mq.MQManagedObject
```

public class **MQManagedObject**
extends **Object**

MQManagedObject est une superclasse pour MQQueueManager, MQQueue et MQProcess. Elle permet de lire et de définir les attributs de ces ressources.

Variables

alternateUserId
`public String alternateUserId`

ID utilisateur secondaire spécifié éventuellement lors de l'ouverture de cette ressource. La définition de cet attribut n'a aucun effet.

name `public String name`

Nom de cette ressource (fourni par la méthode d'accès ou alloué par le gestionnaire de files d'attente dans le cas d'une file d'attente dynamique). La définition de cet attribut n'a aucun effet.

openOptions
`public int openOptions`

Options définies lors de l'ouverture de cette ressource. La définition de cet attribut n'a aucun effet.

isOpen
`public boolean isOpen`

Indique si cette ressource est actuellement ouverte. Cet attribut est *déconseillé* et sa configuration n'a aucun effet.

connectionReference
`public  MQQueueManager connectionReference`

Gestionnaire de files d'attente auquel cette ressource appartient. La définition de cet attribut n'a aucun effet.

closeOptions
`public int closeOptions`

Définissez cet attribut pour spécifier la manière dont la ressource doit être fermée. Sa valeur par défaut est égale à MQC.MQCO_NONE, et c'est la seule valeur autorisée pour les ressources autres que les files d'attente dynamiques permanentes et les files d'attente dynamiques temporaires auxquelles accèdent les objets qui les ont créées. Pour ces dernières, les autres valeurs suivantes sont autorisées :

MQC.MQCO_DELETE
Supprimer la file d'attente s'il n'y a plus de messages.

MQC.MQCO_DELETE_PURGE
Supprimer la file d'attente en purgeant tous les messages restants.

Constructeurs

MQManagedObject

```
protected MQManagedObject()
```

Méthode constructrice.

Méthodes

getDescription

```
public String getDescription()
```

Déclenche l'exception MQException.

Renvoie la description de cette ressource conservée par le gestionnaire de files d'attente.

Si cette méthode est appelée après fermeture de la ressource, l'exception MQException est générée.

inquire

```
public void inquire(int selectors[],
                    int intAttrs[],
                    byte charAttrs[])
```

Déclenche l'exception MQException.

Renvoie un tableau d'entiers et un ensemble de chaînes de caractères contenant les attributs d'un objet (file d'attente, processus ou gestionnaire de files d'attente).

Les attributs à lire sont précisés dans le tableau `selectors`. Pour plus de précisions sur les sélecteurs possibles et les valeurs entières correspondantes, consultez le manuel *MQSeries Application Programming Reference*.

Notez que bon nombre des attributs les plus courants peuvent être lus via les méthodes getXXX() des objets MQManagedObject, MQQueue, MQQueueManager et MQProcess.

Arguments

selectors
: Tableau d'entiers identifiant les attributs dont vous voulez lire les valeurs.

intAttrs
: Tableau dans lequel sont renvoyées les valeurs d'attributs entières. Les valeurs d'attributs entières sont renvoyées dans l'ordre d'apparition des sélecteurs d'attributs entiers dans le tableau `selectors`.

charAttrs
: Tampon dans lequel les attributs caractères sont renvoyés concaténés. Les attributs caractères sont renvoyés dans l'ordre d'apparition des sélecteurs d'attributs caractères dans le tableau `selectors`. La longueur de chaque chaîne d'attribut est fixe pour chaque attribut.

Déclenche l'exception MQException si l'interrogation échoue.

isOpen

```
public boolean isOpen()
```

Renvoie la valeur de la variable isOpen.

set

```
public synchronized void set(int selectors[],
                             int intAttrs[],
                             byte charAttrs[])
```

Déclenche l'exception MQException.

Permet de définir les attributs précisés dans le vecteur selectors.

Les attributs à définir sont spécifiés dans le tableau selectors. Pour plus de précisions sur les sélecteurs possibles et les valeurs entières correspondantes, consultez le manuel *MQSeries Application Programming Reference*.

Notez que certains attributs de files d'attente peuvent être définis via les méthodes setXXX() de MQQueue.

Arguments

selectors
Tableau d'entiers identifiant les attributs dont vous voulez définir les valeurs.

intAttrs
Tableau de valeurs d'attributs entières à définir. Ces valeurs doivent être fournies dans le même ordre que les sélecteurs d'attributs entiers du tableau selectors.

charAttrs
Tampon dans lequel les attributs caractères à définir sont concaténés. Ces valeurs doivent être fournies dans le même ordre que les sélecteurs d'attributs caractères du tableau selectors. La longueur de chaque attribut caractère est fixe.

Déclenche l'exception MQException si la définition échoue.

close

```
public synchronized void close()
```

Déclenche l'exception MQException.

Permet de fermer l'objet. Plus aucune opération sur cette ressource n'est autorisée une fois cette méthode appelée. Le comportement de la méthode close peut être modifié en affectant une valeur à l'attribut closeOptions.

Génère l'exception MQException si l'appel de MQSeries échoue.

MQMessage

```
java.lang.Object
   |
   └─ com.ibm.mq.MQMessage
```

public class **MQMessage**
implements **DataInput**, **DataOutput**

MQMessage représente à la fois le descripteur d'un message MQSeries et les données qu'il contient. Il existe un ensemble de méthodes readXXX pour lire les données d'un message, ainsi qu'un ensemble de méthodes writeXXX pour écrire des données dans un message. Le format des nombres et des chaînes utilisé par ces méthodes de lecture et d'écriture peut être contrôlé via les variables membres encoding et characterSet. Les autres variables membres contiennent des données de contrôle qui accompagnent les données de message applicatif lorsqu'un message transite entre l'application émettrice et l'application réceptrice. L'application peut placer des valeurs dans la variable membre avant d'insérer un message dans une file d'attente et peut y lire des valeurs après extraction d'un message d'une file d'attente.

Variables

report `public int report`

Un rapport est un message concernant un autre message. Cette variable membre permet à l'application émettrice du message d'indiquer quels rapports sont requis, si les données du message applicatif doivent être incluses dans ces rapports, et également de quelle façon les ID de message et de corrélation du rapport ou de la réponse doivent être définis. Vous pouvez demander un, plusieurs ou aucun des types de rapports suivants :
- Exception
- Expiration
- Confirmation à l'arrivée
- Confirmation à la livraison

Pour chaque type de rapport, une seule des trois valeurs correspondantes précisées ci-dessous doit être spécifiée, selon que les données du message applicatif doivent être incluses ou non dans le rapport.

Remarque : Les valeurs signalées par le signe****** dans la liste suivante ne sont pas prises en charge par les gestionnaires de files d'attente MVS et ne doivent pas être utilisées si votre application est susceptible d'accéder à un gestionnaire de files d'attente MVS, quelle que soit la plateforme sur laquelle l'application est exécutée.

Les valeurs possibles sont les suivantes :
- MQC.MQRO_EXCEPTION
- MQC.MQRO_EXCEPTION_WITH_DATA
- MQC.MQRO_EXCEPTION_WITH_FULL_DATA******
- MQC.MQRO_EXPIRATION
- MQC.MQRO_EXPIRATION_WITH_DATA
- MQC.MQRO_EXPIRATION_WITH_FULL_DATA******
- MQC.MQRO_COA (confirmation à l'arrivée)
- MQC.MQRO_COA_WITH_DATA

- MQC.MQRO_COA_WITH_FULL_DATA**
- MQC.MQRO_COD (confirmation à la livraison)
- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

Vous pouvez spécifier l'une des valeurs suivantes pour indiquer de quelle façon l'ID de message doit être généré pour le rapport ou le message de réponse :
- MQC.MQRO_NEW_MSG_ID (créer un nouvel ID)
- MQC.MQRO_PASS_MSG_ID (reprendre l'ID du message d'origine)

Vous pouvez spécifier l'une des valeurs suivantes pour indiquer de quelle façon l'ID de corrélation du rapport ou du message de réponse doit être défini :
- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID (copier son ID de message)
- MQC.MQRO_PASS_CORREL_ID (reprendre l'ID corrélation d'origine)

Vous pouvez spécifier l'une des valeurs suivantes pour indiquer de quelle manière il faut traiter le message d'origine lorsqu'il ne peut être livré dans la file d'attente de destination :
- MQC.MQRO_DEAD_LETTER_Q (file d'attente de rebut)
- MQC.MQRO_DISCARD_MSG ** (supprimer le message)

Si aucune option de rapport n'est spécifiée, les options par défaut sont les suivantes :

```
MQC.MQRO_NEW_MSG_ID |
MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
MQC.MQRO_DEAD_LETTER_Q
```

Vous pouvez spécifier une des valeurs suivantes ou les deux valeurs pour demander que l'application destinatrice envoie un rapport d'action positive ou d'action négative.
- MQRO_PAN (action positive)
- MQRO_NAN (action négative)

messageType

```
public int messageType
```

Indique le type du message. Les valeurs suivantes sont définies par le système :
- MQC.MQMT_DATAGRAM (datagramme)
- MQC.MQMT_REQUEST (demande)
- MQC.MQMT_REPLY (réponse)
- MQC.MQMT_REPORT (rapport)

Des valeurs définies par les applications peuvent également être utilisées. Elles doivent être situées dans la plage qui va de MQC.MQMT_APPL_FIRST à MQC.MQMT_APPL_LAST.

La valeur par défaut de cette zone est MQC.MQMT_DATAGRAM.

expiry `public int expiry`

Délai d'expiration, en dixièmes de seconde, défini par l'application qui insère le message. Lorsque le délai d'expiration d'un message est passé, ce message est susceptible d'être supprimé par le gestionnaire de files

d'attente. Si un des indicateurs MQC_MQRO_EXPIRATION est spécifié dans le message, un rapport est généré lorsque le message est supprimé.

La valeur par défaut de cette variable est MQC.MQEI_UNLIMITED, c'est-à-dire que le message n'expire jamais.

feedback

```
public int feedback
```

Cette variable est utilisée avec un message de type MQC.MQMT_REPORT pour préciser la nature du rapport. Les codes retour suivants sont définis par le système :

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Vous pouvez également utiliser des valeurs de codes retour situées dans la plage MQC.MQFB_APPL_FIRST à MQC.MQFB_APPL_LAST.

La valeur par défaut de cette zone est MQC.MQFB_NONE, ce qui signifie qu'aucun code retour n'est renvoyé.

encoding

```
public int encoding
```

Cette variable membre spécifie la représentation utilisée pour les valeurs numériques dans les données du message applicatif ; cette représentation s'applique aux données binaires, décimales condensées et à virgule flottante. Le comportement des méthodes read et write appliquées à ces formats numériques est modifié de façon correspondante.

Les codages suivants sont définis pour les entiers binaires :

MQC.MQENC_INTEGER_NORMAL
Entiers big-endian, comme dans Java

MQC.MQENC_INTEGER_REVERSED
Entiers little-endian, comme ceux utilisés par les PC.

Les codages suivants sont définis pour les entiers décimaux condensés :

MQC.MQENC_DECIMAL_NORMAL
Décimal condensé big-endian, comme ceux utilisés par System/390.

MQC.MQENC_DECIMAL_REVERSED
Décimal condensé little-endian.

Les codages suivants sont définis pour les nombres à virgule flottante :

MQC.MQENC_FLOAT_IEEE_NORMAL
Variables flottantes IEEE big-endian, comme dans Java.

MQC.MQENC_FLOAT_IEEE_REVERSED
Variables flottantes IEEE little-endian, comme celles utilisées par les PC.

MQC.MQENC_FLOAT_S390
Format à virgule flottante de System/390.

La valeur de la zone encoding doit être élaborée en additionnant une valeur de chacune de ces trois sections (ou en utilisant l'opérateur bit OR). La valeur par défaut est la suivante :

```
MQC.MQENC_INTEGER_NORMAL |
MQC.MQENC_DECIMAL_NORMAL |
MQC.MQENC_FLOAT_IEEE_NORMAL
```

Pour vous faciliter la tâche, cette valeur est également représentée par MQC_MQENC_NATIVE. Avec ce paramétrage, writeInt() écrit un entier big-endian et readInt() lit un entier big-endian. En revanche, si l'indicateur MQC.MQENC_INTEGER_REVERSED a été défini, writeInt() écrit un entier little-endian et readInt() lit un entier little-endian.

Notez que la conversion des nombres à virgule flottante du format IEEE au format System/390 peut entraîner une perte de précision.

characterSet

```
public int characterSet
```

Cette variable indique l'identificateur du jeu de caractères dans lequel sont codées les données du message applicatif. Le comportement des méthodes readString, readLine et writeString est modifié de manière correspondante.

La valeur par défaut de cette zone est égale à MQC.MQCCSI_Q_MGR, qui spécifie que les données du message applicatif sont codées dans le jeu de caractères du gestionnaire de files d'attente. Les valeurs de jeux de caractères supplémentaires figurant dans le tableau 13 sont prises en charge.

Tableau 13. Identificateurs des jeux de caractères pris en charge

characterSet	Description
819	iso-8859-1 / latin1 / ibm819
912	iso-8859-2 / latin2 / ibm912
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cyrillique / ibm915
1089	iso-8859-6 / arabe / ibm1089
813	iso-8859-7 / grec / ibm813
916	iso-8859-8 / hébreu / ibm916
920	iso-8859-9 / latin5 / ibm920
37	ibm037
273	ibm273
277	ibm277
278	ibm278
280	ibm280
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
437	ibm437 / PC d'origine

Tableau 13. Identificateurs des jeux de caractères pris en charge (suite)

characterSet	Description
500	ibm500
737	ibm737 / PC Grec
775	ibm775 / PC Balte
838	ibm838
850	ibm850 / PC Latin 1
852	ibm852 / PC Latin 2
855	ibm855 / PC Cyrillique
856	ibm856
857	ibm857 / PC Turc
860	ibm860 / PC Portugais
861	ibm861 / PC Islandais
862	ibm862 / PC Hébreu
863	ibm863 / PC Canadien français
864	ibm864 / PC Arabe
865	ibm865 / PC Scandinave
866	ibm866 / PC Russe
868	ibm868
869	ibm869 / PC Grec moderne
870	ibm870
871	ibm871
874	ibm874
875	ibm875
918	ibm918
921	ibm921
922	ibm922
930	ibm930
933	ibm933
935	ibm935
937	ibm937
939	ibm939
942	ibm942
948	ibm948
949	ibm949
950	ibm950 / Chinois traditionnel ″Big 5″
964	ibm964 / Chinois traditionnel CNS 11643
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1381	ibm1381
1383	ibm1383
2022	JIS
932	PC Japonais
954	EUCJIS
1250	Windows Latin 2
1251	Windows Cyrillique
1252	Windows Latin 1

Tableau 13. Identificateurs des jeux de caractères pris en charge (suite)

characterSet	Description
1253	Windows Grec
1254	Windows Turque
1255	Windows Hébreu
1256	Windows Arabe
1257	Windows Balte
1258	Windows Vietnamien
33722	ibm33722
5601	ksc-5601 Coréen
1200	Unicode
1208	UTF-8

format

```
public String format
```

Nom de format utilisé par l'émetteur du message pour indiquer au destinataire la nature des données qu'il contient. Vous pouvez définir vos propres noms de format, mais notez que les noms commençant par ″MQ″ ont des significations définies par le gestionnaire de files d'attente. Les formats intégrés aux gestionnaires de files d'attente sont les suivantes :

MQC.MQFMT_NONE
Pas de nom de format

MQC.MQFMT_ADMIN
Message de demande/de réponse du serveur de commandes

MQC.MQFMT_COMMAND_1
Message de réponse à une commande (type 1)

MQC.MQFMT_COMMAND_2
Message de réponse à une commande (type 2)

MQC.MQFMT_DEAD_LETTER_HEADER
En-tête de non-distribution.

MQC.MQFMT_EVENT
Message d'événement.

MQC.MQFMT_PCF
Message défini par l'utilisateur au format de commande programmable (PCF)

MQC.MQFMT_STRING
Message composé uniquement de caractères

MQC.MQFMT_TRIGGER
Message de déclenchement

MQC.MQFMT_XMIT_Q_HEADER
En-tête de file d'attente de transmission

La valeur par défaut de cette variable est égale à MQC.MQFMT_NONE.

priority

```
public int priority
```

Niveau de priorité du message. Dans les messages sortants, la valeur spéciale de priorité MQC.MQPRI_PRIORITY_AS_Q_DEF peut aussi être

attribuée, auquel cas le niveau de priorité du message est égal à l'attribut de priorité par défaut de la file d'attente de destination.

La valeur par défaut de cette variable est égale à MQC.MQPRI_PRIORITY_AS_Q_DEF.

persistence

```
public int persistence
```

Persistance du message. Les valeurs suivantes sont définies :

- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

La valeur par défaut est égale à MQC.MQPER_PERSISTENCE_AS_Q_DEF, ce qui signifie que la persistance du message est égale à l'attribut de persistance par défaut de la file d'attente de destination.

messageId

```
public byte messageId[]
```

Dans le cadre d'un appel MQQueue.get(), cette zone spécifie l'ID du message à extraire. Normalement, le gestionnaire de files d'attente renvoie le premier message dont l'ID de message et l'ID de corrélation correspondent à ceux qui ont été spécifiés. La valeur spéciale MQC.MQMI_NONE permet de spécifier que l'ID de message est *indifférent*.

Dans le cadre d'un appel MQQueue.put(), cette zone spécifie l'ID de message à utiliser. Si c'est MQC.MQMI_NONE qui est spécifié, le gestionnaire de files d'attente génère un ID de message unique lors de l'insertion du message dans la file. La valeur de cette variable est mise à jour après l'insertion, pour indiquer l'ID de message qui a été utilisé.

La valeur par défaut est égale à MQC.MQMI_NONE.

correlationId

```
public byte correlationId[]
```

Dans le cadre d'un appel MQQueue.get(), cette zone spécifie l'ID de corrélation du message à extraire. Normalement, le gestionnaire de files d'attente renvoie le premier message dont l'ID de message et l'ID de corrélation correspondent à ceux qui ont été spécifiés. La valeur spéciale MQC.MQCI_NONE permet de spécifier que l'ID de corrélation est *indifférent*.

Dans le cadre d'un appel MQQueue.put(), cette zone spécifie l'ID de correlation à utiliser.

La valeur par défaut est égale à MQC.MQCI_NONE.

backoutCount

```
public int backoutCount
```

Nombre de fois où le message a déjà été renvoyé par un appel MQQueue.get() dans le cadre d'une unité de travail, puis annulé.

La valeur par défaut est égale à zéro.

replyToQueueName

```
public String replyToQueueName
```

Nom de la file d'attente de messages à laquelle l'application qui a émis la requête d'extraction du message (get) doit envoyer les messages MQC.MQMT_REPLY et MQC.MQMT_REPORT.

La valeur par défaut est égale à "".

replyToQueueManagerName
`public String replyToQueueManagerName`

Nom du gestionnaire de files d'attente à qui les messages de réponse ou de rapport doivent être envoyés.

La valeur par défaut est égale à "".

Si la valeur est égale à "" lors d'un appel de MQQueue.put(), le gestionnaire de files d'attente renseigne cette zone.

userId `public String userId`

Sous-ensemble du contexte d'identification du message, désignant l'utilisateur qui en est à l'origine.

La valeur par défaut est égale à "".

accountingToken
`public byte accountingToken[]`

Sous-ensemble du contexte d'identification du message, permettant à une application d'imputer correctement le travail effectué en conséquence de ce message.

La valeur par défaut est égale à "MQC.MQACT_NONE".

applicationIdData
`public String applicationIdData`

Sous-ensemble du contexte d'identification du message ; ces informations sont définies par la suite d'applications et peuvent être utilisées pour fournir des informations supplémentaires sur le message ou sur son origine.

La valeur par défaut est égale à "".

putApplicationType
`public int putApplicationType`

Type de l'application ayant inséré le message. Il peut s'agir d'une valeur système ou d'une valeur définie par l'utilisateur. Les valeurs suivantes sont définies par le système :

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

Par défaut, cette variable prend la valeur spéciale MQC.MQAT_NO_CONTEXT, qui indique qu'aucune information de contexte n'est présente dans le message.

putApplicationName
```
public String putApplicationName
```
Nom de l'application insérant le message. La valeur par défaut est égale à "".

putDateTime
```
public GregorianCalendar putDateTime
```
Date et heure d'insertion du message.

applicationOriginData
```
public String applicationOriginData
```
Informations définies par l'application et pouvant être utilisées pour fournir des informations complémentaires sur l'origine du message.

La valeur par défaut est égale à "".

groupId
```
public byte[] groupId
```
Chaîne d'octets indiquant à quel groupe de messages appartient le message physique.

La valeur par défaut est égale à "MQC.MQGI_NONE".

messageSequenceNumber
```
public int messageSequenceNumber
```
Numéro de séquence d'un message logique dans un groupe.

offset `public int offset`

Dans le cadre d'un message segmenté, décalage des données d'un message physique par rapport au début du message logique auquel il appartient.

messageFlags
```
public int messageFlags
```
Indicateurs de contrôle de la segmentation et de l'état d'un message.

originalLength
```
public int originalLength
```
Longueur originelle d'un message segmenté.

Constructeurs

MQMessage
```
public MQMessage()
```
Crée un message, avec des informations par défaut dans le descripteur de message et un tampon de message vide.

Méthodes

getTotalMessageLength
```
public int getTotalMessageLength()
```
Nombre total d'octets du message, stocké dans la file d'attente de laquelle le message a été extrait (ou a fait l'objet d'une tentative d'extraction). Lorsque la méthode MQQueue.get() échoue et qu'une erreur de message tronqué est signalée, cette méthode vous permet de connaître la taille globale du message dans la file d'attente.

Voir aussi «MQQueue.get» à la page 139.

getMessageLength

```
public int getMessageLength
```

Déclenche l'exception IOException.

Nombre d'octets de données de message dans cet objet MQMessage.

getDataLength

```
public int getDataLength()
```

Déclenche l'exception MQException.

Nombre d'octets de données de message restant à lire.

seek

```
public void seek(int pos)
```

Déclenche l'exception IOException.

Déplace le curseur à la position absolue *pos* dans le tampon de message. Les lectures (read) et écritures (write) commandées ensuite opéreront à partir de cette position dans le tampon.

Déclenche l'exception EOFException si la position pos est située hors de la plage définie par la longueur des données de message.

setDataOffset

```
public void setDataOffset(int offset)
```

Déclenche l'exception IOException.

Déplace le curseur à la position absolue *offset* dans le tampon de message. Cette méthode est synonyme de seek() ; elle est proposée par souci de compatibilité avec les autres API MQSeries.

getDataOffset

```
public int getDataOffset()
```

Déclenche l'exception IOException.

Renvoie la position en cours du curseur dans les données du message (le point à partir duquel la prochaine opération de lecture ou d'écriture aura lieu).

clearMessage

```
public void clearMessage()
```

Déclenche l'exception IOException.

Supprime toute donnée du tampon de message et remet le décalage à zéro.

getVersion

```
public int getVersion()
```

Renvoie le numéro de version de la structure en cours d'utilisation.

resizeBuffer

```
public void resizeBuffer(int size)
```

Déclenche l'exception IOException.

Indice fourni au nouvel objet MQMessage sur la taille du tampon probablement requis pour mener à bien les prochaines opérations d'extraction (get). Si le message contient actuellement des données, et si la nouvelle taille du tampon est inférieure à la taille du tampon actuel, les données du message sont tronquées.

readBoolean

```
public boolean readBoolean()
```

Déclenche l'exception IOException.

Lit un octet (signé) à la position en cours dans le tampon de message.

readChar

```
public char readChar()
```

Déclenche les exceptions IOException et EOFException.

Lit un caractère Unicode à la position en cours dans le tampon de message.

readDouble

```
public double readDouble()
```

Déclenche les exceptions IOException et EOFException.

Lit un double à la position en cours dans le tampon de message. La valeur de la variable membre encoding détermine le comportement de cette méthode.

Les valeurs MQC.MQENC_FLOAT_IEEE_NORMAL et MQC.MQENC_FLOAT_IEEE_REVERSED permettent de lire respectivement les doubles IEEE aux formats big-endian et little-endian.

La valeur MQC.MQENC_FLOAT_S390 permet de lire un nombre à virgule flottante au format System/390.

readFloat

```
public float readFloat()
```

Déclenche les exceptions IOException et EOFException.

Lit un nombre à virgule flottante à la position en cours dans le tampon de message. La valeur de la variable membre encoding détermine le comportement de cette méthode.

Les valeurs MQC.MQENC_FLOAT_IEEE_NORMAL et MQC.MQENC_FLOAT_IEEE_REVERSED permettent de lire respectivement les variables flottantes IEEE aux formats big-endian et little-endian.

La valeur MQC.MQENC_FLOAT_S390 permet de lire un nombre à virgule flottante au format System/390.

readFully

```
public void readFully(byte b[])
```

Déclenche les exceptions Exception et EOFException.

Charge les données du tampon de message dans le tableau d'octets `b`.

readFully

```
public void readFully(byte b[],
                      int off,
                      int len)
```

Déclenche les exceptions IOException et EOFException.

Charge les données du tampon de message dans *len* éléments du tableau d'octets *b*, en commençant au décalage *off*.

readInt

```
public int readInt()
```

Déclenche les exceptions IOException et EOFException.

Lit un entier à partir de la position en cours dans le tampon de message. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet de lire un entier big-endian, alors que la valeur MQC.MQENC_INTEGER_REVERSED permet de lire un entier little-endian.

readInt4

```
public int readInt4()
```

Déclenche les exceptions IOException et EOFException.

Synonyme de readInt(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

readLine

```
public String readLine()
```

Déclenche l'exception IOException.

Assure la conversion du code identifié par la variable characterSet en code Unicode, puis lit une ligne terminée par \n, \r, \r\n, ou EOF.

readLong

```
public long readLong()
```

Déclenche les exceptions IOException et EOFException.

Lit un entier long à la position en cours dans le tampon de message. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet de lire un entier long big-endian, et la valeur MQC.MQENC_INTEGER_REVERSED permet de lire un entier long little-endian.

readInt8

```
public long readInt8()
```

Déclenche les exceptions IOException et EOFException.

Synonyme de readLong(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

readObject

```
public Object readObject()
```

Déclenche les exceptions OptionalDataException, ClassNotFoundException et IOException.

Lit un objet dans le tampon de message. La classe de l'objet, la signature de la classe et la valeur des zones non provisoires et non statiques de la classe sont lues.

readShort

```
public short readShort()
```

Déclenche les exceptions IOException et EOFException.

readInt2

```
public short readInt2()
```

Déclenche les exceptions IOException et EOFException.

Synonyme de readShort(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

readUTF

```
public String readUTF()
```

Déclenche l'exception IOException.

Lit une chaîne UTF, précédée d'une zone de 2 octets, à la position en cours dans le tampon de message.

readUnsignedByte

```
public int readUnsignedByte()
```

Déclenche les exceptions IOException et EOFException.

Lit un octet non signé à la position en cours dans le tampon de message.

readUnsignedShort

```
public int readUnsignedShort()
```

Déclenche les exceptions IOException et EOFException.

Lit un entier court non signé à la position en cours dans le tampon de message. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet de lire un entier court non signé big-endian, et la valeur MQC.MQENC_INTEGER_REVERSED permet de lire un entier court non signé little-endian.

readUInt2

```
public int readUInt2()
```

Déclenche les exceptions IOException et EOFException.

Synonyme de readUnsignedShort(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

readString

```
public String readString(int length)
```

Déclenche les exceptions IOException et EOFException.

Lit une chaîne exprimée dans le jeu de codes indiqué par la variable membre characterSetRead, puis la convertit en code Unicode.

Arguments

length Nombre de caractères à lire (peut différer du nombre d'octets, car certains jeux de codes expriment chaque caractère en plusieurs octets).

readDecimal2

```
public short readDecimal2()
```

Déclenche les exceptions IOException et EOFException.

Lit un nombre décimal condensé sur 2 octets (-999..999). Le comportement de cette méthode dépend de la valeur de la variable encoding. La valeur MQC.MQENC_DECIMAL_NORMAL permet de lire un nombre décimal condensé big-endian, et la valeur MQC.MQENC_DECIMAL_REVERSED permet de lire un nombre décimal condensé little-endian.

readDecimal4

```
public int readDecimal4()
```

Déclenche les exceptions IOException et EOFException.

Lit un nombre décimal condensé sur 4 octets (-9999999..9999999). Le comportement de cette méthode dépend de la valeur de la variable encoding. La valeur MQC.MQENC_DECIMAL_NORMAL permet de lire un nombre décimal condensé big-endian, et la valeur MQC.MQENC_DECIMAL_REVERSED permet de lire un nombre décimal condensé little-endian.

readDecimal8

```
public long readDecimal8()
```

Déclenche les exceptions IOException et EOFException.

Lit un nombre décimal condensé sur 8 octets (entre -999999999999999 et 999999999999999). Le comportement de cette méthode dépend de la valeur de la variable encoding. La valeur MQC.MQENC_DECIMAL_NORMAL

permet de lire un nombre décimal condensé big-endian, et la valeur MQC.MQENC_DECIMAL_REVERSED permet de lire un nombre décimal condensé little-endian.

setVersion

```
public void setVersion(int version)
```

Spécifie quelle version de la structure doit être utilisée. Les valeurs possibles sont les suivantes :
- MQC.MQMD_VERSION_1
- MQC.MQMD_VERSION_2

Normalement, vous ne devriez pas avoir besoin d'appeler cette méthode, sauf si vous voulez forcer le client à utiliser une structure de version 1 lorsqu'il est connecté à un gestionnaire de files d'attente capable de gérer les structures de version 2. Dans toutes les autres situations, le client détermine la version de la structure à utiliser en interrogeant le gestionnaire de files d'attente.

skipBytes

```
public int skipBytes(int n)
```

Déclenche les exceptions IOException et EOFException.

Avance de n octets dans le tampon de message.

Cette méthode reste bloquée jusqu'à ce qu'un des événements suivants surviennent :
- Tous les octets sont ignorés
- La fin du tampon de message est détectée
- Une exception est déclenchée

Renvoie le nombre d'octets ignorés, qui est toujours n.

write

```
public void write(int b)
```

Déclenche l'exception IOException.

Ecrit un octet dans le tampon de message à la position en cours.

write

```
public void write(byte b[])
```

Déclenche l'exception IOException.

Ecrit un tableau d'octets dans le tampon de message à partir de la position en cours.

write

```
public void write(byte b[],
                  int off,
                  int len)
```

Déclenche l'exception IOException.

Ecrit une série d'octets dans le tampon de message à partir de la position en cours. *len* octets sont prélevés au décalage *off* dans le tableau *b* pour être écrits.

writeBoolean

```
public void writeBoolean(boolean v)
```

Déclenche l'exception IOException.

Ecrit une valeur booléenne dans le tampon de message à la position en cours.

writeByte

```
public void writeByte(int v)
```

Déclenche l'exception IOException.

Ecrit un octet dans le tampon de message à la position en cours.

writeBytes

```
public void writeBytes(String s)
```

Déclenche l'exception IOException.

Recopie la chaîne dans le tampon de message comme une séquence d'octets. Chaque caractère de la chaîne est recopié dans une séquence en éliminant ses huit bits de poids fort.

writeChar

```
public void writeChar(int v)
```

Déclenche l'exception IOException.

Ecrit un caractère Unicode dans le tampon de message à la position en cours.

writeChars

```
public void writeChars(String s)
```

Déclenche l'exception IOException.

Ecrit une chaîne sous forme de suite de caractères Unicode dans le tampon de message à partir de la position en cours.

writeDouble

```
public void writeDouble(double v)
```

Déclenche l'exception IOException.

Ecrit un double dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

Les valeurs MQC.MQENC_FLOAT_IEEE_NORMAL et MQC.MQENC_FLOAT_IEEE_REVERSED permettent respectivement d'écrire les variables flottantes IEEE aux formats big-endian et little-endian.

La valeur MQC.MQENC_FLOAT_S390 permet d'écrire un nombre à virgule flottante au format System/390. Notez que la plage des doubles IEEE est plus étendue que celle des nombres à virgule flottante double précision S/390, et que les très grands nombres ne peuvent donc pas être convertis.

writeFloat

```
public void writeFloat(float v)
```

Déclenche l'exception IOException.

Ecrit un nombre à virgule flottante dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

Les valeurs MQC.MQENC_FLOAT_IEEE_NORMAL et MQC.MQENC_FLOAT_IEEE_REVERSED permettent respectivement d'écrire les variables flottantes IEEE aux formats big-endian et little-endian.

La valeur MQC.MQENC_FLOAT_S390 permet d'écrire un nombre à virgule flottante au format System/390.

writeInt

```
public void writeInt(int v)
```

Déclenche l'exception IOException.

Ecrit un entier dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet d'écrire un entier big-endian, alors que la valeur MQC.MQENC_INTEGER_REVERSED permet d'écrire un entier little-endian.

writeInt4

```
public void writeInt4(int v)
```

Déclenche l'exception IOException.

Synonyme de writeInt(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

writeLong

```
public void writeLong(long v)
```

Déclenche l'exception IOException.

Ecrit un entier long dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet d'écrire un entier long big-endian, alors que la valeur MQC.MQENC_INTEGER_REVERSED permet d'écrire un entier long little-endian.

writeInt8

```
public void writeInt8(long v)
```

Déclenche l'exception IOException.

Synonyme de writeLong(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

writeObject

```
public void writeObject(Object obj)
```

Déclenche l'exception IOException.

Ecrit l'objet spécifié dans le tampon de message. La classe de l'objet, la signature de la classe, la valeur des zones non provisoires et non statiques de la classe et tous ses supertypes sont écrits.

writeShort

```
public void writeShort(int v)
```

Déclenche l'exception IOException.

Ecrit un entier court dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_INTEGER_NORMAL permet d'écrire un entier court big-endian, alors que la valeur MQC.MQENC_INTEGER_REVERSED permet d'écrire un entier court little-endian.

writeInt2

```
public void writeInt2(int v)
```

Déclenche l'exception IOException.

Synonyme de writeShort(), proposé par souci de compatibilité avec les API MQSeries d'autres langages.

writeDecimal2

```
public void writeDecimal2(short v)
```

Déclenche l'exception IOException.

Ecrit un nombre au format décimal condensé sur 2 octets dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_DECIMAL_NORMAL permet d'écrire un décimal condensé big-endian, alors que la valeur MQC.MQENC_DECIMAL_REVERSED permet d'écrire un décimal condensé little-endian.

Arguments

v doit se situer dans la plage -999 à 999.

writeDecimal4

```
public void writeDecimal4(int v)
```

Déclenche l'exception IOException.

Ecrit un nombre au format décimal condensé sur 4 octets dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_DECIMAL_NORMAL permet d'écrire un décimal condensé big-endian, alors que la valeur MQC.MQENC_DECIMAL_REVERSED permet d'écrire un décimal condensé little-endian.

Arguments

v doit se situer dans la plage -9999999 à 9999999.

writeDecimal8

```
public void writeDecimal8(long v)
```

Déclenche l'exception IOException.

Ecrit un nombre au format décimal condensé sur 8 octets dans le tampon de message à la position en cours. La valeur de la variable membre encoding détermine le comportement de cette méthode.

La valeur MQC.MQENC_DECIMAL_NORMAL permet d'écrire un décimal condensé big-endian, alors que la valeur MQC.MQENC_DECIMAL_REVERSED permet d'écrire un décimal condensé little-endian.

Arguments

v doit se situer dans la plage -999999999999999 à 999999999999999.

writeUTF

```
public void writeUTF(String str)
```

Déclenche l'exception IOException.

Ecrit une chaîne UTF, précédée d'une zone de 2 octets, à la position en cours dans le tampon de message.

writeString

```
public void writeString(String str)
```

Déclenche l'exception IOException.

Ecrit une chaîne dans le tampon de message à la position en cours, en la convertissant dans le jeu de codes indiqué par la variable membre characterSet.

MQMessageTracker

```
java.lang.Object
   |
   └─ com.ibm.mq.MQMessageTracker
```

public abstract class **MQMessageTracker**
développe **Object**

Remarque : Vous ne pouvez utiliser cette classe que lorsque vous êtes connecté à un gestionnaire de files d'attente MQSeries version 5 (ou ultérieure).

MQDistributionListItem (on page 91) hérite de cette classe pour personnaliser les paramètres d'un message liés à une destination donnée dans une liste de diffusion.

Variables

feedback

```
public int feedback
```

Cette variable est utilisée avec un message de type MQC.MQMT_REPORT pour préciser la nature du rapport. Les codes retour suivants sont définis par le système :

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Vous pouvez également utiliser des valeurs de code retour définies par l'application, situées dans la plage MQC.MQFB_APPL_FIRST à MQC.MQFB_APPL_LAST.

La valeur par défaut de cette zone est MQC.MQFB_NONE, ce qui signifie qu'aucun code retour n'est renvoyé.

messageId

```
public byte messageId[]
```

Cette variable indique l'ID de message à utiliser lors de l'insertion du message. Si c'est MQC.MQMI_NONE qui est spécifié, le gestionnaire de files d'attente génère un ID de message unique lors de l'insertion du message dans la file. La valeur de cette variable est mise à jour après l'insertion, pour indiquer l'ID de message qui a été utilisé.

La valeur par défaut est égale à MQC.MQMI_NONE.

correlationId

```
public byte correlationId[]
```

Cette variable indique l'ID de corrélation à utiliser lors de l'insertion du message.

La valeur par défaut est MQC.MQCI_NONE.

accountingToken

```
public byte accountingToken[]
```

Sous-ensemble du contexte d'identification du message. Cette variable permet à une application d'imputer correctement le travail effectué en conséquence de ce message.

La valeur par défaut est MQC.MQACT_NONE.

groupId

```
public byte[] groupId
```

Chaîne d'octets indiquant à quel groupe de messages appartient le message physique.

La valeur par défaut est MQC.MQGI_NONE.

MQPoolServices

```
java.lang.Object
   |
   └─ com.ibm.mq.MQPoolServices
```

public class **MQPoolServices**
extends **Object**

Remarque : Les applications n'utilisent généralement pas cette classe.

La classe MQPoolServices peut être utilisée par les mises en oeuvre de ConnectionManager conçues pour être utilisées comme le ConnectionManager par défaut pour les connexions MQSeries.

Un ConnectionManager peut construire un objet MQPoolServices et, grâce à lui, enregistrer un programme d'écoute. Ce programme d'écoute reçoit des événements liés à l'ensemble d'objets MQPoolTokens que gère MQEnvironment. Le ConnectionManager peut utiliser ces informations pour réaliser les travaux d'initialisation ou de nettoyage nécessaires.

Voir aussi «MQPoolServicesEvent» à la page 130 et «MQPoolServicesEventListener» à la page 160.

Constructeurs

MQPoolServices

```
public MQPoolServices()
```

Construit un nouvel objet MQPoolServices.

Méthodes

addMQPoolServicesEventListener

```
public void addMQPoolServicesEventListener
                                (MQPoolServicesEventListener listener)
```

Ajoute un MQPoolServicesEventListener. Le programme d'écoute reçoit un événement à chaque fois qu'un jeton est ajouté ou supprimé de l'ensemble d'objets MQPoolToken que contrôle MQEnvironment, ou à chaque fois que le ConnectionManager par défaut change.

removeMQPoolServicesEventListener

```
public void removeMQPoolServicesEventListener
                                   (MQPoolServicesEventListener listener)
```

Supprime un MQPoolServicesEventListener.

getTokenCount

```
public int getTokenCount()
```

Renvoie le nombre d'objets MQPoolToken enregistrés avec MQEnvironment.

MQPoolServicesEvent

```
java.lang.Object
   |
   +-- java.util.EventObject
          |
          +-- com.ibm.mq.MQPoolServicesEvent
```

Remarque : Les applications n'utilisent généralement pas cette classe.

Un MQPoolServicesEvent est généré à chaque fois qu'un MQPoolToken est ajouté, ou supprimé de l'ensemble de jetons que contrôle MQEnvironment. Un événement est également généré lorsque le ConnectionManager par défaut est modifié.

Voir aussi «MQPoolServices» à la page 129 et «MQPoolServicesEventListener» à la page 160.

Variables

TOKEN_ADDED
`public static final int TOKEN_ADDED`

ID d'événement utilisé lorsqu'un MQPoolToken est ajouté à l'ensemble.

TOKEN_REMOVED
`public static final int TOKEN_REMOVED`

ID d'événement utilisé lorsqu'un MQPoolToken est supprimé de l'ensemble.

DEFAULT_POOL_CHANGED
`public static final int DEFAULT_POOL_CHANGED`

ID d'événement utilisé lorsque le ConnectionManager par défaut change.

ID `protected int ID`

ID d'événement. Les valeurs possibles sont les suivantes :

TOKEN_ADDED
TOKEN_REMOVED
DEFAULT_POOL_CHANGED

token `protected MQPoolToken token`

Jeton. Lorsque l'ID d'événement est DEFAULT_POOL_CHANGED, la valeur de cette zone est NULL.

Constructeurs

MQPoolServicesEvent
`public MQPoolServicesEvent(Object source, int eid, MQPoolToken token)`

Construit un MQPoolServicesEvent basé sur l'ID d'événement et sur le jeton.

MQPoolServicesEvent
`public MQPoolServicesEvent(Object source, int eid)`

Construit un MQPoolServicesEvent basé sur l'ID d'événement.

Méthodes

getId `public int getId()`

Extrait l'ID d'événement.

Valeur renvoyée

ID d'événement, avec une des valeurs suivantes :

TOKEN_ADDED

TOKEN_REMOVED

DEFAULT_POOL_CHANGED

getToken

`public MQPoolToken getToken()`

Renvoie le jeton qui a été ajouté ou supprimé de l'ensemble. Lorsque l'ID d'événement est DEFAULT_POOL_CHANGED, la valeur de cette zone est NULL.

MQPoolToken

```
java.lang.Object
   |
   └─ com.ibm.mq.MQPoolToken
```

public class **MQPoolToken**
extends **Object**

Un MQPoolToken peut être utilisé pour activer le pool de connexion par défaut. Les MQPoolTokens sont enregistrés avec la classe MQEnvironment avant qu'un composant d'application établisse une connexion à MQSeries. Plus tard, ils sont désenregistrés lorsque le composant a fini d'utiliser MQSeries. Généralement, le ConnectionManager par défaut est actif tant que l'ensemble des objets MQPoolToken enregistrés n'est pas vide.

MQPoolToken ne fournit pas de méthodes ou de variables. Les fournisseurs de ConnectionManager peuvent choisir d'étendre MQPoolToken de façon à ce que des indices puissent être transmis au ConnectionManager.

Voir «MQEnvironment.addConnectionPoolToken» à la page 98 et «MQEnvironment.removeConnectionPoolToken» à la page 98.

Constructeurs

MQPoolToken

```
public MQPoolToken()
```

Construit un nouvel objet MQPoolToken.

MQProcess

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
            |
            +-- com.ibm.mq.MQProcess
```

public class **MQProcess**
extends **MQManagedObject**. (Voir page 105.)

MQProcess fournit des informations sur les processus MQSeries.

Constructeurs

MQProcess

```
public MQProcess(MQQueueManager qMgr,
                                 String processName,
                                 int openOptions,
                                 String queueManagerName,
                                 String alternateUserId)
                 throws MQException
```

Permet d'accéder à un processus du gestionnaire de files d'attente qMgr. Voir accessProcess dans la section«MQQueueManager» à la page 147 pour plus de détails sur les autres arguments.

Méthodes

getApplicationId

```
public String getApplicationId()
```

Chaîne de caractères identifiant l'application à lancer. Cette information est utilisée par une application de gestion des déclenchements qui traite les messages de la file d'attente d'initialisation ; l'information est envoyée à la file d'attente d'initialisation dans le cadre du message de déclenchement.

Déclenche l'exception MQException si vous l'appelez après avoir fermé le processus.

getApplicationType

```
public int getApplicationType()
```

Déclenche l'exception MQException (voir page 99).

Cette méthode permet d'identifier le type de programme à lancer sur réception d'un message de déclenchement. Elle accepte a priori n'importe quelle valeur, mais nous vous recommandons d'utiliser les valeurs suivantes pour les types standard.

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_WINDOWS_NT

- MQC.MQAT_USER_FIRST (plus petite valeur de type d'application utilisateur)
- MQC.MQAT_USER_LAST (plus grande valeur de type d'application utilisateur)

getEnvironmentData

```
public String getEnvironmentData()
```

Déclenche l'exception MQException.

Chaîne de caractères contenant des informations sur l'environnement de l'application qui doit être lancée.

getUserData

```
public String getUserData()
```

Déclenche l'exception MQException.

Chaîne de caractères contenant des informations utilisateur liées à l'application qui doit être lancée.

close

```
public synchronized void close()
```

Déclenche l'exception MQException.

Remplacement de «MQManagedObject.close» à la page 107.

MQPutMessageOptions

```
java.lang.Object
   |
   └─ com.ibm.mq.MQPutMessageOptions
```

public class **MQPutMessageOptions**
extends **Object**

Cette classe contient des options qui déterminent le comportement de MQQueue.put().

Remarque : Le comportement de certaines des options disponibles dans cette classe dépend de l'environnement dans lequel elles sont utilisées. Ces options sont signalées par un astérisque (*****). Pour plus de précisions, voir la section «Extensions de la version 5 fonctionnant dans d'autres environnements» à la page 79.

Variables

options

```
public int options
```

Options déterminant le comportement de la méthode MQQueue.put. Vous pouvez définir, une, plusieurs ou aucune des valeurs indiquées ci-dessous. Pour définir plusieurs options, vous pouvez additionner leurs valeurs ou les combiner à l'aide de l'opérateur bit OR.

MQC.MQPMO_SYNCPOINT
Insérer un message sous contrôle d'un point de synchronisation. Le message n'est pas visible en-dehors de l'unité de travail tant que celle-ci n'est pas validée. Si l'unité de travail est annulée, le message est supprimé.

MQC.MQPMO_NO_SYNCPOINT
Insérer un message sans contrôle de point de synchronisation. Notez que si l'option de contrôle de point de synchronisation n'est pas spécifiée, l'option 'no syncpoint' est prise par défaut. Cette option s'applique à toutes les plateformes prises en charge, y compris OS/390.

MQC.MQPMO_NO_CONTEXT
Il est inutile d'associer un contexte au message.

MQC.MQPMO_DEFAULT_CONTEXT
Associer un contexte par défaut au message.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Définir le contexte d'identification en fonction de l'application.

MQC.MQPMO_SET_ALL_CONTEXT
Définir la totalité du contexte en fonction de l'application.

MQC.MQPMO_FAIL_IF_QUIESCING
Provoquer une erreur si le gestionnaire de files d'attente est au repos.

MQC.MQPMO_NEW_MSG_ID*
Générer un nouvel ID de message pour chaque message envoyé.

MQC.MQPMO_NEW_CORREL_ID*
Générer un nouvel ID corrélation pour chaque message envoyé.

MQC.MQPMO_LOGICAL_ORDER*
Placer les messages logiques et les segments des groupes de messages dans leur ordre logique.

MQC.MQPMO_NONE
Pas d'option spécifiée. Ne pas utiliser en conjonction avec d'autres options.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Reprendre le contexte d'identification d'un identificateur de file d'attente d'entrée.

MQC.MQPMO_PASS_ALL_CONTEXT
Reprendre l'intégralité du contexte d'un identificateur de file d'attente d'entrée.

contextReference
`public MQQueue ContextReference`

Zone d'entrée indiquant la source des informations de contexte.

Si la zone `options` contient MQC.MQPMO_PASS_IDENTITY_CONTEXT ou MQC.MQPMO_PASS_ALL_CONTEXT, placez dans cette zone une référence à la file d'attente MQQueue de laquelle il faut tirer ces informations de contexte.

La valeur initiale de cette zone est la valeur NULL.

recordFields *
`public int recordFields`

Indicateurs désignant les zones des enregistrements de mise en file d'attente de messages à personnaliser lors de l'insertion d'un message dans une liste de diffusion. Vous pouvez spécifier un ou plusieurs des indicateurs suivants :

MQC.MQPMRF_MSG_ID
Utiliser l'attribut messageId de l'élément MQDistributionListItem.

MQC.MQPMRF_CORREL_ID
Utiliser l'attribut correlationId de l'élément MQDistributionListItem.

MQC.MQPMRF_GROUP_ID
Utiliser l'attribut groupId de l'élément MQDistributionListItem.

MQC.MQPMRF_FEEDBACK
Utiliser l'attribut feedback de l'élément MQDistributionListItem.

MQC.MQPMRF_ACCOUNTING_TOKEN
Utiliser l'attribut accountingToken de l'élément MQDistributionListItem.

La valeur spéciale MQC.MQPMRF_NONE signale qu'aucune zone ne doit être personnalisée.

resolvedQueueName
`public String resolvedQueueName`

Zone de sortie définie par le gestionnaire de files d'attente et contenant le nom de la file d'attente dans laquelle le message est inséré. Ce nom peut

être différent de celui utilisé pour ouvrir la file d'attente si l'opération d'ouverture portait sur une file d'attente alias ou modèle.

resolvedQueueManagerName

```
public String resolvedQueueManagerName
```

Zone de sortie définie par le gestionnaire de files d'attente et contenant le nom du gestionnaire de files d'attente propriétaire de la file d'attente spécifiée par le nom de file éloignée. Ce nom peut être différent de celui du gestionnaire de files d'attente à partir duquel l'accès à la file d'attente a été effectué, si cette file d'attente est éloignée.

knownDestCount *

```
public int knownDestCount
```

Zone de sortie définie par le gestionnaire de files d'attente et contenant le nombre de messages envoyés avec succès par l'appel en cours vers des files d'attente locales. Cette zone est également définie lorsque vous ouvrez une file d'attente unique ne faisant pas partie d'une liste de diffusion.

unknownDestCount *

```
public int unknownDestCount
```

Zone de sortie définie par le gestionnaire de files d'attente et contenant le nombre de messages envoyés avec succès par l'appel en cours vers des files d'attente éloignées. Cette zone est également définie lorsque vous ouvrez une file d'attente unique ne faisant pas partie d'une liste de diffusion.

invalidDestCount *

```
public int invalidDestCount
```

Zone de sortie définie par le gestionnaire de files d'attente et contenant le nombre de messages n'ayant pu être envoyés vers des files d'attente d'une liste de diffusion. Ce nombre comprend à la fois les files d'attente qui n'ont pas pu être ouvertes et celles qui ont pu être ouvertes mais dans lesquelles l'opération d'insertion a échoué. Cette zone est également définie lorsque vous ouvrez une file d'attente unique ne faisant pas partie d'une liste de diffusion.

Constructeurs

MQPutMessageOptions

```
public MQPutMessageOptions()
```

Permet de construire un nouvel objet MQPutMessageOptions sans aucune option définie et avec des zones resolvedQueueName et resolvedQueueManagerName vides.

MQQueue

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueue
```

public class **MQQueue**
extends **MQManagedObject**. (Voir page 105.)

MQQueue permet d'effectuer des opérations d'interrogation, de définition, d'insertion et d'extraction dans les files d'attente MQSeries. Les fonctionnalités d'interrogation et de définition sont héritées de MQ.MQManagedObject.

Voir aussi la section «MQQueueManager.accessQueue» à la page 152.

Constructeurs

MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName, String dynamicQueueName,
               String alternateUserId)
               throws MQException
```

Permet d'accéder à une file d'attente gérée par le gestionnaire de files `qMgr`.

Pour plus de précisions sur les autres arguments, voir la section «MQQueueManager.accessQueue» à la page 152.

Méthodes

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions,
                             int MaxMsgSize)
```

Déclenche l'exception MQException.

Extrait un message de la file d'attente, à concurrence de la taille maximale de message spécifiée.

Cette méthode utilise un objet MQMessage comme argument. Elle tient compte de la valeur de certaines zones de cet objet, notamment messageId et correlationId, c'est pourquoi vous devez vous assurer que ces valeurs sont correctes. (Voir «Message» à la page 272.)

Si l'extraction échoue, l'objet MQMessage reste inchangé. En cas de succès, le descripteur du message MQMessage (ses variables) et ses données sont remplacés par ceux du message entrant.

Notez que tous les appels à MQSeries provenant d'un gestionnaire MQQueueManager donné sont synchrones. Par conséquent, si vous effectuez une extraction avec attente, aucune autre unité d'exécution utilisant le même gestionnaire de files d'attente ne peut placer d'autre appel MQSeries tant que votre 'get' n'est pas terminé. Si vous avez besoin

de plusieurs unités d'exécution pour effectuer plusieurs accès simultanés à MQSeries, chaque unité d'exécution doit créer son propre objet MQQueueManager.

Arguments

message
Argument d'entrée et de sortie contenant les données du descripteur de message et les données de message renvoyées.

getMessageOptions
Options de paramétrage de l'extraction. (Voir la section «MQGetMessageOptions» à la page 101.)

MaxMsgSize
Taille maximale des messages pouvant être reçus lors de cet appel. Si le message qui se trouve dans la file d'attente dépasse cette taille maximale, il peut se produire l'une des deux choses suivantes :

1. Si l'indicateur MQC.MQGMO_ACCEPT_TRUNCATED_MSG est défini dans la variable membre options de l'objet MQGetMessageOptions, la plus grande partie possible du message est chargée dans le tampon, et une exception avec code achèvement MQException.MQCC_WARNING et code anomalie MQException.MQRC_TRUNCATED_MSG_ACCEPTED est déclenchée.
2. Si l'indicateur MQC.MQGMO_ACCEPT_TRUNCATED_MSG n'est pas défini, le message est laissé dans la file d'attente, et une exception MQException avec code achèvement MQException.MQCC_WARNING et code anomalie MQException.MQRC_TRUNCATED_MSG_FAILED est déclenchée.

Déclenche l'exception MQException en cas d'échec de l'extraction.

get

```
public synchronized void get(MQMessage message,
                             MQGetMessageOptions getMessageOptions)
```

Déclenche l'exception MQException.

Extrait un message de la file d'attente, quelle que soit la taille de ce message. Lorsqu'il s'agit de messages très volumineux, il est possible que la méthode get effectue deux appels à MQSeries pour votre compte, le premier pour établir la taille de tampon requise, et le second pour charger les données du message.

Cette méthode prend un objet MQMessage comme argument. Elle tient compte de la valeur de certaines zones de cet objet, notamment messageId et correlationId, c'est pourquoi vous devez vous assurer que ces valeurs sont correctes. (Voir «Message» à la page 272.)

Si l'extraction échoue, l'objet MQMessage reste inchangé. Si elle réussit, le descripteur du message (ses variables) et ses données sont remplacés par ceux du message entrant.

Notez que tous les appels à MQSeries provenant d'un gestionnaire MQQueueManager donné sont synchrones. Par conséquent, si vous effectuez une extraction avec attente, aucune autre unité d'exécution utilisant le même gestionnaire de files d'attente ne peut placer d'autre appel MQSeries tant que votre 'get' n'est pas terminé. Si vous avez besoin de plusieurs unités d'exécution pour effectuer plusieurs accès simultanés à MQSeries, chaque unité d'exécution doit créer son propre objet MQQueueManager.

Arguments

message
Argument d'entrée et de sortie contenant les données du descripteur de message et les données de message renvoyées.

getMessageOptions
Options de paramétrage de l'extraction. (Pour plus de précisions, voir la section «MQGetMessageOptions» à la page 101.)

Déclenche l'exception MQException en cas d'échec de l'extraction.

get

```
public synchronized void get(MQMessage message)
```

Version simplifiée de la méthode get décrite précédemment.

Arguments

MQMessage
Argument d'entrée et de sortie contenant les données du descripteur de message et les données de message renvoyées.

Cette méthode fait appel à une instance par défaut de MQGETMessageOptions pour effectuer l'extraction. L'option de message utilisée est MQGMO_NOWAIT.

put

```
public synchronized void put(MQMessage message,
                             MQPutMessageOptions putMessageOptions)
```

Déclenche l'exception MQException.

Insère un message dans la file d'attente.

Cette méthode prend un objet MQMessage comme argument. Il est possible que, suite à son exécution, les propriétés du descripteur de message soient modifiées. Les valeurs de ces propriétés après l'exécution sont celles qui ont été insérées dans la file d'attente MQSeries.

Toute modification apportée à l'objet MQMessage après l'exécution de l'insertion (put) n'est pas répercutée sur le message placé dans la file d'attente MQSeries.

L'insertion met à jour les valeurs messageId et correlationId. Vous devez en tenir compte si vous effectuez d'autres appels d'insertion/extraction en utilisant le même objet MQMessage. Notez aussi que l'appel de la méthode put ne réinitialise pas les données du message.

Exemple :

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

Ces lignes de code insèrent deux messages. Le premier contient «a» et le second contient «ab».

Arguments

message
Tampon de message contenant le descripteur et les données du message à envoyer.

putMessageOptions
Options de paramétrage de l'insertion. (Voir la section «MQPutMessageOptions» à la page 135)

Déclenche l'exception MQException en cas d'échec de l'insertion.

put

```
public synchronized void put(MQMessage message)
```

Version simplifiée de la méthode put décrite précédemment.

Arguments

MQMessage
Tampon de message contenant le descripteur et les données du message à envoyer.

Cette méthode fait appel à une instance par défaut de MQPutMessageOptions pour effectuer l'insertion.

Remarque : Toutes les méthodes qui suivent déclenchent l'exception MQException si vous appelez la méthode après avoir fermé la file d'attente.

getCreationDateTime

```
public GregorianCalendar getCreationDateTime()
```

Déclenche l'exception MQException.

Date et heure de création de cette file d'attente.

getQueueType

```
public int getQueueType()
```

Déclenche l'exceptionMQException.

Valeur renvoyée
Type de cette file d'attente, exprimé par l'une des valeurs suivantes :
- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

getCurrentDepth

```
public int getCurrentDepth()
```

Déclenche l'exception MQException.

Lit le nombre de messages qui se trouvent dans la file d'attente. Cette valeur augmente d'une unité à chaque appel de la méthode put et chaque fois qu'un appel à la méthode get est annulé. Elle diminue d'une unité à chaque appel de la méthode get (extractive) et chaque fois qu'un appel de la méthode put est annulé.

getDefinitionType

```
public int getDefinitionType()
```

Déclenche l'exception MQException.

Indique de quelle façon la file d'attente a été définie.

Valeur renvoyée

Une des valeurs suivantes :

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

getMaximumDepth

```
public int getMaximumDepth()
```

Déclenche l'exception MQException.

Nombre maximal de messages pouvant coexister dans la file d'attente. Toute tentative d'insertion de message dans une file d'attente contenant déjà ce nombre maximal de messages échoue avec le code anomalie MQException.MQRC_Q_FULL.

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

Déclenche l'exception MQException.

Longueur maximale des données applicatives d'un message de cette file d'attente. Toute tentative d'insertion d'un message contenant plus de données que ce maximum échoue avec le code anomalie MQException.MQRC_MSG_TOO_BIG_FOR_Q.

getOpenInputCount

```
public int getOpenInputCount()
```

Déclenche l'exception MQException.

Nombre d'identificateurs valides représentant des messages pouvant être supprimés de la file d'attente. Il s'agit du nombre *total* d'identificateurs connus du gestionnaire de files d'attente local et non uniquement des identificateurs créés par les classes MQSeries pour Java (via accessQueue).

getOpenOutputCount

```
public int getOpenOutputCount()
```

Déclenche l'exception MQException.

Nombre d'identificateurs valides représentant des messages pouvant être ajoutés à la file d'attente. Il s'agit du nombre *total* d'identificateurs connus du gestionnaire de files d'attente local et non uniquement des identificateurs créés par les classes MQSeries pour Java (via accessQueue).

getShareability

```
public int getShareability()
```

Déclenche l'exception MQException.

Indique si la file d'attente peut être ouverte plusieurs fois en entrée.

Valeur renvoyée

Une des valeurs suivantes :
- MQC.MQQA_SHAREABLE (ouverture multiple possible)
- MQC.MQQA_NOT_SHAREABLE (ouverture multiple impossible)

getInhibitPut

```
public int getInhibitPut()
```

Déclenche l'exception MQException.

Indique si les opérations d'insertion dans cette file d'attente sont autorisées ou non.

Valeur renvoyée

Une des valeurs suivantes :
- MQC.MQQA_PUT_INHIBITED (interdites)
- MQC.MQQA_PUT_ALLOWED (autorisées)

setInhibitPut

```
public void setInhibitPut(int inhibit)
```

Déclenche l'exception MQException.

Permet de définir si les opérations d'insertion dans cette file d'attente sont autorisées ou non. Les valeurs possibles sont les suivantes :
- MQC.MQQA_PUT_INHIBITED (interdites)
- MQC.MQQA_PUT_ALLOWED (autorisées)

getInhibitGet

```
public int getInhibitGet()
```

Déclenche l'exception MQException.

Indique si les opérations d'extraction de cette file d'attente sont autorisées ou non.

Valeur renvoyée

Les valeurs possibles sont les suivantes :
- MQC.MQQA_GET_INHIBITED (interdites)
- MQC.MQQA_GET_ALLOWED (autorisées)

setInhibitGet

```
public void setInhibitGet(int inhibit)
```

Déclenche l'exception MQException.

Permet de définir si les opérations d'extraction de cette file d'attente sont autorisées ou non. Les valeurs possibles sont les suivantes :
- MQC.MQQA_GET_INHIBITED (interdites)
- MQC.MQQA_GET_ALLOWED (autorisées)

getTriggerControl

```
public int getTriggerControl()
```

Déclenche l'exception MQException.

Indique si oui ou non des messages de déclenchement sont placés dans une file d'attente d'initialisation pour lancer une application de traitement de messages.

Valeur renvoyée

Les valeurs possibles sont les suivantes :
- MQC.MQTC_OFF (non)
- MQC.MQTC_ON (oui)

setTriggerControl

```
public void setTriggerControl(int trigger)
```

Déclenche l'exception MQException.

Permet de définir si oui ou non des messages de déclenchement sont placés dans une file d'attente d'initialisation pour lancer une application de traitement de messages. Les valeurs possibles sont les suivantes :
- MQC.MQTC_OFF (non)
- MQC.MQTC_ON (oui)

getTriggerData

```
public String getTriggerData()
```

Déclenche l'exception MQException.

Données de format libre ajoutées par le gestionnaire de files d'attente dans le message de déclenchement lorsque l'insertion d'un nouveau message dans la file d'attente entraîne l'écriture d'un message de déclenchement dans la file d'attente d'initialisation.

setTriggerData

```
public void setTriggerData(String data)
```

Déclenche l'exception MQException.

Permet de définir les données de format libre insérées par le gestionnaire de files d'attente dans le message de déclenchement lorsque l'arrivée d'un nouveau message dans la file d'attente entraîne l'écriture d'un message de déclenchement dans la file d'attente d'initialisation. La longueur maximale de la chaîne est précisée par MQC.MQ_TRIGGER_DATA_LENGTH.

getTriggerDepth

```
public int getTriggerDepth()
```

Déclenche l'exception MQException.

Nombre de messages devant se trouver dans la file d'attente pour qu'un message de déclenchement soit écrit lorsque le type de déclenchement est égal à MQC.MQTT_DEPTH.

setTriggerDepth

```
public void setTriggerDepth(int depth)
```

Déclenche l'exception MQException.

Permet de définir le nombre de messages devant se trouver dans la file d'attente pour qu'un message de déclenchement soit écrit lorsque le type de déclenchement est égal à MQC.MQTT_DEPTH.

getTriggerMessagePriority

```
public int getTriggerMessagePriority()
```

Déclenche l'exception MQException.

Niveau de priorité en dessous duquel un message ne contribue pas à la création de messages de déclenchement (en d'autres termes, le gestionnaire de files d'attente ignore ces messages lorsqu'il détermine s'il doit générer un message de déclenchement). La valeur zéro signifie que tous les messages contribuent à la création de messages de déclenchement.

setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority)
```

Déclenche l'exception MQException.

Permet de définir le niveau de priorité en dessous duquel un message ne contribue pas à la création de messages de déclenchement (en d'autres termes, le gestionnaire de files d'attente ignore ces messages lorsqu'il détermine s'il doit générer un message de déclenchement). La valeur zéro signifie que tous les messages contribuent à la création de messages de déclenchement.

getTriggerType

```
public int getTriggerType()
```

Déclenche l'exception MQException.

Critère d'écriture d'un message de déclenchement en fonction de l'arrivée de nouveaux messages dans la file d'attente.

Valeur renvoyée

Les valeurs possibles sont les suivantes :

- MQC.MQTT_NONE
- MQC.MQTT_FIRST (au premier message)
- MQC.MQTT_EVERY (à chaque message)
- MQC.MQTT_DEPTH

setTriggerType

```
public void setTriggerType(int type)
```

Déclenche l'exception MQException.

Permet de définir le critère d'écriture d'un message de déclenchement en fonction de l'arrivée de nouveaux messages dans la file d'attente. Les valeurs possibles sont les suivantes :

- MQC.MQTT_NONE
- MQC.MQTT_FIRST (au premier message)
- MQC.MQTT_EVERY (à chaque message)
- MQC.MQTT_DEPTH

close

```
public synchronized void close()
```

Déclenche l'exception MQException.

Remplacement de «MQManagedObject.close» à la page 107.

MQQueueManager

```
java.lang.Object
   |
   +-- com.ibm.mq.MQManagedObject
          |
          +-- com.ibm.mq.MQQueueManager
```

public class **MQQueueManager**
extends **MQManagedObject**. (Voir page 105.)

Remarque : Le comportement de certaines des options disponibles dans cette classe dépend de l'environnement dans lequel elles sont utilisées. Ces options sont signalées par un astérisque (*****). Pour plus de précisions, reportez-vous au «Chapitre 8. Comportement selon l'environnement» à la page 75.

Variables

isConnected

```
public boolean isConnected
```

Vraie si la connexion au gestionnaire de files d'attente est toujours ouverte.

Constructeurs

MQQueueManager

```
public MQQueueManager(String queueManagerName)
```

Déclenche l'exception MQException.

Crée une connexion au gestionnaire de files d'attente nommé.

Remarque : Lorsque vous utilisez les classes MQSeries pour Java, le nom d'hôte, le nom de canal et le numéro de port à utiliser pour la connexion sont spécifiés dans la classe MQEnvironment. Vous devez les spécifier *avant* d'appeler ce constructeur.

L'exemple ci-dessous illustre l'établissement d'une connexion au gestionnaire de files d'attentes ″MYQM″, hébergé par une machine dont le nom d'hôte est fred.mq.com.

```
MQEnvironment.hostname = "fred.mq.com";  // Hôte concerné.
MQEnvironment.port     = 1414;           // Port auquel se connecter.
                                         // En cas d'omission, la
                                         // valeur par défaut est
                                         // 1414.
MQEnvironment.channel  = "channel.name"; // Nom du canal SVR CONN
                                         // sur le gest. de files ;
                                         // attention, distinction
                                         // minuscules/MAJUSCULES.
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

Si le nom du gestionnaire de files d'attente n'est pas précisé (nul ou ″″), la connexion est établie avec le gestionnaire de files d'attente par défaut.

Voir aussi «MQEnvironment» à la page 93.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur se connecte au gestionnaire de files d'attente spécifié, en utilisant les propriétés de MQEnvironment. Le MQConnectionManager spécifié gère la connexion.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur se connecte au gestionnaire de files d'attente spécifié, en utilisant les propriétés de MQEnvironment. Le ConnectionManager spécifié gère la connexion.

Cette méthode nécessite un JVM au niveau Java 2 version 1.3 ou suivante, avec JAAS version 1.0 ou suivante installé.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options)
```

Déclenche l'exception MQException.

Cette version du constructeur est conçue pour une utilisation en connexion directe uniquement et elle fait appel à l'API de connexion étendue (MQCONNX) pour établir la connexion au gestionnaire de files d'attente. L'argument *options* vous permet de sélectionner des liaisons rapides ou normales. Les valeurs possibles sont les suivantes :

- MQC.MQCNO_FASTPATH_BINDING pour des liaisons rapides *****.
- MQC.MQCNO_STANDARD_BINDING pour des liaisons normales

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      MQConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur exécute une instance de MQCONNX, en transmettant les options fournies. Le MQConnectionManager spécifié gère la connexion.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      int options,
                      ConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur exécute une instance de MQCONNX, en transmettant les options fournies. Le ConnectionManager spécifié gère la connexion.

Cette méthode nécessite un JVM au niveau Java 2 version 1.3 ou suivante, avec JAAS version 1.0 ou suivante installé.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      java.util.Hashtable properties)
```

L'argument properties se compose d'une série de paires clé/valeur décrivant l'environnement MQSeries pour ce gestionnaire de files d'attente. Ces propriétés, lorsqu'elles sont spécifiées, ont la priorité sur les valeurs définies dans la classe MQEnvironment ; elles permettent donc de personnaliser les attributs de chaque gestionnaire de files d'attente si nécessaire. Voir «″MQEnvironment.properties″» à la page 95.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      MQConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur se connecte au gestionnaire de files d'attente nommé, en utilisant les tables de hachage fournies dans les propriétés pour remplacer celles de MQEnvironment. Le MQConnectionManager spécifié gère la connexion.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      Hashtable properties,
                      ConnectionManager cxManager)
```

Déclenche l'exception MQException.

Ce constructeur se connecte au gestionnaire de files d'attente nommé, en utilisant les tables de hachage fournies dans les propriétés pour remplacer celles de MQEnvironment. Le ConnectionManager spécifié gère la connexion.

Cette méthode nécessite un JVM au niveau Java 2 version 1.3 ou suivante, avec JAAS version 1.0 ou suivante installé.

Méthodes

getCharacterSet

```
public int getCharacterSet()
```

Déclenche l'exception MQException.

Renvoie le CCSID (*Coded Character Set Identifier* ou ID de jeu de caractères) utilisé par le gestionnaire de files d'attente. Il s'agit du jeu de caractères utilisé par le gestionnaire de files d'attente pour traiter les chaînes de caractères transitant par l'API.

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

getMaximumMessageLength

```
public int getMaximumMessageLength()
```

Déclenche l'exception MQException.

Renvoie la longueur maximale des message (en octets) que peut traiter le gestionnaire de files d'attente. Il est impossible de définir une file d'attente dont la longueur maximale de message soit supérieure à cette valeur.

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

getCommandLevel

```
public int getCommandLevel()
```

Déclenche l'exception MQException.

Indique le niveau de prise en charge des commandes système par le gestionnaire de files d'attente. L'ensemble de commandes système correspondant à un niveau donné varie selon l'architecture de la plateforme sur laquelle est exécuté le gestionnaire de files d'attente. Pour plus de précisions, consultez la documentation MQSeries conçue pour votre plateforme.

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

Valeur renvoyée

Une des constantes MQC.MQCMDL_LEVEL_xxx

getCommandInputQueueName

```
public String getCommandInputQueueName()
```

Déclenche l'exception MQException.

Renvoie le nom de la file d'attente d'entrée de commandes définie pour le gestionnaire de files d'attentes. Il s'agit d'une file d'attente à laquelle les applications peuvent envoyer des commandes, si elles en ont le droit.

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

getMaximumPriority

```
public int getMaximumPriority()
```

Déclenche l'exception MQException.

Renvoie le niveau de priorité de message le plus élevé pris en charge par le gestionnaire de files d'attente. Les niveaux de priorité vont de zéro (niveau le plus bas) à cette valeur-ci.

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

getSyncpointAvailability

```
public int getSyncpointAvailability()
```

Déclenche l'exception MQException.

Indique si le gestionnaire de files d'attente prend en charge les unités de travail et les points de synchronisation avec les méthodes MQQueue.get et MQQueue.put.

Valeur renvoyée
- MQC.MQSP_AVAILABLE si les points de synchronisation sont disponibles
- MQC.MQSP_NOT_AVAILABLE si les points de synchronisation ne sont pas disponibles

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

getDistributionListCapable

```
public boolean getDistributionListCapable()
```

Indique si le gestionnaire de files d'attente prend en charge les listes de diffusion.

disconnect

```
public synchronized void disconnect()
```

Déclenche l'exception MQException.

Met fin à la connexion avec le gestionnaire de files d'attente. Tous les processus et files d'attente ouverts gérés par ce gestionnaire sont refermés et deviennent inutilisables. Lorsque vous êtes déconnecté d'un gestionnaire de files d'attente, la seule manière de vous y reconnecter consiste à créer un nouvel objet MQQueueManager.

Tous les travaux réalisés dans le cadre d'une unité de travail sont normalement validés. Toutefois, si cette connexion est gérée par un ConnectionManager, plutôt que par un MQConnectionManager, l'unité de travail peut être annulée.

commit

```
public synchronized void commit()
```

Déclenche l'exception MQException.

L'appel de cette méthode indique au gestionnaire de files d'attentes que l'application a atteint un point de synchronisation, et que toutes les extractions et les insertions de messages qui ont eu lieu depuis le point de synchronisation précédent doivent être validées de manière permanente. Les messages insérés dans le cadre d'une unité de travail (l'indicateur MQC.MQPMO_SYNCPOINT étant défini dans la zone options de MQPutMessageOptions) sont mis à la disposition des autres applications. Les messages extraits dans le cadre d'une unité de travail (l'indicateur MQC.MQGMO_SYNCPOINT étant défini dans la zone options de MQGetMessageOptions) sont supprimés.

Voir aussi la description de la méthode «backout» ci-dessous.

backout

```
public synchronized void backout()
```

Déclenche l'exception MQException.

L'appel de cette méthode indique au gestionnaire de files d'attentes que toutes les extractions et les insertions de messages qui ont eu lieu depuis le dernier point de synchronisation doivent être annulées. Les messages insérés dans le cadre d'une unité de travail (l'indicateur MQC.MQPMO_SYNCPOINT étant défini dans la zone options de MQPutMessageOptions) sont supprimés ; les messages extraits dans le cadre d'une unité de travail (l'indicateur MQC.MQGMO_SYNCPOINT étant défini dans la zone options de MQGetMessageOptions) sont replacés dans la file d'attente.

Voir aussi la description de la méthode «commit» ci-dessus.

accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName, int openOptions,
                        String queueManagerName,
                        String dynamicQueueName,
                        String alternateUserId
                        )
```

Déclenche l'exception MQException.

Donne accès à une file d'attente MQSeries de ce gestionnaire de files d'attente, dans le but d'extraire ou de parcourir des messages, d'insérer des messages, de lire ou de définir les attributs de la file d'attente.

Si la file d'attente nommée est un modèle de file d'attente, une file d'attente locale dynamique est créée. Le nom de la file d'attente créée peut être défini en vérifiant l'attribut `name` de l'objet MQQueue renvoyé.

Arguments

queueName
: Nom de la file d'attente à ouvrir.

openOptions
: Options relatives à l'ouverture de la file d'attente. Les options possibles sont les suivantes :

MQC.MQOO_BROWSE
: Ouvrir pour parcourir un message.

MQC.MQOO_INPUT_AS_Q_DEF
: Ouvrir pour extraire des messages en utilisant les paramètres par défaut de la file d'attente.

MQC.MQOO_INPUT_SHARED
: Ouvrir pour extraire des messages en accès partagé.

MQC.MQOO_INPUT_EXCLUSIVE
: Ouvrir pour extraire des messages en accès exclusif.

MQC.MQOO_OUTPUT
: Ouvrir pour insérer des messages.

MQC.MQOO_INQUIRE
: Ouvrir pour interrogation ; option requise si vous voulez lire des propriétés.

MQC.MQOO_SET
: Ouvrir pour définir des attributs.

MQC.MQOO_SAVE_ALL_CONTEXT
Sauvegarder le contexte lors de l'extraction d'un message *****.

MQC.MQOO_SET_IDENTITY_CONTEXT
Permet de définir le contexte d'identification.

MQC.MQOO_SET_ALL_CONTEXT
Permet de définir l'intégralité du contexte.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Valider à l'aide de l'ID utilisateur spécifié.

MQC.MQOO_FAIL_IF_QUIESCING
Provoquer un échec si le gestionnaire de files d'attente est au repos.

MQC.MQOO_BIND_AS_QDEF
Utiliser la liaison par défaut pour la file d'attente.

MQC.MQOO_BIND_ON_OPEN
Lier un identificateur à la destination lors de l'ouverture de la file d'attente.

MQC.MQOO_BIND_NOT_FIXED
Ne pas faire de liaison avec une destination spécifique.

MQC.MQOO_PASS_ALL_CONTEXT
Permet la transmission de l'intégralité du contexte.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Permet la transmission du contexte d'identification.

Pour définir plusieurs options, vous pouvez additionner leurs valeurs ou les combiner à l'aide de l'opérateur bit OR. Consultez le manuel *MQSeries Application Programming Reference* pour une description plus complète de ces options.

queueManagerName
Nom du gestionnaire de files d'attente sur lequel la file est définie. Si cette zone est vide, ou contient la valeur NULL, elle désigne le gestionnaire de files d'attente auquel l'objet MQQueueManager est connecté.

dynamicQueueName
Cet argument est ignoré, sauf si queueName contient le nom d'un modèle de file d'attente. Dans ce cas, cet argument indique le nom de la file d'attente dynamique à créer. Si queueName contient un nom de modèle de file d'attente, cette zone ne peut pas être vide ou avoir la valeur NULL. Si le dernier caractère (autre qu'un espace) de ce nom est un astérisque (*), le gestionnaire de files d'attente remplace cet astérisque par une chaîne de caractères qui assure l'unicité du nom de la file d'attente créée sur ce gestionnaire.

alternateUserId
Si l'option MQOO_ALTERNATE_USER_AUTHORITY est spécifiée dans l'argument openOptions, cet argument précise l'ID utilisateur secondaire à utiliser pour obtenir l'autorisation d'ouverture. Si l'option MQOO_ALTERNATE_USER_AUTHORITY n'est pas spécifiée, cet argument peut être laissé à blanc (ou nul).

Valeur renvoyée

Objet MQQueue dont l'ouverture a abouti.

Déclenche l'exception MQException en cas d'échec de l'ouverture.

Voir aussi «″accessProcess″».

accessQueue

```
public synchronized MQQueue accessQueue
                        (
                        String queueName,
                        int openOptions
                        )
```

Cette méthode déclenche l'exception MQException si vous l'appelez après déconnexion du gestionnaire de files d'attente.

Arguments

queueName

Nom de la file d'attente à ouvrir

openOptions

Options qui contrôlent l'ouverture de la file d'attente.

Pour plus de précisions sur les arguments, voir «MQQueueManager.accessQueue» à la page 152.

queueManagerName, dynamicQueueName, et *alternateUserId* se voient attribuer la valeur *″″*.

accessProcess

```
public synchronized MQProcess accessProcess
                       (
                       String processName,
                       int openOptions,
                       String queueManagerName,
                       String alternateUserId
                       )
```

Déclenche l'exception MQException.

Donne accès à un processus MQSeries de ce gestionnaire de files d'attente en vue d'interroger les attributs de ce processus.

Arguments

processName

Nom du processus à ouvrir.

openOptions

Options qui contrôlent l'ouverture du processus. L'interrogation (*Inquire*) est automatiquement ajoutée aux options spécifiées, et vous n'avez donc pas besoin de la spécifier explicitement.

Les options possibles sont les suivantes :

MQC.MQOO_ALTERNATE_USER_AUTHORITY

Valider à l'aide de l'ID utilisateur spécifié.

MQC.MQOO_FAIL_IF_QUIESCING
Provoquer un échec si le gestionnaire de files d'attente est au repos.

Pour définir plusieurs options, vous pouvez additionner leurs valeurs ou les combiner à l'aide de l'opérateur bit OR. Consultez le manuel *MQSeries Application Programming Reference* pour une description plus complète de ces options.

queueManagerName
Nom du gestionnaire de files d'attente sur lequel le processus est défini. Les applications doivent laisser cet argument à blanc (ou nul).

alternateUserId
Si l'option MQOO_ALTERNATE_USER_AUTHORITY est spécifiée dans l'argument openOptions, cet argument précise l'ID utilisateur secondaire à utiliser pour obtenir l'autorisation d'ouverture. Si MQOO_ALTERNATE_USER_AUTHORITY n'est pas spécifié, cet argument peut être laissé à blanc (ou nul).

Valeur renvoyée
Objet MQProcess dont l'ouverture a abouti.

Déclenche l'exception MQException en cas d'échec de l'ouverture.

Voir aussi «MQQueueManager.accessQueue» à la page 152.

accessProcess

Version simplifiée de la méthode AccessProcess décrite précédemment.

```
public synchronized MQProcess accessProcess
                      (
                     String processName,
                     int openOptions
                     )
```

Version simplifiée de la méthode AccessQueue décrite précédemment.

Arguments

processName
Nom du processus à ouvrir.

openOptions
Options qui contrôlent l'ouverture du processus.

Pour plus de précisions sur les options, voir «″accessProcess″» à la page 154.

queueManagerName et *alternateUserId* se voient attribuer la valeur ″″.

accessDistributionList

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems, int openOptions,
           String alternateUserId
          )
```

Déclenche l'exception MQException.

Arguments

litems Eléments à inclure dans la liste de diffusion.

openOptions
Options qui contrôlent l'ouverture de la liste de diffusion.

alternateUserId
Si l'option MQOO_ALTERNATE_USER_AUTHORITY est spécifiée dans l'argument openOptions, cet argument précise l'ID utilisateur secondaire à utiliser pour obtenir l'autorisation d'ouverture. Si l'option MQOO_ALTERNATE_USER_AUTHORITY n'est pas spécifiée, cet argument peut être laissé à blanc (ou nul).

Valeur renvoyée
Un nouvel objet MQDistributionList ouvert et prêt pour les opérations d'insertion.

Déclenche l'exception MQException en cas d'échec de l'ouverture.

Voir aussi «MQQueueManager.accessQueue» à la page 152.

accessDistributionList

Version simplifiée de la méthode AccessDistributionList décrite précédemment.

```
public synchronized MQDistributionList accessDistributionList
          (
           MQDistributionListItem[] litems,
           int openOptions,
          )
```

Arguments

litems Eléments à inclure dans la liste de diffusion.

openOptions
Options qui contrôlent l'ouverture de la liste de diffusion.

Pour plus de précisions sur les arguments, voir «accessDistributionList» à la page 155.

alternateUserId se voit attribuer la valeur *""*.

begin* (connexion directe uniquement)

```
public synchronized void begin()
```

Déclenche l'exception MQException.

Cette méthode est uniquement prise en charge par les classes MQSeries pour Java en mode de connexion directe et elle signale au gestionnaire de file d'attente qu'une nouvelle unité de travail démarre.

N'utilisez pas cette méthode pour les applications qui utilisent des transactions locales à phase unique.

isConnected

```
public boolean isConnected()
```

Renvoie la valeur de la variable isConnected.

MQSimpleConnectionManager

```
java.lang.Object      com.ibm.mq.MQConnectionManager
   |                          |
   └─ com.ibm.mq.MQSimpleConnectionManager
```

public class **MQSimpleConnectionManager**
implements **MQConnectionManager** (Voir page 161.)

Un MQSimpleConnectionManager fournit une fonctionnalité basique de regroupement de connexions. Vous pouvez utiliser un MQSimpleConnectionManager comme gestionnaire de connexion par défaut, ou comme argument pour un constructeur MQQueueManager. Lorsqu'un MQQueueManager est construit, la connexion la plus récemment utilisée du pool de connexion est réutilisée.

Les connexions sont supprimées (par une unité d'exécution distincte) lorsqu'elles ne sont pas utilisées pendant une période définie, ou lorsque le nombre spécifié de connexions inactives dans le pool est dépassé. Vous pouvez spécifier le délai d'expiration et le nombre maximal de connexions inutilisées.

Variables

MODE_AUTO
`public static final int MODE_AUTO`. Voir «setActive».

MODE_ACTIVE
`public static final int MODE_ACTIVE`. Voir «setActive».

MODE_INACTIVE
`public static final int MODE_INACTIVE`. Voir «setActive».

Constructeurs

MQSimpleConnectionManager
`public MQSimpleConnectionManager()`

Construit un MQSimpleConnectionManager.

Méthodes

setActive
`public void setActive(int mode)`

Définit le mode actif du pool de connexion.

Arguments

mode Mode actif requis pour le pool de connexion. Les valeurs possibles sont les suivantes :

MODE_AUTO
Le pool de connexion est actif lorsque le gestionnaire de connexion est le gestionnaire de connexion par défaut et qu'il y a au moins un jeton dans l'ensemble d'objets MQPoolToken géré par MQEnvironment. C'est le mode par défaut.

MODE_ACTIVE
Le pool de connexion est toujours actif. Lorsque

MQQueueManager.disconnect() est appelé, la connexion sous-jacente est ajouté au pool, et elle pourra être réutilisée la prochaine fois qu'un objet MQQueueManager sera construit. Les connexions seront supprimées par une unité d'exécution distincte si elles restent inutilisées au-delà du délai autorisé, ou si la taille du pool est supérieure à la valeur de HighThreshold.

MODE_INACTIVE
Le pool de connexion est toujours inactif. Lorsque ce mode est défini, le pool de connexion de MQSeries est supprimé. Lorsque MQQueueManager.disconnect() est appelé, la connexion sous-jacente à tous les objets MQQueueManager actifs est supprimée.

getActive

```
public int getActive()
```

Indique le mode du pool de connexion.

Valeur renvoyée

Mode actif en cours du pool de connexion, avec une des valeurs suivantes (voir «setActive» à la page 157) :

MODE_AUTO

MODE_ACTIVE

MODE_INACTIVE

setTimeout

```
public void setTimeout(long timeout)
```

Définit la valeur Timeout, qui indique le moment où les connexions restées inutilisées pendant cette période de temps sont supprimées par une unité d'exécution distincte.

Arguments

timeout
Valeur du délai d'expiration en millisecondes.

getTimeout

```
public long getTimeout()
```

Renvoie la valeur Timeout.

setHighThreshold

```
public void setHighThreshold(int threshold)
```

Définit la valeur HighThreshold. Si le nombre de connexions inutilisées du pool est supérieur à cette valeur, la plus ancienne connexion inutilisée du pool est supprimée.

Arguments

threshold
Nombre maximal de connexions inutilisées du pool.

getHighThreshold

```
public int getHighThreshold ()
```

Renvoie la valeur HighThreshold.

MQC

public interface **MQC**
extends **Object**

L'interface MQC définit toutes les constantes utilisées par l'interface de programmation MQ Java (à l'exception des constantes de code d'achèvement et de code d'erreur). Pour faire référence à une de ces constantes dans vos programmes, ajoutez à son nom le préfixe «MQC». Vous pouvez, par exemple, définir les options de fermeture d'une file d'attente de la façon suivante :

```
MQQueue queue;
 ...
queue.closeOptions = MQC.MQCO_DELETE; // Supprime la
                                      // file d'attente lors
                                      // de sa fermeture.
 ...
```

Vous trouverez une description complète de ces constantes dans le manuel *MQSeries Application Programming Reference* book.

Les constantes de code d'achèvement et de code d'erreur sont définies dans la classe MQException. Voir «MQException» à la page 99.

MQPoolServicesEventListener

public interface **MQPoolServicesEventListener**
extends **Object**

Remarque : Les applications n'utilisent généralement pas cette interface.

MQPoolServicesEventListener permet aux fournisseurs de mettre en oeuvre des ConnectionManagers par défaut. Lorsqu'un MQPoolServicesEventListener est enregistré avec un objet MQPoolServices, le programme d'écoute des événements reçoit un événement à chaque fois qu'un objet MQPoolToken est ajouté ou supprimé de l'ensemble d'objets MQPoolToken que gère MQEnvironment. Il reçoit également un événement à chaque fois que le ConnectionManager par défaut change.

Voir aussi «MQPoolServices» à la page 129 et «MQPoolServicesEvent» à la page 130.

Méthodes

tokenAdded

```
public void tokenAdded(MQPoolServicesEvent event)
```

Appelée lorsqu'un objet MQPoolToken est ajouté à l'ensemble.

tokenRemoved

```
public void tokenRemoved(MQPoolServicesEvent event)
```

Appelée lorsqu'un objet MQPoolToken est supprimé de l'ensemble.

defaultConnectionManagerChanged

```
public void defaultConnectionManagerChanged(MQPoolServicesEvent event)
```

Appelée lorsque le ConnectionManager par défaut est défini. L'ensemble d'objets MQPoolToken sera supprimé.

MQConnectionManager

Interface privée ne pouvant être mise en oeuvre par les applications. Le module classes MQSeries pour Java fournit une mise en oeuvre de cette interface (MQSimpleConnectionManager), que vous pouvez indiquer dans le constructeur MQQueueManager ou dans MQEnvironment.setDefaultConnectionManager.

Voir «MQSimpleConnectionManager» à la page 157.

Les applications ou les logiciels intermédiaires qui souhaitent fournir leur propre ConnectionManager doivent mettre en oeuvre javax.resource.spi.ConnectionManager. Cela suppose que Java 2 version 1.3 avec JAAS 1.0 soit installé.

MQReceiveExit

public interface **MQReceiveExit**
extends **Object**

L'interface d'exit de réception vous permet d'examiner et éventuellement de modifier les données reçues par les classes MQSeries pour Java en provenance d'un gestionnaire de files d'attente.

Remarque : Cette interface n'est pas applicable lorsque la connexion à MQSeries est directe (mode liens).

Pour créer votre propre exit de réception, définissez une classe mettant en œuvre cette interface. Créez une nouvelle instance de votre classe puis affectez-lui la variable MQEnvironment.receiveExit avant de construire votre objet MQQueueManager. Par exemple :

```
// Dans MyReceiveExit.java
class MyReceiveExit implements MQReceiveExit {
  // vous devez fournir la mise en oeuvre
  // de la méthode receiveExit.
  public byte[] receiveExit(
   MQChannelExit       channelExitParms,
   MQChannelDefinition channelDefinition,
   byte[]              agentBuffer)
  {
   // Placez ici le code de votre exit...
  }
}
// Dans votre programme principal...
MQEnvironment.receiveExit = new MyReceiveExit();
 ...    // Autres lignes d'initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Méthodes

receiveExit

```
public abstract byte[]
receiveExit(MQChannelExit channelExitParms,
                          MQChannelDefinition channelDefinition,
                          byte agentBuffer[])
```

Méthode d'exit de réception que doit fournir votre classe. Cette méthode sera appelée à chaque fois que les classes MQSeries pour Java recevront des données provenant du gestionnaire de files d'attente.

Arguments

channelExitParms
Contient des informations relatives au contexte dans lequel l'exit a été appelé. La variable membre exitResponse est un argument fourni en sortie que vous utilisez pour indiquer aux classes MQSeries pour Java l'action à effectuer ensuite. Voir «MQChannelExit» à la page 86 pour plus de précisions.

channelDefinition
Contient des détails relatifs au canal par lequel passent toutes les communications avec le gestionnaire de files d'attente.

agentBuffer
Si la zone channelExitParms.exitReason est égale à MQChannelExit.MQXR_XMIT, agentBuffer contient les données reçues du gestionnaire de files d'attente ; dans les autres cas, agentBuffer est nul.

Valeur renvoyée

Si la valeur du code de réponse de l'exit (dans channelExitParms) indique que les classes MQSeries pour Java peuvent maintenant traiter les données (MQXCC_OK), votre méthode d'exit de réception doit renvoyer les données à traiter. L'exit de réception le plus simple possible consiste donc en une seule ligne de code : «return agentBuffer;».

Voir aussi :
- «MQC» à la page 159
- «MQChannelDefinition» à la page 84

MQSecurityExit

public interface **MQSecurityExit**
extends **Object**

L'interface d'exit de sécurité vous permet de personnaliser les flux de sécurité occasionnés par toute tentative de connexion à un gestionnaire de files d'attente.

Remarque : Cette interface n'est pas applicable lorsque la connexion à MQSeries est directe (mode liens).

Pour créer votre propre exit de sécurité, définissez une classe mettant en œuvre cette interface. Créez une nouvelle instance de votre classe puis affectez-lui la variable MQEnvironment.securityExit avant de construire votre objet MQQueueManager. Par exemple :

```
// Dans MySecurityExit.java
class MySecurityExit implements MQSecurityExit {
  // vous devez fournir la mise en oeuvre
  // de la méthode securityExit.
  public byte[] securityExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // Placez ici le code de votre exit...
  }
}
// Dans votre programme principal...
MQEnvironment.securityExit = new MySecurityExit();
 ...    // Autres lignes d'initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Méthodes

securityExit

```
public abstract byte[]
securityExit(MQChannelExit channelExitParms,
                           MQChannelDefinition channelDefinition,
                           byte agentBuffer[])
```

Méthode d'exit de sécurité que doit fournir votre classe.

Arguments

channelExitParms
Contient des informations relatives au contexte dans lequel l'exit a été appelé. La variable membre exitResponse est un argument fourni en sortie que vous utilisez pour indiquer aux MQSeries Client pour Java l'action à effectuer ensuite. Voir la section «MQChannelExit» à la page 86 pour plus de précisions.

channelDefinition
Contient des détails relatifs au canal par lequel passent toutes les communications avec le gestionnaire de files d'attente.

agentBuffer
Si la zone channelExitParms.exitReason est égale à

MQChannelExit.MQXR_SEC_MSG, agentBuffer contient le message de sécurité reçu du gestionnaire de files d'attente ; dans les autres cas, agentBuffer est nul.

Valeur renvoyée

Si la valeur du code de réponse de l'exit (dans channelExitParms) indique qu'un message doit être transmis au gestionnaire de files d'attente, votre méthode d'exit de sécurité doit renvoyer les données à transmettre.

Voir aussi :.
- «MQC» à la page 159
- «MQChannelDefinition» à la page 84

MQSendExit

public interface **MQSendExit**
extends **Object**

L'interface d'exit d'envoi vous permet d'examiner et éventuellement de modifier les données envoyées par MQSeries Client pour Java à un gestionnaire de files d'attente.

Remarque : Cette interface n'est pas applicable lorsque la connexion à MQSeries est directe (mode liens).

Pour créer votre propre exit d'envoi, définissez une classe mettant en œuvre cette interface. Créez une nouvelle instance de votre classe puis affectez-lui la variable MQEnvironment.sendExit avant de construire votre objet MQQueueManager. Par exemple :

```
// Dans MySendExit.java
class MySendExit implements MQSendExit {
  // Vous devez fournir la mise en œuvre de la méthode sendExit
  public byte[] sendExit(
    MQChannelExit       channelExitParms,
    MQChannelDefinition channelDefinition,
    byte[]              agentBuffer)
  {
    // Placez ici le code de votre exit...
  }
}
// Dans votre programme principal...
MQEnvironment.sendExit = new MySendExit();
 ...    // Autres lignes d'initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Méthodes

sendExit

```
public abstract byte[]
sendExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefinition,
                            byte agentBuffer[])
```

Méthode d'exit d'envoi que doit fournir votre classe. Cette méthode sera appelée à chaque fois que les classes MQSeries pour Java enverront des données au gestionnaire de files d'attente.

Arguments

channelExitParms
Contient des informations relatives au contexte dans lequel l'exit a été appelé. La variable membre exitResponse est un argument fourni en sortie que vous utilisez pour indiquer aux classes MQSeries pour Java l'action à effectuer ensuite. Pour plus de précisions, voir «MQChannelExit» à la page 86.

channelDefinition
Contient des détails relatifs au canal par lequel passent toutes les communications avec le gestionnaire de files d'attente.

agentBuffer
Si la zone channelExitParms.exitReason est égale à

MQChannelExit.MQXR_XMIT, agentBuffer contient les données à envoyer au gestionnaire de files d'attente ; dans les autres cas, agentBuffer est nul.

Valeur renvoyée

Si la valeur du code de réponse de l'exit (dans channelExitParms) indique qu'un message doit être transmis au gestionnaire de files d'attente (MQXCC_OK), votre méthode d'exit d'envoi doit renvoyer les données à transmettre. L'exit d'envoi le plus simple possible consiste donc en une seule ligne de code : «return agentBuffer;».

Voir aussi :
- «MQC» à la page 159
- «MQChannelDefinition» à la page 84

ManagedConnection

public interface **javax.resource.spi.ManagedConnection**

Remarque : Normalement, les applications n'utilisent pas cette classe. Elle est conçue pour les mises en oeuvre de ConnectionManager.

Les classes MQSeries pour Java fournissent une mise en oeuvre de ManagedConnection qui est renvoyée par ManagedConnectionFactory.createManagedConnection. Cet objet représente une connexion à un gestionnaire de files d'attente MQSeries.

Méthodes

getConnection

```
public Object getConnection(javax.security.auth.Subject subject,
                            ConnectionRequestInfo cxRequestInfo)
```

Déclenche l'exception ResourceException.

Crée un nouveau descripteur de connexion pour la connexion physique représentée par l'objet ManagedConnection. Pour les classes MQSeries pour Java, cette méthode renvoie un objet MQQueueManager. Le ConnectionManager renvoie normalement cet objet depuis allocateConnection.

L'argument objet est ignoré. Si l'argument cxRequestInfo est inapproprié, une exception ResourceException est déclenchée. Il est possible d'utiliser plusieurs descripteurs de connexion simultanément pour chaque objet ManagedConnection.

destroy

```
public void destroy()
```

Déclenche l'exception ResourceException.

Supprime la connexion physique avec le gestionnaire de files d'attente MQSeries. Toute transaction locale en attente est validée. Pour plus de détails, voir «getLocalTransaction» à la page 169.

cleanup

```
public void cleanup()
```

Déclenche l'exception ResourceException.

Ferme tous les descripteurs de connexion ouverts et réinitialise la connexion physique à son état initial afin qu'elle soit prête à être remise dans le pool. Toute transaction locale en attente est annulée. Pour plus de détails, voir «getLocalTransaction» à la page 169.

associateConnection

```
public void associateConnection(Object connection)
```

Déclenche l'exception ResourceException.

Les classes MQSeries pour Java ne prennent actuellement pas en charge cette méthode. Une exception javax.resource.NotSupportedException est déclenchée.

addConnectionEventListener

```
public void addConnectionEventListener(ConnectionEventListener listener)
```

Ajoute un ConnectionEventListener à l'instance de ManagedConnection.

Si une erreur grave se produit au niveau de l'instance de ManagedConnection, ou lorsque MQQueueManager.disconnect() est appelé au niveau d'un descripteur de connexion qui est associé à cette instance de ManagedConnection, le programme d'écoute reçoit un avertissement. Le programme d'écoute n'est pas averti des événements liés aux transactions locales (voir «getLocalTransaction»).

removeConnectionEventListener

```
public void removeConnectionEventListener(ConnectionEventListener listener)
```

Supprime un ConnectionEventListener enregistré.

getXAResource

```
public javax.transaction.xa.XAResource getXAResource
```

Déclenche l'exception ResourceException.

Les classes MQSeries pour Java ne prennent actuellement pas en charge cette méthode. Une exception javax.resource.NotSupportedException est déclenchée.

getLocalTransaction

```
public LocalTransaction getLocalTransaction()
```

Les classes MQSeries pour Java ne prennent actuellement pas en charge cette méthode. Une exception javax.resource.NotSupportedException est déclenchée.

Actuellement, un ConnectionManager ne peut pas gérer la transaction locale MQSeries, et les objets ConnectionEventListener enregistrés ne sont pas informés des événements liés à la transaction locale. Lorsqu'une opération cleanup() est effectuée, toute unité de travail en cours est annulée. Lorsqu'une opération destroy() est effectuée, toute unité de travail en cours est validée.

Le comportement actuel de l'API est le suivant : une unité de travail en cours est validée au niveau de MQQueueManager.disconnect(). Ce comportement n'est préservé que si c'est un MQConnectionManager (et non un ConnectionManager) qui gère la connexion.

getMetaData

```
public ManagedConnectionMetaData getMetaData()
```

Déclenche l'exception ResourceException.

Extrait les informations liées aux métadonnées pour le gestionnaire de files d'attente sous-jacent. Voir «ManagedConnectionMetaData» à la page 173.

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

Déclenche l'exception ResourceException.

Définit l'éditeur de fichier journal pour cet objet ManagedConnection. Lorsqu'un objet ManagedConnection est créé, il hérite de l'éditeur de fichier journal de ManagedConnectionFactory.

Les classes MQSeries pour Java n'utilisent actuellement pas l'éditeur de fichier journal. Pour plus d'informations sur la consignation, voir «MQException.log» à la page 99.

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

Déclenche l'exception ResourceException.

Renvoie l'éditeur de fichier journal pour cet objet ManagedConnection.

Les classes MQSeries pour Java n'utilisent actuellement pas l'éditeur de fichier journal. Pour plus d'informations sur la consignation, voir «MQException.log» à la page 99.

ManagedConnectionFactory

public interface **javax.resource.spi.ManagedConnectionFactory**

Remarque : Normalement, les applications n'utilisent pas cette classe.

Les classes MQSeries pour Java fournissent une mise en oeuvre de cette interface pour ConnectionManagers. Une classe ManagedConnectionFactory est utilisée pour construire des objets ManagedConnections et pour sélectionner un objet ManagedConnection approprié dans un ensemble de candidats. Pour plus d'informations sur cette interface, consultez la spécification de l'architecture de connecteur J2EE (visitez le site Web de Sun à l'adresse `http://java.sun.com`).

Méthodes

createConnectionFactory

```
public Object createConnectionFactory()
```

Déclenche l'exception ResourceException.

Les classes MQSeries pour Java ne prennent actuellement pas en charge les méthodes createConnectionFactory. Cette méthode déclenche une exception javax.resource.NotSupportedException.

createConnectionFactory

```
public Object createConnectionFactory(ConnectionManager cxManager)
```

Déclenche l'exception ResourceException.

Les classes MQSeries pour Java ne prennent actuellement pas en charge les méthodes createConnectionFactory. Cette méthode déclenche une exception javax.resource.NotSupportedException.

createManagedConnection

```
public ManagedConnection createManagedConnection
                                (javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

Déclenche l'exception ResourceException.

Crée une nouvelle connexion physique avec un gestionnaire de files d'attente MQSeries et renvoie un objet ManagedConnection qui représente cette connexion. MQSeries ignore l'argument objet.

matchManagedConnection

```
public ManagedConnection matchManagedConnection
                                (java.util.Set connectionSet,
                                 javax.security.auth.Subject subject,
                                 ConnectionRequestInfo cxRequestInfo)
```

Déclenche l'exception ResourceException.

Recherche dans l'ensemble d'objets ManagedConnections candidats fourni un objet ManagedConnection approprié. Renvoie la valeur NULL ou un objet ManagedConnection approprié de l'ensemble, qui répond aux critères de connexion.

setLogWriter

```
public void setLogWriter(java.io.PrintWriter out)
```

Déclenche l'exception ResourceException.

Définit l'éditeur de fichier journal pour cet objet ManagedConnectionFactory. Lorsqu'un objet ManagedConnection est créé, il hérite de l'éditeur de fichier journal de son objet ManagedConnectionFactory.

Les classes MQSeries pour Java n'utilisent actuellement pas l'éditeur de fichier journal. Pour plus d'informations sur la consignation, voir «MQException.log» à la page 99.

getLogWriter

```
public java.io.PrintWriter getLogWriter()
```

Déclenche l'exception ResourceException.

Renvoie l'éditeur de fichier journal pour cet objet ManagedConnectionFactory.

Les classes MQSeries pour Java n'utilisent actuellement pas l'éditeur de fichier journal. Pour plus d'informations sur la consignation, voir «MQException.log» à la page 99.

hashCode

```
public int hashCode()
```

Renvoie le code hashCode pour cet objet ManagedConnectionFactory.

equals

```
public boolean equals(Object other)
```

Vérifie si cet objetManagedConnectionFactory est égal à un autre objet ManagedConnectionFactory. Renvoie la valeur vrai si les deux objets ManagedConnectionFactories décrivent le même gestionnaire de files d'attente cible.

ManagedConnectionMetaData

public interface **javax.resource.spi.ManagedConnectionMetaData**

Remarque : Normalement, les applications n'utilisent pas cette classe. Elle est conçue pour les mises en oeuvre de ConnectionManager.

Un ConnectionManager peut utiliser cette classe pour extraire des métadonnées ayant trait à une connexion physique sous-jacente à un gestionnaire de files d'attente. Une mise en oeuvre de cette classe est renvoyée par ManagedConnection.getMetaData().

Méthodes

getEISProductName

```
public String getEISProductName()
```

Déclenche l'exception ResourceException.

Renvoie "IBM MQSeries".

getProductVersion

```
public String getProductVersion()
```

Déclenche l'exception ResourceException.

Renvoie une chaîne qui décrit le niveau de commande du gestionnaire de files d'attente MQSeries auquel l'objet ManagedConnection est connecté.

getMaxConnections

```
public int getMaxConnections()
```

Déclenche l'exception ResourceException.

Renvoie la valeur 0.

getUserName

```
public String getUserName()
```

Déclenche l'exception ResourceException.

Si l'objet ManagedConnection représente une connexion client à un gestionnaire de files d'attente, renvoie l'ID utilisateur utilisé pour la connexion. Sinon, renvoie une chaîne vide.

ManagedConnectionMetaData

Partie 3. Programmation avec MQ JMS

Chapitre 10. Ecriture de programmes MQ JMS

Le présent chapitre fournit des informations qui vous aideront à écrire des applications MQ JMS. Il présente brièvement le modèle JMS et fournit des informations détaillées sur la programmation de certaines tâches courantes que les programmes d'application risquent de devoir exécuter.

Modèle JMS

JMS définit une vue générale de service de transmission de messages. Il est important de bien comprendre cette vue et la manière dont elle est mappée dans le transfert MQSeries sous-jacent.

Le modèle générique JMS est fondé sur les interfaces suivantes, définies dans le module `javax.jms` de Sun :

Connexion
Permet d'accéder au transfert sous-jacent et de créer des **Sessions**.

Session
Fournit un contexte pour l'envoi et la réception de messages, y compris les méthodes utilisées pour créer des objets **MessageProducers** et **MessageConsumers**.

MessageProducer
Permet d'envoyer des messages.

MessageConsumer
Permet de recevoir les messages.

Notez qu'une **Connexion** prend en charge les unités d'exécution multiples, contrairement aux **Sessions**, aux **MessageProducers** et aux **MessageConsumers**. La stratégie recommandée est d'utiliser une Session par unité d'exécution d'application.

Dans MQSeries :

Connexion
Définit la portée des files d'attente temporaires. Indique également un emplacement pour les arguments de contrôle des connexion à MQSeries. Parmi les exemples d'arguments, on compte le nom du gestionnaire de files d'attente et celui de l'hôte éloigné si vous utilisez la connectivité client Java de MQSeries.

Session
Contient un objet `HCONN` et définit donc une portée transactionnelle.

MessageProducer et MessageConsumer
Contiennent un objet `HOBJ` qui définit une file d'attente à lire ou dans laquelle écrire.

Notez que les règles MQSeries normales sont applicables :

- Chaque HCONN ne prend en charge qu'une opération à la fois. Aussi, les MessageProducers ou MessageConsumers associés à une Session ne peuvent pas être appelés en même temps. Ce comportement suit la logique de la restriction JMS qui ne prend en charge qu'une unité d'exécution par Session.

- Les insertions (PUT) peuvent utiliser des files d'attente éloignées, tandis que les extractions (GET) ne sont applicables qu'aux files d'attente du gestionnaires de files d'attente local.

Les interfaces génériques JMS sont classées en versions adaptées au comportement 'point à point' et 'publication/souscription'.

Les versions point à point sont les suivantes :
- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

Une des notions de base de JMS est qu'il est possible, et fortement recommandé, d'écrire des programmes d'application qui utilisent uniquement des références aux interfaces de `javax.jms`. Les données propres aux fournisseurs sont encapsulées dans la mise en oeuvre des éléments suivants :
- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

Ces objets sont des 'objets gérés par l'administrateur', c'est-à-dire qu'ils peuvent être construits à l'aide d'outils d'administration propres à certains fabricants et qu'ils peuvent être stockés dans un espace annuaire JNDI. Une application JMS peut extraire ces objets de l'espace annuaire et les utiliser sans connaître l'identité du fournisseur.

Construction d'une connexion

Les connexions ne sont pas créées directement. Elles sont construites à l'aide d'une fabrique de connexion. Les objets fabrique peuvent être stockés dans un espace annuaire JNDI, isolant ainsi l'application JMS des données fournisseur. Pour plus de détails sur la création et le stockage d'objets fabrique, reportez-vous au «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33.

Si vous ne disposez d'aucun espace annuaire JNDI, reportez-vous à la section «Création de fabriques en phase d'exécution» à la page 179.

Extraction de la fabrique depuis JNDI

Pour extraire un objet d'un espace annuaire JNDI, vous devez définir un contexte initial, comme indiqué dans l'extrait suivant d'un fichier exemple de `IVTRun` :

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

où :

icf définit une classe de fabrique pour le contexte initial

url définit une URL spécifique du contexte

Pour plus de détails sur l'utilisation de JNDI, reportez-vous à la documentation JNDI.

Remarque : Certaines combinaisons de modules JNDI et de fournisseurs de services LDAP risquent de générer une erreur LDAP 84. Pour résoudre cet incident, insérez la ligne suivante avant l'appel de `InitialDirContext`.

```
environment.put(Context.REFERRAL, "throw");
```

Une fois le contexte initial obtenu, les objets sont extraits de l'espace annuaire à l'aide de la méthode `lookup()`. Le code suivant extrait un objet QueueConnectionFactory nommé `ivtQCF` d'un espace annuaire de type LDAP.

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

Utilisation de la fabrique pour créer une connexion

La méthode `createQueueConnection()` de l'objet fabrique permet de créer une connexion, comme l'indique le code suivant :

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

Création de fabriques en phase d'exécution

Si aucun espace annuaire JNDI n'est disponible, vous pouvez créer des objets fabrique en phase d'exécution. Toutefois, cette méthode réduit les caractéristiques de portabilité de l'application JMS, car elle requiert des références à des classes MQSeries spécifiques.

Le code suivant permet de créer un objet QueueConnectionFactory avec tous les arguments par défaut :

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(Le préfixe `com.ibm.mq.jms.` est inutile si vous importez le module `com.ibm.mq.jms`.)

Une connexion créée à partir de la fabrique ci-dessus utilise les liaisons Java pour se connecter au gestionnaire de files d'attente par défaut du poste local. Les méthodes set figurant dans le tableau 14 peuvent permettre de personnaliser la fabrique à l'aide de données spécifiques MQSeries.

Lancement de la connexion

Avec JMS, les connexions doivent être créées à l'état 'arrêté'. Tant que la connexion n'est pas lancée, les objets MessageConsumers qui lui sont associés ne peuvent recevoir aucun message. Pour lancer la connexion, entrez la commande suivante :

```
connection.start();
```

Tableau 14. Méthodes Set de MQQueueConnectionFactory

Méthode	Description
setCCSID(int)	Permet de définir la propriété `MQEnvironment.CCSID`
setChannel(String)	Nom de canal d'une connexion client
setHostName(String)	Nom d'hôte d'une connexion client
setPort(int)	Numéro de port d'une connexion client
setQueueManager(String)	Nom du gestionnaire de files d'attente

Tableau 14. Méthodes Set de MQQueueConnectionFactory (suite)

Méthode	Description
setTemporaryModel(String)	Nom d'une file d'attente modèle permettant de générer une destination temporaire comme résultat d'un appel de `QueueSession.createTemporaryQueue()`. Il est recommandé de définir le nom d'une file d'attente dynamique temporaire, plutôt que celui d'une file d'attente dynamique permanente.
setTransportType(int)	Indique comment se connecter à MQSeries. Les options disponibles sont les suivantes : • JMSC.MQJMS_TP_BINDINGS_MQ (par défaut) • JMSC.MQJMS_TP_CLIENT_MQ_TCPIP. `JMSC` correspond au module `com.ibm.mq.jms`
setReceiveExit(String) setSecurityExit(String) setSendExit(String) setReceiveExitInit(String) setSecutityExitInit(String) setSendExitInit(String)	Ces méthodes permettent l'utilisation des exits d'envoi, de réception et de sécurité fournis par les classes MQSeries pour Java sous-jacentes. Les méthodes `set*Exit` prennent le nom d'une classe permettant de mettre en oeuvre les méthodes d'exit appropriées. (Pour plus de détails, reportez-vous à la documentation MQSeries 5.1.) En outre, la classe doit mettre en oeuvre un constructeur doté d'un seul argument `String`. Ce dernier fournit les données d'initialisation requises par l'exit et est associé à une valeur fournie dans la méthode `set*ExitInit` correspondante.

Choix d'un transfert client ou de liaisons

MQ JMS peut communiquer avec MQSeries à l'aide de transferts client ou de liaisons. Si vous utilisez les liaisons Java, l'application JMS et le gestionnaire de files d'attente MQSeries doivent se trouver sur le même poste. Si vous utilisez le client, le gestionnaire de files d'attente peut se trouver sur un autre poste que l'application.

Le contenu de l'objet fabrique de la connexion permet de déterminer le type de transfert à utiliser. Le «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33 décrit comment définir un objet fabrique à utiliser avec un transfert client ou de liaisons.

L'extrait de code suivant illustre comment définir le transfert au sein d'une application :

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

Obtention d'une session

Une fois la connexion établie, utilisez la méthode createQueueSession de QueueConnection pour obtenir une session.

La méthode prend deux arguments :

1. Argument booléen qui détermine si la session est 'transactionnelle' ou non.
2. Argument qui prend en charge le mode 'accusé de réception'.

Le cas le plus simple est celui d'une session 'non transactionnelle' en mode AUTO_ACKNOWLEDGE, comme la suivante :

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

Remarque : Une connexion prend en charge les unités d'exécution multiples, contrairement aux session (et aux objets dont la création dépend des sessions). Pour les applications à plusieurs unités d'exécution, il est recommandé d'utiliser une session par unité.

Envoi de message

Les messages sont envoyés à l'aide de l'objet MessageProducer. Pour les connexions point à point, il s'agit d'un objet QueueSender créé à l'aide de la méthode createSender d'une QueueSession. Un objet QueueSender est généralement créé pour une file d'attente spécifique, de sorte que tous les messages envoyés à l'aide de cet émetteur atteignent la même destination. Cette dernière est indiquée à l'aide d'un objet Queue qui peut être créé en phase d'exécution ou construit et stocké dans un espace annuaire JNDI.

Les objets Queue sont extraits de JNDI de la manière suivante :

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

MQ JMS offre une mise en oeuvre de Queue dans com.ibm.mq.jms.MQQueue. Cela permet de définir les propriétés de l'objet qui permettent de contrôler le comportement de MQSeries, mais il est souvent possible d'utiliser les valeurs par défaut. JMS définit un moyen standard pour indiquer la destination en minimisant le code MQSeries dans l'application. Ce mécanisme utilise la méthode QueueSession.createQueue qui prend un argument string pour décrire la destination. Cet argument consiste en une chaîne qui apparaît dans un format propre au fournisseur, mais il s'agit d'une approche plus souple que celle qui consiste à référencer directement les classes du fournisseur.

MQ JMS accepte deux formats pour l'argument string de createQueue().

- Le premier est le nom de la file d'attente MQSeries, comme l'illustre l'extrait suivant du programme IVTRun qui se trouve dans le répertoire samples :

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- Le second format, plus performant, est fondé sur les URI ('identificateurs de ressources uniformes'). Ce format permet d'indiquer des files d'attente éloignées

(files d'attente d'un gestionnaire de files d'attente autre que celui auquel vous êtes connecté). Il permet également de définir les autre propriétés contenues dans un objet com.ibm.mq.jms.MQQueue.

L'URI d'une file d'attente commence par la séquence queue://, suivie du nom du gestionnaire de file d'attente sur lequel la file d'attente réside. Cette séquence est suivie d'une barre oblique '/', du nom de la file d'attente et, en option, de la liste de combinaisons nom-valeur qui définit les autres propriétés de la file d'attente. Par exemple, l'URI associée à l'exemple précédent est la suivante :

```
ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
```

Notez que le nom du gestionnaire de file d'attente n'apparaît pas. Dans ce cas, c'est le gestionnaire auquel l'objet QueueConnection propriétaire est connecté au moment où l'objet Queue est utilisé qui est pris en compte.

L'exemple suivant se connecte à la file d'attente 'Q1' du gestionnaire de files d'attente 'HOST1.QM1' et permet d'envoyer tous les messages non persistants et de priorité de niveau 5 :

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

Le tableau 15 répertorie les noms à utiliser dans la partie nom-valeur de l'URI. Ce format présente l'inconvénient de ne pas prendre en charge les noms symboliques pour les valeurs. Aussi, le tableau peut parfois indiquer des valeurs 'spéciales'. Notez que ces dernières sont susceptibles d'être modifiées. (Pour voir une autre méthode de définition des propriétés, reportez-vous à la section «Définition de propriétés à l'aide de la méthode 'set'» à la page 183.)

Tableau 15. Noms de propriété pour les URI de file d'attente

Propriété	Description	Valeurs
expiry	Durée de vie du message, en millisecondes	0 = illimité, entiers positifs = délai (ms)
priority	Niveau de priorité du message	0 à 9, -1 = QDEF, -2 = APP
persistence	Indique un message à sauvegarder sur disque	1 = non persistant, 2 = persistant, -1 = QDEF, -2 = APP
CCSID	Jeu de caractères de la destination	entiers - valeurs correctes répertoriées dans la documentation de base MQSeries
targetClient	Indique si l'application destinataire est ou non compatible JMS	0 = JMS, 1 = MQ
encoding	Indique comment représenter les zones numériques	Entier, comme décrit dans la documentation de base MQSeries

QDEF - valeur spéciale qui indique que la propriété doit être déterminée par la configuration de la file d'attente de MQSeries.

APP - valeur spéciale qui indique que l'application JMS peut contrôler cette propriété.

Une fois l'objet Queue obtenu (à l'aide de createQueue comme ci-dessus, ou à partir de JNDI), il doit être transmis à la méthode createSender pour créer un objet QueueSender :

```
QueueSender queueSender = session.createSender(ioQueue);
```

L'objet queueSender résultant de cette opération est utilisé pour envoyer des messages à l'aide de la méthode send :

```
queueSender.send(outMessage);
```

Définition de propriétés à l'aide de la méthode 'set'

Vous pouvez définir des propriétés Queue en créant d'abord une instance de`com.ibm.mq.jms.MQQueue` à l'aide du constructeur par défaut. Vous pouvez ensuite indiquer les valeurs obligatoires à l'aide des méthodes `set` publiques. Cette méthode signifie que vous pouvez utiliser des noms symboliques pour les valeurs de propriété. Toutefois, ces valeurs étant propres aux fournisseurs, et incluses dans le code, les applications perdent leurs caractéristiques de portabilité.

L'extrait de code suivant montre la définition des propriétés de file d'attente à l'aide de la méthode `set`.

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

Le tableau 16 répertorie les valeurs symboliques de propriétés fournies avec MQ JMS pour être utilisées avec les méthodes `set`.

Tableau 16. Valeurs symboliques pour les propriétés de file d'attente

Propriété	**Mot-clé de l'outil d'admin**	**Valeurs**
expiry	UNLIM APP	JMSC.MQJMS_EXP_UNLIMITED JMSC.MQJMS_EXP_APP
priority	APP QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
persistence	APP QDEF PERS NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
targetClient	JMS MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
encoding	Integer(N) Integer(R) Decimal(N) Decimal(R) Float(N) Float(R) Native	JMSC.MQJMS_ENCODING_INTEGER_NORMAL JMSC.MQJMS_ENCODING_INTEGER_REVERSED JMSC.MQJMS_ENCODING_DECIMAL_NORMAL JMSC.MQJMS_ENCODING_DECIMAL_REVERSED JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED JMSC.MQJMS_ENCODING_NATIVE

Pour plus de détails sur la propriété encoding, reportez-vous à la section «Propriété ENCODING» à la page 44.

Types de messages

JMS propose différents types de messages, chacun correspondant à un contenu particulier. Pour éviter d'utiliser les noms de classes propres aux fournisseurs dans les messages, des méthodes sont fournies dans l'objet Session pour créer des messages.

Dans le programme exemple, un message de type texte est créé comme suit :

Envoi de message

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

Les types de messages disponibles sont les suivants :
- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Pour plus de détails sur ces types de messages, reportez-vous au «Chapitre 14. Interfaces et classes JMS» à la page 237.

Réception d'un message

Les messages sont reçus grâce à l'objet QueueReceiver créé à partir d'une Session à l'aide de la méthode `createReceiver()`. Cette méthode prend un argument Queue qui définit l'origine du message. Pour plus de détails sur la création d'un objet Queue, reportez-vous à la section «Envoi de message» à la page 181.

Le programme exemple crée un récepteur et relit le message test à l'aide du code suivant :

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

L'argument de l'appel de réception est un délai exprimé en millisecondes. Cet argument définit la période d'attente de la méthode si aucun message n'est immédiatement disponible. Cet argument n'est pas obligatoire mais si vous ne l'indiquez pas, l'appel est bloqué. Si vous ne souhaitez pas définir de délai, utilisez la méthode `receiveNoWait()`.

Les méthodes de réception renvoient un message du type approprié. Si, par exemple, un message `TextMessage` est mis en file d'attente, lorsqu'il est reçu, l'objet renvoyé est une instance de `TextMessage`.

Pour extraire le contenu du corps du message, vous devez convertir la classe générique Message (type de retour déclaré des méthodes de réception) en une sous-classe plus précise, telle que `TextMessage`. Si le type de message reçu est inconnu, vous pouvez utiliser l'opérateur 'instanceof'pour déterminer son type. Il est recommandé de prendre l'habitude de tester la classe de message avant de la convertir, afin de permettre un traitement optimal des erreurs inattendues.

Le code suivant illustre l'utilisation de 'instanceof', et l'extraction du contenu d'un message TextMessage :

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Impression du message d'erreur si le message n'est pas du type TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Sélecteurs de messages

JMS fournit un mécanisme permettant de sélectionner un sous-ensemble de messages dans une file d'attente de sorte que ce sous-ensemble soit renvoyé par un appel de réception. Lors de la création d'un objet QueueReceiver, vous pouvez indiquer une chaîne contenant une expression SQL pour identifier les messages à extraire. Le sélecteur peut se reporter à des zones d'en-tête du message JMS ainsi qu'à des zones des propriétés du message (qui sont en fait des en-têtes définis par l'application). Pour plus de détails sur les noms des zones d'en-tête et la syntaxe du sélecteur SQL, reportez-vous au «Chapitre 12. Messages JMS» à la page 199.

L'exemple suivant illustre comment sélectionner une propriété utilisateur nommée myProp:

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

Remarque : Les caractéristiques JMS ne permettent pas au sélecteur associé au récepteur de subir une modification. Lorsqu'un récepteur est créé, le sélecteur est définitif pour ce récepteur. Ce qui signifie que si vous souhaitez utiliser d'autres sélecteurs, vous devez créer des récepteurs.

Livraison asynchrone

Pour émettre des appels de QueueReceiver.receive(), vous pouvez également enregistrer une méthode appelée automatiquement lorsqu'un message approprié est disponible. L'extrait suivant illustre ce mécanisme :

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // Méthode qui sera appelée par JMS lorsqu'un
  // message sera disponible.
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // traitement propre à l'application
    .
    .
    .
  }
}

 .
 .
 .
  // Dans le programme principal (éventuellement d'une autre classe)
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // le programme principal peut poursuivre avec un autre
  // comportement propre à l'application.
```

Remarque : L'utilisation de la livraison asynchrone dans un objet QueueReceiver marque la totalité de la session comme asynchrone. L'émission d'un appel explicite vers des méthodes receive de l'objet QueueReceiver associé à une Session qui utilise la livraison asynchrone est considérée comme une erreur.

Fermeture

La récupération d'espace mémoire à elle seule ne peut pas restaurer toutes les ressources MQSeries à temps. Surtout si l'application doit créer de nombreux objets JMS de courte durée au niveau de la Session ou en dessous. Il est donc important d'appeler les méthodes `close()` des différentes classes (QueueConnection, QueueSession, QueueSender et QueueReceiver) lorsque les ressources sont devenues inutiles.

JVM (Java Virtual Machine) se bloque à la fermeture

Si une application MQ JMS se termine sans appeler `Connection.close()`, certaines machines JVM se bloquent. Si cet incident se produit, modifiez l'application pour inclure un appel à `Connection.close()`, ou fermez la machine JVM à l'aide des touches `Ctrl-C`.

Traitement des erreurs

Toutes les erreurs d'exécution d'une application JMS sont consignées en tant qu'exceptions. La plupart des méthodes de JMS déclenchent des exceptions JMSExceptions pour signaler une erreur. Cette pratique de programmation est excellente pour détecter les exceptions et les afficher dans un format approprié.

Contrairement aux exceptions Java classiques, une exception JMSException peut en contenir une autre. Pour JMS, il peut s'agir d'un bon moyen de transmettre des détails importants à partir d'un transfert sous-jacent. Pour MQ JMS, lorsque MQSeries génère une MQException, cette exception est généralement intégrée dans une JMSException.

La mise en oeuvre d'une exception JMSException ne fait pas apparaître l'exception intégrée dans le résultat de sa méthode `toString()`. Aussi, vous devez vérifier la présence d'exceptions intégrées et les imprimer, comme l'illustre l'extrait suivant :

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

Programme d'écoute d'exception

Pour une livraison asynchrone de messages, le code d'application ne peut pas détecter les exceptions générées par des échecs de réception des messages. Cela est dû au fait que le code de l'application n'émet pas d'appel explicite vers des méthodes `receive()`. Pour gérer cette situation, vous pouvez enregistrer un`ExceptionListener`, qui est une instance d'une classe mettant en oeuvre la méthode `onException()`. Lorsqu'une erreur grave se produit, cette méthode est appelée et présente JMSException comme seul argument. Pour plus de détails, reportez-vous à la documentation Sun sur JMS.

Chapitre 11. Programmation d'applications de publication/souscription

La présente section décrit le modèle de programmation à partir duquel vous pouvez écrire des applications de publication/souscription (Publish/Subscribe) qui utilisent MQSeries Classes pour Java - Message Service.

Ecriture d'une application de publication/souscription simple

La présente section examine une application MQ JMS simple.

Importation des modules requis

Une application MQSeries Classes pour Java - Message Service commence par des instructions d'importation comprenant au moins les suivantes :

```
import javax.jms.*;                // Interfaces JMS
import javax.nommage.*;             // Pour la recherche JNDI
import javax.annuaire.nommage.*;   //    d'objets gérés
```

Extraction ou création d'objets JMS

L'étape suivante consiste à extraire ou à créer des objets JMS :

1. Extraction d'un objet TopicConnectionFactory
2. Création d'un objet TopicConnection
3. Création d'un objet TopicSession
4. Extraction d'un objet Topic à partir de JNDI
5. Création des objets TopicPublisher et TopicSubscriber

La plupart des processus ci-dessus sont similaires à ceux mis en oeuvre dans un domaine point à point, comme illustré ci-après :

Extraction d'un objet TopicConnectionFactory

La méthode privilégiée consiste à employer l'interface de recherche JNDI pour conserver la portabilité du code de l'application. Le code suivant permet d'initialiser un contexte JNDI :

```
String CTX_FACTORY = "com.sun.jndi.ldap.LdapCtxFactory";
String INIT_URL    = "ldap://server.company.com/o=company_us,c=us";

Java.util.Hashtable env = new java.util.Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, CTX_FACTORY );
env.put( Context.PROVIDER_URL,            INIT_URL );
env.put( Context.REFERRAL,                "throw" );

Context ctx = null;
try {
   ctx = new InitialDirContext( env );
} catch( NamingException nx ) {
   // Ajout de code pour le traitement d'une non-connexion au contexte JNDI
}
```

Remarque : Les variables `CTX_FACTORY` et `INIT_URL` doivent être personnalisées en fonction de l'installation et du fournisseur de services JNDI.

Les propriétés requises par l'initialisation JNDI consistent en une table de hachage transmise par le constructeur InitialDirContext. Si la connexion échoue, une exception est déclenchée pour indiquer que les objets gérés requis ultérieurement dans l'application sont indisponibles.

Procédez-vous à présent à l'extraction d'un objet TopicConnectionFactory à l'aide d'une clé de recherche définie par l'administrateur :

```
TopicConnectionFactory factory;
factory = (TopicConnectionFactory)lookup("cn=sample.tcf");
```

Si aucun espace annuaire JNDI n'est disponible, vous pouvez créer l'objet TopicConnectionFactory en phase d'exécution. Pour créer un objet com.ibm.mq.jms.MQTopicConnectionFactory, utilisez la méthode décrite pour l'objet QueueConnectionFactory (reportez-vous à la section «Création de fabriques en phase d'exécution» à la page 179).

Création d'un objet TopicConnection

Cette connexion est créée à partir de l'objet TopicConnectionFactory. Les connexions sont toujours initialisées à l'état d'arrêt. Leur démarrage s'effectue à l'aide du code suivant :

```
TopicConnection conn;
conn = factory.createTopicConnection();
conn.start();
```

Création d'un objet TopicSession

Cette session est créée à l'aide de l'objet TopicConnection. Cette méthode accepte deux paramètres d'entrée : l'un indique si la session est transactionnelle et l'autre précise le mode d'accusé de réception attendu.

```
TopicSession session = conn.createTopicSession( false,
                                                Session.AUTO_ACKNOWLEDGE );
```

Extraction d'un objet Topic

Cet objet peut être extrait à partir de JNDI en vue de son utilisation avec les objets TopicPublisher et TopicSubscriber qui seront créés ultérieurement. Le code suivant permet d'extraire un objet Topic :

```
Topic topic = null;
 try {
    topic = (Topic)ctx.lookup( "cn=sample.topic" );
 } catch( NamingException nx ) {
    // Ajout de code pour le traitement de la non-extraction de l'objet Topic de JNDI
 }
```

Si aucun espace annuaire JNDI n'est disponible, vous pouvez créer l'objet Topic en phase d'exécution (reportez-vous à la section «Création de rubriques en phase d'exécution» à la page 191).

Création de clients et de producteurs de publications

Selon le type de l'application client JMS que vous écrivez, vous devez créer un souscripteur (subscriber) et/ou un serveur d'informations (publisher). Utilisez les méthodes createPublisher et createSubscriber comme suit :

```
// Création d'un serveur d'informations publiant la rubrique indiquée
 TopicPublisher pub = session.createPublisher( topic );
// Création d'un souscripteur abonné à la rubrique indiquée
 TopicSubscriber sub = session.createSubscriber( topic );
```

Publication des messages

L'objet TopicPublisher pub permet de publier des messages, à l'instar de l'objet QueueSender qui est utilisé dans le domaine point à point. Le fragment de code ci-après crée un objet TextMessage à l'aide de la session, puis publie le message :

```
// Création de l'objet TextMessage et insertion de données dans celui-ci
TextMessage outMsg = session.createTextMessage();
outMsg.setText( "This is a short test string!" );

// Utilisation du serveur d'informations pour la publication du message
pub.publish( outMsg );
```

Réception des souscriptions

Les souscripteurs doivent pouvoir lire les souscriptions délivrées, comme indiqué dans le code suivant :

```
// Extraction de la souscription suivante en attente
TextMessage inMsg = (TextMessage)sub.receive();

// Extraction du contenu du message
String payload = inMsg.getText();
```

Ce fragment de code exécute une extraction après attente, car l'appel receive (réception) est bloqué jusqu'à ce qu'un message devienne disponible. Il existe d'autres versions de l'appel receive, telles que ″receiveNoWait″. Pour plus d'informations, reportez-vous à la section «TopicSubscriber» à la page 346.

Fermeture des ressources inutiles

Il est important de libérer toutes les ressources utilisées par l'application de publication/souscription lorsque celle-ci prend fin. Pour ce faire, lancez la méthode `close()` sur les objets qui peuvent être fermés (les serveurs d'informations, les souscripteurs, les sessions et les connexions) :

```
// Fermeture des serveurs d'informations et des souscripteurs
pub.close();
sub.close();

// Fermeture des sessions et des connexions
session.close();
conn.close();
```

Utilisation des rubriques

La présente section traite de l'utilisation des objets Topic (rubrique) JMS dans les applications MQSeries Classes pour Java - Message Service.

Noms de rubriques

La présente section décrit l'utilisation des noms de rubriques dans MQ JMS.

Remarque : La spécification JMS ne fournit pas d'information précise sur l'utilisation et la maintenance des hiérarchies de rubriques. Par conséquent, les instructions peuvent varier d'un fournisseur à l'autre.

Les noms de rubriques dans MQ JMS sont organisés en hiérarchie (figure 3 à la page 190).

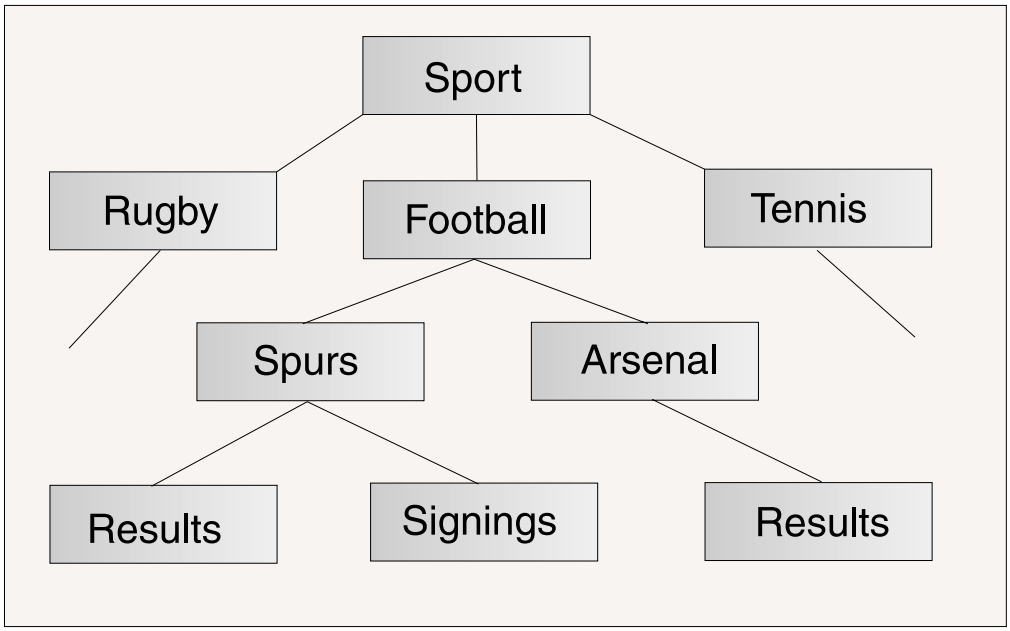


Figure 3. Hiérarchie des noms de rubriques

Dans un nom de rubrique, les différents niveaux de la hiérarchie sont séparés par une barre oblique (″/″). Par exemple, le noeud ″Signings″ est représenté par le nom de rubrique suivant :

```
Sport/Football/Spurs/Signings
```

L'utilisation de caractères génériques constitue une fonction puissante du système de rubriques MQSeries Classes pour Java - Message Service. Cette fonction permet aux souscripteurs de s'abonner simultanément à plusieurs rubriques. ″*″ remplace zéro ou plusieurs caractères et ″?″, un seul caractère.

Si un souscripteur s'abonne à un objet Topic représenté par le nom de rubrique suivant :

```
Sport/Football/*/Results
```

il reçoit des publications sur les rubriques ci-après :
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

Si le souscripteur s'abonne à la rubrique suivante :

```
Sport/Football/Spurs/*
```

il reçoit des publications sur les rubriques ci-après :
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

Vous n'avez pas besoin de gérer les hiérarchies de rubriques que vous utilisez explicitement côté courtier du système. Une fois le premier objet publisher ou susbcriber créé pour une rubrique donnée, le courtier génère l'état des rubriques publiées en cours et associées à une souscription.

Remarque : Un serveur d'informations ne peut pas publier de rubrique dont le nom contient des caractères génériques.

Création de rubriques en phase d'exécution

Il existe quatre méthodes pour la création d'objets Topic en phase d'exécution. Vous pouvez utiliser, au choix :

1. le constructeur MQTopic à argument unique,
2. le constructeur MQTopic, puis appeler la méthode setBaseTopicName(..),
3. la méthode createTopic(..) de la session,
4. la méthode createTemporaryTopic() de la session.

Méthode 1 : utilisation de MQTopic(..)

Cette méthode exige de faire référence à l'interface Topic de JMS et, par conséquent, rend le code non portable.

Le constructeur accepte un argument, celui-ci étant obligatoirement un identificateur de ressource universel (URI). Pour les objets Topic MQSeries Classes pour Java - Message Service, celui doit se présenter comme suit :

```
topic://TopicName[?property=valeur[&property=valeur]*]
```

Pour plus d'informations sur les URI et sur les paires nom-valeur admises, reportez-vous à la section «Envoi de message» à la page 181.

Le code suivant crée une rubrique pour des messages non persistants de priorité 5 :

```
// Création d'une rubrique à l'aide du constructeur MQTopic à argument unique
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

Méthode 2 : utilisation de MQTopic(), puis de setBaseTopicName(..)

Cette méthode utilise le constructeur MQTopic par défaut et, par conséquent, rend le code non portable.

Une fois l'objet Topic créé, définissez la propriété baseTopicName à l'aide de la méthode setBaseTopicName, en transmettant le nom de rubrique requis.

Remarque : Le nom de rubrique utilisé ici ne se présente pas sous la forme d'un URI et ne peut comporter aucune paire nom-valeur. Pour définir les paires nom-valeur, employez la méthode ″set″ (reportez-vous à la section «Définition de propriétés à l'aide de la méthode 'set'» à la page 183). Le code suivant crée une rubrique à l'aide de cette méthode :

```
// Création d'une rubrique à l'aide du constructeur MQTopic par défaut
Topic rtTopic = new MQTopic();

// Définition des propriétés de l'objet à l'aide des méthodes set
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

Méthode 3 : utilisation de session.createTopic(..)

Vous pouvez également créer un objet Topic à l'aide de la méthode createTopic de l'objet TopicSession, qui accepte un URI de rubrique comme suit :

```
// Création d'une rubrique à l'aide de la méthode de la fabrique de session
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Méthode 4 : utilisation de session.createTemporaryTopic()

Un objet TemporaryTopic est une rubrique disponible uniquement pour les souscripteurs créés par la même connexion TopicConnection. Cet objet TemporaryTopic est créé comme suit :

```
// Création d'une rubrique temporaire à l'aide de la méthode de la fabrique de session
Topic rtTopic = session.createTemporaryTopic();
```

Options relatives aux souscripteurs

Les objets subscriber (souscripteur) de JMS permettent plusieurs modes d'utilisation. La présente section en décrit quelques exemples.

JMS accepte deux types d'objets susbcriber :

Objets subscriber non durables
Les souscripteurs ne reçoivent des messages relatifs à la rubrique choisie que s'ils sont actifs lors de la publication de ces messages.

Objets subscriber durables
Les souscripteurs reçoivent tous les messages relatifs à une rubrique, même s'ils sont inactifs lors de la publication de ces messages.

Création d'objets subscriber non durables

L'objet subscriber créé à la section «Création de clients et de producteurs de publications» à la page 188 est de type non durable. Il est généré par le code suivant :

```
// Création d'un souscripteur abonné à la rubrique indiquée
 TopicSubscriber sub = session.createSubscriber( topic );
```

Création d'objets subscriber durables

La création d'un objet subscriber durable est similaire à celle d'un objet subscriber non durable, excepté que vous devez entrer un nom identifiant le souscripteur de manière unique :

```
// Création d'un objet subscriber durable portant un nom d'identification unique
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

Les objets subscriber non durables résilient eux-mêmes leur souscription lors de l'appel de la méthode `close()` (ou à la fin de leur période d'activité). En revanche, vous devez notifier explicitement au système votre décision de mettre fin à une souscription durable. Pour ce faire, utilisez la méthode `unsubscribe()` de la session en transmettant le nom unique sous lequel le souscripteur a été créé :

```
// Radiation de l'objet subscriber durable créé à l'étape précédente
session.unsubscribe( "D_SUB_000001" );
```

Un objet subscriber durable est créé au niveau du gestionnaire de files d'attente indiqué par le paramètre correspondant de l'objet MQTopicConnectionFactory. Si vous tentez ultérieurement de créer un objet subscriber durable de même nom, mais au niveau d'un gestionnaire de files d'attente différent, vous obtiendrez en retour un nouvel objet subscriber durable, totalement distinct.

Utilisation des sélecteurs de messages

Les sélecteurs de messages permettent d'éviter les messages qui ne correspondent pas aux critères de filtrage définis. Pour plus d'informations sur les sélecteurs de messages, reportez-vous à la section «Sélecteurs de messages» à la page 185. Les sélecteurs de messages sont associés à un souscripteur comme suit :

```
//  Association d'un sélecteur de messages à un objet subscriber non durable
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

Suppression de publications locales

Vous pouvez créer un souscripteur qui ignore les publications publiées sur sa propre connexion. Pour ce faire, attribuez comme suit la valeur true (vrai) au troisième paramètre de l'appel createSubscriber :

```
// Création d'un objet subscriber non durable associé à l'option noLocal
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

Combinaison d'options relatives aux souscripteurs

Vous pouvez combiner les options relatives aux souscripteurs pour créer, par exemple, un objet subscriber durable qui applique un sélecteur de messages et ignore les publications locales. Le fragment de code suivant illustre l'utilisation d'options combinées :

```
// Création d'un objet subscriber durable associé aux options noLocal et selector
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                       selector, true );
```

Configuration de la file d'attente de base du souscripteur

La version 5.2 de MQ JMS offre deux modes pour la configuration des souscripteurs :

- Mode des files d'attente multiples

  Chaque souscripteur possède une file d'attente exclusive à partir de laquelle il extrait tous ses messages. JMS crée une file d'attente par souscripteur. C'est le seul mode disponible dans la version 1.1 de MQ JMS.

- Mode de la file d'attente partagée

  Le souscripteur utilise une file d'attente partagée, à partir de laquelle il extrait ses messages, comme les autres souscripteurs. Ce mode ne requiert qu'une seule file d'attente pour plusieurs souscripteurs. C'est le mode par défaut utilisé dans la version 5.2 de MQ JMS.

La version 5.2 de MQ JMS permet de choisir le mode voulu et de configurer les files d'attente à utiliser.

En général, le mode de la file d'attente partagée offre un léger avantage sur le plan des performances. Pour les systèmes à débit élevé, il confère d'importants avantages en matière d'architecture et d'administration, en raison de la réduction importante du nombre de files d'attente nécessaires.

Toutefois, l'utilisation des files d'attente multiples est justifiée dans certains cas pour les motifs suivants :

- La capacité physique théorique de stockage des messages est supérieure.

  Une file d'attente MQSeries ne peut pas contenir plus de 64 0000 messages, et dans le cas d'une file d'attente partagée, cette capacité doit être divisée entre tous les souscripteurs concernés. Cette question est particulièrement importante pour les objets subscriber durables dont la durée de vie est généralement plus longue que celle des objets subscriber non durables et pour lesquels un plus grand nombre de messages peut s'accumuler.

- L'administration externe des files d'attente de souscription est facilitée.

  Dans certains types d'application, les administrateurs peuvent surveiller l'état et la longueur de files d'attente de souscripteurs déterminés. Cette tâche est beaucoup plus facile lorsqu'il existe une correspondance bi-univoque entre le souscripteur et la file d'attente.

Configuration par défaut

La configuration par défaut utilise les files d'attente de souscription partagées suivantes :

- SYSTEM.JMS.ND.SUBSCRIPTION.QUEUE pour les souscriptions non durables,
- SYSTEM.JMS.D.SUBSCRIPTION.QUEUE pour les souscriptions durables.

Ces files d'attente sont créées automatiquement lorsque vous exécutez le script MQJMS_PSQ.MQSC.

Si nécessaire, vous pouvez indiquer d'autres files d'attente physiques. Vous pouvez également modifier la configuration pour utiliser des files d'attente multiples.

Configuration des objets subscriber non durables

Vous pouvez définir la propriété relative au nom de file d'attente souscripteur non durable de l'une des façons suivantes :

- Définissez la propriété BROKERSUBQ à l'aide de l'outil d'administration MQ JMS (pour les objets extraits de JNDI).
- Utilisez la méthode setBrokerSubQueue() dans votre programme.

Pour les souscriptions non durables, le nom de file d'attente que vous indiquez doit commencer par les caractères suivants :

SYSTEM.JMS.ND.

Pour sélectionner le mode de la file d'attente partagée, indiquez le nom explicite de la file d'attente à partager. Vous devez désigner une file d'attente qui existe déjà physiquement pour pouvoir créer la souscription.

Pour sélectionner le mode des files d'attente multiples, indiquez un nom de file d'attente qui se termine par un astérisque (*). Par la suite, chaque souscripteur associé à ce nom de file d'attente peut créer une file d'attente dynamique exclusivement destinée à son usage personnel. MQ JMS crée ce type de file d'attente à partir de sa propre file d'attente modèle interne. Par conséquent, dans le cas des files d'attente multiples, toutes les files d'attente requises sont créées en mode dynamique.

Dans le cas des files d'attente multiples, vous ne pouvez pas entrer de nom explicite de file d'attente. Vous pouvez cependant indiquer un préfixe de file d'attente qui permet de créer différents domaines de file d'attente souscripteur. Par exemple, vous pouvez utiliser :

SYSTEM.JMS.ND.MYDOMAIN.*

Les caractères qui précèdent l'astérisque (*) servent de préfixe. Par conséquent, toutes les files d'attente dynamiques associées à cette souscription portent des noms commençant par SYSTEM.JMS.ND.MYDOMAIN.

Configuration des objets subscriber durables

Comme indiqué précédemment, l'utilisation de files d'attentes multiples peut être entièrement justifiée pour les souscriptions durables. Les souscriptions de ce type possèdent probablement une durée de vie plus longue et, de ce fait, un grand nombre de messages non extraits peut s'accumuler dans la file d'attente.

Par conséquent, la propriété relative au nom de file d'attente de souscripteur durable est définie dans l'objet Topic, offrant une gestion plus facile qu'au niveau de la fabrique TopicConnectionFactory. Ainsi, vous pouvez indiquer des noms

différents de files d'attente souscripteur, sans devoir recréer plusieurs objets à partir de la fabrique TopicConnectionFactory.

Vous pouvez définir le nom de file d'attente de souscripteur durable de l'une des façons suivantes :

- Définissez la propriété BROKERDURSUBQ à l'aide de l'outil d'administration MQ JMS (pour les objets extraits de JNDI).
- Utilisez la méthode setBrokerDurSubQueue() dans votre programme :

```
// Définition du nom de file d'attente souscripteur durable MQTopic
// à l'aide du mode des files d'attente multiples
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

  Une fois initialisé, l'objet Topic est transmis à la méthode TopicSession createDurableSubscriber() pour la création de la souscription indiquée :

```
// Création d'un objet subscriber durable avec l'objet Topic précédent
TopicSubscriber sub = new session.createDurableSubscriber
                                          (sportsTopic, "D_SUB_SPORT_001");
```

Pour les souscriptions durables, le nom de file d'attente que vous indiquez doit commencer par les caractères suivants :

SYSTEM.JMS.D.

Pour sélectionner le mode de la file d'attente partagée, indiquez le nom explicite de la file d'attente à partager. Vous devez désigner une file d'attente qui existe déjà physiquement pour pouvoir créer la souscription.

Pour sélectionner le mode des files d'attente multiples, indiquez un nom de file d'attente qui se termine par un astérisque (*). Par la suite, chaque souscripteur associé à ce nom de file d'attente peut créer une file d'attente dynamique exclusivement destinée à son usage personnel. MQ JMS crée ce type de file d'attente à partir de sa propre file d'attente modèle interne. Par conséquent, dans le cas des files d'attente multiples, toutes les files d'attente requises sont créées en mode dynamique.

Dans le cas des files d'attente multiples, vous ne pouvez pas entrer de nom explicite de file d'attente. Vous pouvez cependant indiquer un préfixe de file d'attente qui permet de créer différents domaines de file d'attente souscripteur. Par exemple, vous pouvez utiliser :

SYSTEM.JMS.D.MYDOMAIN.*

Les caractères qui précèdent l'astérisque (*) servent de préfixe. Par conséquent, toutes les files d'attente dynamiques associées à cette souscription portent des noms commençant par SYSTEM.JMS.D.MYDOMAIN.

Remarques sur la recréation et la migration des objets subscriber durables

Dans le cas d'un objet subscriber durable, ne tentez pas de reconfigurer le nom de file d'attente associé tant que ce souscripteur n'a pas été radié. Exécutez la méthode unsubscribe(), puis recréez entièrement la file d'attente (n'oubliez pas que les anciens messages du souscripteur sont supprimés).

Toutefois, si vous avez créé le souscripteur sous la version 1.1 de MQ JMS, celui-ci sera reconnu lors de la migration vers la version en cours. Vous n'avez pas besoin de supprimer la souscription qui reste valide, et fonctionne sous le mode des files d'attente multiples.

Résolution des incidents de publication/souscription

La présente section décrit quelques incidents pouvant se produire lorsque vous développez des applications client JMS qui utilisent le domaine de publication / souscription (publish / subscribe). Notez que seuls les incidents spécifiques de ce domaine sont traités. Pour consulter des instructions plus générales sur l'identification et la résolution des incidents, reportez-vous aux sections «Traitement des erreurs» à la page 186 et «Résolution des incidents» à la page 30.

Fermeture incomplète d'une application de publication/souscription

Il est important que les applications client JMS abandonnent toutes les ressources externes lorsqu'elles prennent fin. Pour ce faire, lancez la méthode `close()` sur tous les objets qui peuvent être fermés dès qu'ils ne sont plus nécessaires. Pour le domaine Publication / souscription, ces objets sont :

- TopicConnection,
- TopicSession,
- TopicPublisher,
- TopicSubscriber.

Cette tâche est facilitée par MQSeries Classes pour Java - Message Service, qui permet une fermeture en cascade. L'appel de la méthode close sur l'objet TopicConnection entraîne l'appel de cette même méthode sur chaque objet TopicSession créé, puis sur tous les objets TopicSubscriber et TopicPublisher générés par les sessions.

Par conséquent, la libération correcte des ressources externes exige l'appel de la méthode `connection.close()` pour chaque connexion créée par une application.

L'exécution de la procédure ″close″ peut échouer, par exemple :

- en cas de perte d'une connexion client MQSeries, ou
- en cas d'arrêt imprévu de l'application.

Dans cette situation, la méthode `close()` n'est pas appelée et les ressources externes restent ouvertes pour le compte de l'application fermée. Les principales conséquences en sont les suivantes :

Incohérence de l'état du courtier
Le courtier de messages MQSeries peut contenir des informations relatives à l'inscription d'objets subscriber et publisher qui n'existent plus. Cela signifie que le courtier peut continuer à expédier des messages à des souscripteurs qui ne les recevront jamais.

Conservation des messages et des files d'attente des souscripteurs
La procédure de radiation d'un souscripteur consiste en partie à retirer les messages associés à celui-ci. La file d'attente MQSeries sous-jacente éventuellement utilisée pour la réception des souscriptions est également enlevée. En l'absence de procédure de fermeture normale, ces messages et ces files d'attente sont conservés. En cas d'incohérence de l'état du courtier, les files d'attente continuent à recevoir de messages qui ne seront jamais lus.

Utilitaire de nettoyage d'objets subscriber

MQ JMS comporte un utilitaire de nettoyage d'objets subscriber pour éviter les incidents liés à la fermeture brutale d'objets subscriber. L'utilitaire s'exécute pour

un gestionnaire de files d'attente lors de l'initialisation du premier objet TopicConnection qui utilise ce gestionnaire. Si tous les objets TopicConnection pour un gestionnaire de files d'attente déterminé se ferment, l'utilitaire s'exécute à nouveau lors de l'initialisation de l'objet TopicConnection suivant.

L'utilitaire de nettoyage tente de détecter les incidents de publication / souscription MQ JMS précédents qui ont pu se produire à partir d'autres applications. S'il en détecte, il nettoie les ressources correspondantes par :

- la radiation du souscripteur auprès du courtier de messages MQSeries,
- le nettoyage des messages non extraits et des files d'attente associés à la souscription.

L'utilitaire de nettoyage s'exécute de manière transparente en arrière-plan pendant une courte période de temps. Il n'a généralement aucune incidence sur les autres opérations MQ JMS. Lorsque de nombreux incidents sont détectés pour un gestionnaire de files d'attente déterminé, l'initialisation peut être légèrement retardée lors du nettoyage des ressources.

Remarque : Nous vous recommandons fortement d'utiliser dès que possible la procédure de fermeture d'objets subscriber pour éviter l'accumulation d'incidents liés aux souscripteurs.

Traitement des rapports envoyés par le courtier

MQ JMS confirme les commandes d'inscription et de radiation à l'aide des messages de rapport envoyés par le courtier. Ces rapports sont normalement utilisés par MQSeries Classes pour Java - Message Service, mais ils peuvent rester dans la file d'attente dans certains cas. Les messages sont envoyés dans la file d'attente `SYSTEM.JMS.REPORT.QUEUE` associée au gestionnaire de files d'attente local.

L'application Java `PSReportDump`, fournie avec MQSeries Classes pour Java - Message Service, vide le contenu de la file d'attente dans un format texte. Les informations peuvent ensuite être analysées par l'utilisateur ou par le support technique IBM. L'application permet également d'effacer des messages de la file d'attente après le diagnostic ou la résolution d'un incident.

La version compilée de l'outil est installée dans le répertoire `<MQ_JAVA_INSTALL_PATH>/bin`. Pour appeler cet outil, accédez à ce répertoire, puis exécutez la commande suivante :

```
java PSReportDump [-m queueManager] [-clear]
```

où :

-m queueManager
= nom du gestionnaire de files d'attente à utiliser.

-clear = efface les messages de la file d'attente dont le contenu a été vidé.

Les résultats sont envoyés à l'écran et peuvent être réacheminés vers un fichier.

Chapitre 12. Messages JMS

Les messages JMS comprennent les parties suivantes :

En-tête Tous les messages prennent en charge le même jeu de zones d'en-tête. Les zones d'en-tête contiennent des valeurs qui permettent aux clients et aux fournisseurs d'identifier et d'acheminer les messages.

Propriétés Chaque message contient une fonction intégrée pour la prise en charge des valeurs de propriétés définies par l'application. Les propriétés offrent une méthode de filtrage efficace des messages définis par l'application.

Corps JMS définit plusieurs types de corps de message s'appliquant la plupart des styles de message en vigueur.

Cinq types de corps de message sont disponibles :

Flot Flot de valeurs Java primitive. L'écriture et la lecture de ce flot s'effectuent en mode séquentiel.

Mappe Ensemble de paires nom-valeur dans lesquelles les noms sont de type String (chaîne) et les valeurs de type Java primitive. Ces entrées sont accessibles en mode séquentiel ou en mode direct par le nom. L'ordre des entrées n'est pas défini.

Texte Message contenant une chaîne java.util.String.

Objet Message contenant un objet Java sérialisable.

Octets Flots d'octets non interprétés. Ce type de message permet de coder en littéral un corps de message pour qu'il corresponde à un format de message existant.

La zone d'en-tête JMSCorrelationID permet de relier deux messages, en général, une réponse et la demande correspondante. JMSCorrelationID peut contenir un ID message propre au fournisseur, une chaîne (String) propre à l'application, ou une valeur fournisseur en octets.

Sélecteurs de messages

Un message contient une fonction intégrée pour la prise en charge des valeurs de propriétés définies par l'application. Cette fonction permet d'ajouter dans un message des zones d'en-tête propres à l'application. A l'aide des propriétés, une application peut, par des sélecteurs de messages, permettre à un fournisseur JMS de sélectionner ou de filtrer des messages pour son compte, sur des critères propres à cette application. Les propriétés définies par l'application respectent les règles suivantes :

- Les noms de propriétés peuvent suivre les règles applicables à un identificateur de sélecteur de messages.
- Les types de valeur de propriété possibles sont les suivants : boolean (booléen), byte (octet), short (entier court), int (entier), long (entier long), float (variable flottante), double (valeur double) et string (chaîne).

- Les préfixes de noms ci-après sont réservés : JMSX, JMS_.

Les valeurs de propriétés sont définies avant l'envoi d'un message. Lorsqu'un client reçoit un message, les propriétés de ce message sont accessibles en lecture seule. Si un client tente de définir les propriétés à ce stade, une exception MessageNotWriteableException est émise. Si la méthode clearProperties est appelée, les propriétés deviennent accessibles en lecture et en écriture.

Une valeur de propriété peut dupliquer ou non une valeur dans un corps de message. JMS ne définit aucune stratégie relative aux éléments à insérer ou non dans une propriété. Toutefois, les développeurs d'applications doivent prendre note que les fournisseurs JMS traitent probablement avec plus d'efficacité les données intégrées au corps de message que celles contenues dans les propriétés de message. Pour assurer de meilleures performances, les applications ne doivent utiliser les propriétés des messages que lorsqu'elles ont besoin de personnaliser un en-tête de message, notamment dans le but de prendre en charge la sélection de messages personnalisés.

Un sélecteur de messages JMS permet à un client d'indiquer par l'en-tête de message les messages qui l'intéressent. Seuls les messages dont l'en-tête correspond au sélecteur sont livrés.

Les sélecteurs de messages ne peuvent pas utiliser de valeur de corps de message.

Un sélecteur de messages et un message sont en concordance lorsque le sélecteur évalue que la condition suivante est vérifiée : la zone d'en-tête et les valeurs de propriétés du message sont remplacées par leurs identificateurs respectifs dans le sélecteur.

Un sélecteur de messages est une chaîne (String) dont la syntaxe découle d'un sous-ensemble de la syntaxe des expressions conditionnelles SQL92. Un sélecteur de messages est évalué de la gauche vers la droite, en fonction d'un niveau de priorité. Vous pouvez modifier cet ordre avec des parenthèses. Aucune distinction entre les majuscules et les minuscules n'est effectuée dans les littéraux prédéfinis et les noms d'opérateurs, même s'ils sont indiqués en majuscules dans les paragraphes ci-après.

Un sélecteur peut contenir les éléments suivants :

- Littéraux :
  - Un littéral de type chaîne est placé entre apostrophes. Une apostrophe doublée représente une apostrophe simple. Par exemple, 'littéral' et 'littéral d''un sélecteur'. A l'instar des littéraux chaîne Java, ils utilisent le codage Unicode.
  - Un littéral numérique exact est une valeur numérique sans séparateur décimal (par exemple 57, -957, +62). Les valeurs de type nombre entier Java (long) sont prises en charge.
  - Un littéral numérique arrondi est une valeur numérique en notation scientifique (par exemple 7E3 ou -57,9E2) ou une valeur numérique décimale (par exemple 7., -95,7 ou +6,2). Les valeurs doubles Java sont prises en charge.
  - TRUE et FALSE sont des littéraux booléens.
- Identificateurs :
  - Un identificateur est une suite de longueur non limitée de lettres Java et de chiffres Java, dont l'initiale est obligatoirement une lettre Java. Une lettre est un caractère pour lequel la méthode Character.isJavaLetter renvoie la valeur

true. Il peut s'agir des caractères ″_″ et ″$″. Une lettre ou un chiffre est un caractère pour lequel la méthode Character.isJavaLetterOrDigit renvoie la valeur true.

  - Les noms NULL, TRUE et FALSE ne peuvent pas être des identificateurs.
  - Les opérateurs NOT, AND, OR, BETWEEN, LIKE, IN et IS ne peuvent pas être des identificateurs.
  - Les identificateurs sont des références à des zones d'en-tête ou à des propriétés.
  - Les identificateurs respectent la distinction entre les majuscules et les minuscules.
  - Les références aux zones d'en-tête de message sont les uniquement les suivantes :
    - JMSDeliveryMode,
    - JMSPriority,
    - JMSMessageID,
    - JMSTimestamp,
    - JMSCorrelationID,
    - JMSType.

    JMSMessageID, JMSTimestamp, JMSCorrelationID et JMSType peuvent avoir des valeurs non définies qui, dans ce cas, sont traitées en tant que valeurs NULL.
  - Tout nom commençant par ″JMSX″ est un nom de propriété défini par JMS.
  - Tout nom commençant par ″JMS_″ est un nom de propriété propre au fournisseur.
  - Tout nom ne commençant pas par ″JMS″ est un nom de propriété propre à l'application. Lorsqu'il est fait référence à une propriété qui n'existe pas dans un message, la valeur de celle-ci est NULL. Si la propriété existe, elle prend la valeur de propriété correspondante.
- Le caractère blanc est identique à celui défini pour Java : espace, tabulation horizontale, saut de page et fin de ligne.
- Expressions :
  - Un sélecteur est une expression conditionnelle. La concordance est ou n'est pas vérifiée selon que la condition évaluée par le sélecteur a pour valeur true (vrai) ou false (faux).
  - Les expressions arithmétiques sont composées d'expressions et d'opérations arithmétiques, d'identificateurs (avec des valeurs traitées en tant que littéraux numériques) et de littéraux numériques.
  - Les expressions conditionnelles sont composées d'expressions conditionnelles, d'opérations de comparaison et d'opérations logiques.
- La mise entre parenthèses standard () pour la définition de l'ordre d'évaluation des expressions est prise en charge.
- Les opérateurs logiques sont les suivants, dans l'ordre de priorité : NOT, AND, OR.
- Les opérateurs de comparaison sont les suivants : =, >, >=, <, <=, <> (différent de).
  - Seules des valeurs de type identique peuvent être comparées. Toutefois, il existe une exception : vous pouvez comparer des valeurs numériques exactes et des valeurs numériques arrondies. (La conversion de type nécessaire est définie par les règles de promotion numérique Java.) Si vous tentez de comparer des types différents, le sélecteur prendra la valeur false.

  - La comparaison de chaînes et de variables booléennes est limitée à = et <>. Deux chaînes sont égales si et seulement si elles contiennent la même séquence de caractères.
- Les opérateurs arithmétiques sont les suivants, 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. Si vous tentez de les utiliser, le sélecteur prendra la valeur false.
  - Les opérations arithmétiques doivent employer la promotion numérique Java.
- Opérateur de comparaison expression-arithmétique1 [NOT] BETWEEN expression-arithmétique2 AND expression-arithmétique3. Par exemple :
  - âge BETWEEN 15 AND 19 est l'équivalent de âge >= 15 AND âge <= 19.
  - âge NOT BETWEEN 15 AND 19 est l'équivalent de âge < 15 OR âge > 19.
  - L'opération BETWEEN prend la valeur false lorsqu'elle comporte une expression de type NULL. L'opération NOT BETWEEN prend la valeur true lorsqu'elle comporte une expression de type NULL.
- Opérateur de comparaison identificateur [NOT] IN (littéral-chaîne1, littéral-chaîne2,...), dans lequel l'identificateur a une valeur de type String (chaîne) ou NULL. Par exemple :
  - L'expression Pays IN (' UK', 'US', 'France') est vraie pour 'UK' et fausse pour 'Pérou'. Elle équivaut à l'expression (Pays = ' UK') OR (Pays = ' US') OR (Pays = 'France').
  - L'expression Pays NOT IN (' UK', 'US', 'France') est fausse pour 'UK' et vraie pour 'Pérou'. Elle équivaut à l'expression NOT ((Pays = ' UK') OR (Pays = ' US') OR (Pays = ' France')).
  - Dans le cas d'un identificateur de type NULL dans une opération IN ou NOT IN, la valeur de cette opération est inconnue.
- Opérateur de comparaison identificateur [NOT] LIKE valeur-masque [ESCAPE caractère-échappement], dans lequel l'identificateur a une valeur de type String (chaîne). La variable valeur-masque est un littéral de type chaîne, dans lequel ″_″ représente tout caractère simple et ″%″, toute séquence de caractères (y compris la séquence vide). Tous les autres caractères représentent leurs valeurs respectives. La variable caractère-échappement (facultative) est un littéral de type chaîne monocaractère qui sert de caractère d'échappement pour la signification spéciale de ″_″ et de ″%″ dans la variable valeur-masque. Par exemple :
  - L'expression téléphone LIKE '12%3' est vraie pour '123' '12993' et fausse pour '1234'.
  - L'expression LIKE 'r_se' est vraie pour 'rose' et fausse pour 'rosse'.
  - L'expression LIKE '_%' ESCAPE '\' est vraie pour '_fichier' et fausse pour 'barre'.
  - L'expression téléphone NOT LIKE '12%3' est fausse pour '123' '12993' et vraie pour '1234'.
  - Dans le cas d'un idenficateur de type NULL dans une opération LIKE ou NOT LIKE, la valeur de cette opération est inconnue.
- L'opérateur de comparaison IS NULL vérifie la présence d'une valeur de type NULL dans une zone d'en-tête ou l'absence d'une valeur de propriété. Par exemple :
  - nom_prop IS NULL.

- L'opérateur de comparaison IS NOT NULL vérifie la présence d'une valeur de type non NULL dans une zone d'en-tête ou d'une valeur de propriété. Par exemple :
  - nom_prop IS NOT NULL.

Le sélecteur de messages ci-après permet de retenir les messages correspondant au type ″véhicule″, à la ″couleur bleue″ et à un ″poids″ supérieur à 1000 kg :

```
"JMSType = 'véhicule' AND couleur = 'bleue' AND poids > 1000"
```

Comme indiqué précédemment, les valeurs de propriétés peuvent être de type NULL. L'évaluation des expressions de sélecteur qui contiennent des valeurs NULL est définie par la sémantique NULL SQL92. En voici une description succincte :

- SQL considère qu'une valeur NULL est inconnue.
- Les comparaisons ou les expressions arithmétiques qui comportent une valeur inconnue ont toujours pour résultat une valeur inconnue.
- Les opérateurs NULL et IS NOT NULL convertissent une valeur inconnue respectivement en valeurs TRUE et FALSE.

Même si SQL prend en charge les comparaisons et les expressions arithmétiques à décimales fixes, les sélecteurs de messages JMS ne les acceptent pas. C'est pourquoi seuls les littéraux numériques exacts sans valeur décimale sont admis. C'est également la raison pour laquelle il existe des expressions numériques comportant une valeur décimale en tant que représentation de remplacement d'une valeur numérique arrondie.

Les commentaires SQL ne sont pas pris en charge.

Mappage de messages JMS vers les messages MQSeries

La présente section décrit le mode selon lequel la structure de message JMS (traitée dans la première partie de ce chapitre) est mappée vers un message MQSeries. Elle intéresse particulièrement les programmeurs qui souhaitent transmettre des messages entre des applications JMS et des applications MQSeries classiques. Elle s'adresse également aux utilisateurs qui veulent manipuler des messages transmis entre deux applications JMS, par exemple dans une structure où le courtier de messages est mis en oeuvre.

Les messages MQSeries comportent trois composants :
- le descripteur de message MQSeries (MQMD),
- un en-tête MQSeries MQRFH2,
- le corps de message.

L'en-tête MQRFH2 est facultatif et son insertion dans un message sortant est contrôlée par un indicateur qui figure dans la classe cible JMS. Vous pouvez définir cet indicateur à l'aide de l'outil d'administration JMS MQSeries. L'en-tête MQRFH2 contient des informations propres à JMS et, par conséquent, vous devez toujours l'insérer dans le message lorsque l'expéditeur sait que le destinataire est une application JMS. Normalement, vous pouvez omettre l'en-tête MQRFH2 lorsque le message est directement envoyé à une application non JMS (application native MQSeries). En effet, une application de ce type n'attend pas d'en-tête MQRFH2 dans le message MQSeries.

Mappage des messages JMS

La figure 4 illustre la transformation des structures :

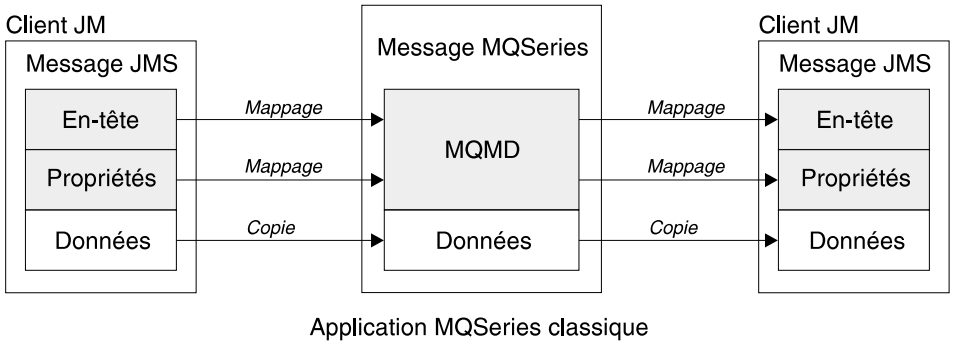


Figure 4. Modèle de mappage entre JMS et MQSeries

La transformation des structures est effectuée de deux façons :

Par mappage
Lorsque le descripteur MQMD comporte une zone équivalente à la zone JMS, celle-ci est mappée vers la zone MQMD. Des zones MQMD supplémentaires sont présentées en tant que propriétés JMS, car une application JMS peut avoir besoin d'extraire ou de définir ces zones lorsqu'elle dialogue avec une application non JMS.

Par copie
En l'absence de zone MQMD équivalente, une zone d'en-tête ou une propriété JMS est transmise, après transformation éventuelle, en tant que zone de l'en-tête MQRFH2.

En-tête MQRFH2

La présente section décrit l'en-tête MQRFH version 2 qui contient des données propres à JMS associées au contenu du message. L'en-tête MQRFH2 version 2 est extensible et peut contenir d'autres informations non directement liées à JMS. Toutefois, seule l'utilisation par JMS est traitée dans cette section.

L'en-tête comporte deux sections, l'une fixe et l'autre variable.

Section fixe
La section fixe est basée sur le modèle d'en-tête MQSeries standard 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 selon la procédure habituelle.

Version (MQLONG)
Numéro de version de la structure.

Doit être MQRFH_VERSION_2 (valeur: 2) (valeur initiale).

StrucLength (MQLONG)
Longueur totale de l'en-tête MQRFH2, zones NameValueData incluses.

La valeur entrée dans StrucLength doit être un multiple de 4 (la valeur de la zone NameValueData peut être complétée par des espaces si nécessaire).

Encoding (MQLONG)
Codage des données.

Codage des données numériques dans la section de message placée après l'en-tête MQRFH2 (l'en-tête suivant ou les données de message figurant après cet en-tête).

CodedCharSetId (MQLONG)
ID de jeu de caractères codés (CCSID).

Représentation des données de type caractères dans la section de message placée après l'en-tête MQRFH2 (l'en-tête suivant ou les données de message figurant après cet en-tête).

Format (MQCHAR8)
Nom de format.

Nom du format applicable à la section de message placée après l'en-tête MQRFH2.

Indicateurs (MQLONG)
Indicateurs.

MQRFH_NO_FLAGS =0. Aucun indicateur n'est défini.

NameValueCCSID (MQLONG)
ID de jeu de caractères codés (CCSID) pour les chaînes de caractères NameValueData contenues dans l'en-tête. Les chaînes NameValueData peuvent être codées dans un jeu de caractères différent de celui adopté pour les autres chaînes de caractères de l'en-tête (StrucID et Format).

Si la valeur entrée dans la zone NameValueCCSID est un CCSID au format Unicode à deux octets (1200, 13488 ou 17584), l'ordre des octets Unicode est identique à celui des zones numériques de l'en-tête MQRFH2. Par exemple : Version, StrucLength, NameValueCCSID.

La zone NameValueCCSID accepte uniquement des valeurs de la liste suivante :

1200	UCS2 extensible
1208	UTF8
13488	Sous-ensemble UCS2 2.0
17584	Sous-ensemble UCS2 2.1 (dont le symbole de l'euro)

Section variable
La section variable est placée à la suite de la section fixe. Elle contient un nombre variable de dossiers MQRFH2. Chaque dossier comporte un nombre variable d'éléments ou des propriétés. Les dossiers regroupent des propriétés apparentées. Les en-têtes MQRFH2 créés par JMS peuvent contenir jusqu'à trois dossiers :

Dossier <mcd>
Contient des propriétés qui décrivent la forme (shape) ou le format du message. Par exemple, la propriété `msd` identifie le message comme étant de type Texte, Octets, Flot, Mappe, Objet ou NULL. Ce dossier est toujours présent dans un en-tête JMS MQRFH2.

Dossier <jms>
Permet de transporter des zones d'en-tête JMS et les propriétés JMSX non entièrement exprimables dans le descripteur MQMD. Ce dossier est toujours présent dans un en-tête JMS MQRFH2.

Dossier <usr>
Permet de transporter les propriétés définies par l'application et associées au message. Ce dossier n'est présent que s'il existe des propriétés définies par l'application.

Le tableau 17 présente la liste complète des noms de propriétés.

Tableau 17. Dossiers et propriétés MQRFH2 utilisés par JMS

Zones JMS		**Zones MQRFH2**		
Nom	**Type Java**	**Nom de dossier**	**Nom de propriété**	**Type/valeurs**
JMSDestination	Destination	jms	Dst	string
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	string
JMSReplyTo	Destination	jms	Rto	string
JMSType	String	mcd	Type	string
JMSXGroupID	String	jms	Gid	string
JMSXGroupSeq	int	jms	Seq	i4
xxx (défini par l'utilisateur)	Any	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

La syntaxe qui permet d'exprimer les propriétés dans la section variable se présente comme suit :

NameValueLength (MQLONG)
Longueur (en octets) de la chaîne NameValueData placée immédiatement après cette zone (la longueur de celle-ci n'est pas incluse). La valeur entrée dans NameValueLength doit être un multiple de 4 (la valeur de la zone NameValueData peut être complétée par des espaces si nécessaire).

NameValueData (MQCHARn)
Chaîne de caractères SBCS dont la longueur en octets est indiquée par la zone NameValueLength précédente. Elle contient un dossier comportant une séquence de propriétés. Chaque propriété est un triplet ″nom/type/valeur″ d'un élément XML désigné comme suit sous le nom du dossier :

```
<foldername> triplet1 triplet2 .....   tripletn </foldername>
```

La balise de fermeture </foldername> peut être complétée par des espaces de remplissage. Chaque triplet est codé avec une syntaxe de type XML :

```
<name dt='datatype'>value</name>
```

L'élément dt='datatype' , facultatif, est omis pour de nombreuses propriétés, car leur type de données est prédéfini. Si vous incluez l'élément datatype, faites précéder la balise dt= d'un ou plusieurs espaces.

name : nom de la propriété (reportez-vous au tableau 17 à la page 206).

datatype : doit correspondre, après pliage, à un ou plusieurs littéraux du tableau 18.

value : représentation sous forme de chaîne de la valeur à transmettre (voir le tableau 18).

Une valeur NULL est codée avec la syntaxe suivante :

```
<name/>
```

Tableau 18. Types de données et valeurs des propriétés

Type de données (datatype)	**Valeur (value)**
string	Toute séquence de caractères, sauf < et &
boolean	Caractère 0 ou 1 (1 = ″true″)
bin.hex	Chiffres hexadécimaux représentant des octets
i1	Nombre composé de chiffres de 0 à 9, précédé ou non d'un signe (pas de fraction ni d'exposant). Doit être compris entre -128 et 127 inclus.
i2	Nombre composé de chiffres de 0 à 9, précédé ou non d'un signe (pas de fraction ni d'exposant). Doit être compris entre -32768 et 32767 inclus.
i4	Nombre composé de chiffres de 0 à 9, précédé ou non d'un signe (pas de fraction ni d'exposant). Doit être compris entre -2147483648 et 2147483647 inclus.
i8	Nombre composé de chiffres de 0 à 9, précédé ou non d'un signe (pas de fraction ni d'exposant). Doit être compris entre -9223372036854775808 et 9223372036854775807 inclus.
int	Nombre composé de chiffres de 0 à 9, précédé ou non d'un signe (pas de fraction ni d'exposant). Doit être compris dans la même plage de valeurs que le type de données i8. Peut remplacer l'un des types de données ″i*″ si l'expéditeur ne souhaite pas associer de type précis à la propriété.
r4	Nombre en virgule flottante, grandeur <= 3.40282347E+38, >= 1.175E-37, composé de chiffres de 0 à 9 : signe, fraction et exposant facultatifs.
r8	Nombre en virgule flottante, grandeur <= 1.7976931348623E+308, >= 2.225E-307, composé de chiffres de 0 à 9 ; signe, fraction et exposant facultatifs.

Une valeur de type string (chaîne) peut contenir des espaces. Les séquences d'échappement suivantes sont obligatoires :

& pour le caractère &

< pour le caractère <

Les séquences d'échappement suivantes sont facultatives :

> pour le caractère >
' pour le caractère '
" pour le caractère "

Zones et propriétés JMS et zones MQMD correspondantes

Le tableau 19 répertorie les propriétés qui sont directement mappées vers des zones MQMD.

Tableau 19. Mappage des propriétés JMS vers des zones MQMD

Zone JMS		**Zone MQMD**	
En-tête	**Type Java**	**Zone**	**Type C**
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	String	MessageID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	String	CorrelId	MQBYTE24
Propriétés			
JMSXUserID	String	UserIdentifier	MQCHAR12
JMSXAppID	String	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	String	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG
Propre au fournisseur			
JMS_IBM_Report_Exception	int	Report	MQLONG
JMS_IBM_Report_Expiration	int	Report	MQLONG
JMS_IBM_Report_COA	int	Report	MQLONG
JMS_IBM_Report_COD	int	Report	MQLONG
JMS_IBM_Report_PAN	int	Report	MQLONG
JMS_IBM_Report_NAN	int	Report	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Report	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Report	MQLONG
JMS_IBM_Report_Discard_Msg	int	Report	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	String	Format	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId	MQLONG

Mappage des zones JMS vers des zones MQSeries (messages sortants)

Le tableau 20 présente le mode selon lequel les zones d'en-tête et de propriétés sont mappées vers des zones MQMD/RFH2 lors de l'exécution de la méthode send() ou publish() (envoi ou publication).

Dans le cas des zones définies par l'objet message, la valeur transmise est celle contenue dans le message JMS juste avant l'exécution la méthode send() ou publish(). La valeur incluse dans le message JMS n'est pas modifiée par la méthode send/publish().

Dans le cas des zones définies par la méthode Send, une valeur est affectée lors de l'exécution de la méthode send/publish() (toute valeur contenue dans le message JMS est ignorée). La valeur incluse dans le message JMS est mise à jour pour indiquer la valeur utilisée.

Les zones définies en tant que zones de réception uniquement ne sont pas transmises et ne sont pas modifiées dans le message par la méthode send() ou publish().

Tableau 20. Mappage des zones liées aux messages sortants

Zones JMS	**Mode de transmission**		**Mode de définition**
Nom	**Zone MQMD**	**En-tête**	
JMSDestination		MQRFH2	Méthode Send
JMSDeliveryMode	Persistence	MQRFH2	Méthode Send
JMSExpiration	Expiry	MQRFH2	Méthode Send
JMSPriority	Priority	MQRFH2	Méthode Send
JMSMessageID	MessageID		Méthode Send
JMSTimestamp	PutDate/PutTime		Méthode Send
JMSCorrelationID	CorrelId	MQRFH2	Objet message
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Objet message
JMSType		MQRFH2	Objet message
JMSRedelivered			Réception uniquement
Propriétés			
JMSXUserID	UserIdentifier		Méthode Send
JMSXAppID	PutApplName		Méthode Send
JMSXDeliveryCount			Réception uniquement
JMSXGroupID	GroupId	MQRFH2	Objet message
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Objet message
Propre au fournisseur			
JMS_IBM_Report_Exception	Report		Objet message
JMS_IBM_Report_Expiration	Report		Objet message
JMS_IBM_Report_COA/COD	Report		Objet message
JMS_IBM_Report_NAN/PAN	Report		Objet message

Tableau 20. Mappage des zones liées aux messages sortants (suite)

Zones JMS	Mode de transmission		Mode de définition
Nom	**Zone MQMD**	**En-tête**	
JMS_IBM_Report_Pass_Msg_ID	Report		Objet message
JMS_IBM_Report_Pass_Correl_ID	Report		Objet message
JMS_IBM_Report_Discard_Msg	Report		Objet message
JMS_IBM_MsgType	MsgType		Objet message
JMS_IBM_Feedback	Feedback		Objet message
JMS_IBM_Format	Format		Objet message
JMS_IBM_PutApplType	PutApplType		Méthode Send
JMS_IBM_Encoding	Encoding		Objet message
JMS_IBM_Character_Set	CodedCharacterSetId		Objet message

Mappage des zones d'en-tête JMS lors de l'exécution de la méthode send() ou publish()

Les remarques ci-après concernent le mappage des zones JMS lors de l'exécution de la méthode send() ou publish() :

- **JMS Destination vers MQRFH2 :** Chaîne sérialisant les caractéristiques saillantes d'un objet destination, de sorte qu'une application JMS réceptrice peut reconstituer un objet destination équivalent. La zone MQRFH2 est codée en identicateur de ressource universel (URI) (pour plus d'informations sur la notation sous forme d'URI, reportez-vous à la section «uniform resource identifiers» à la page 181).
- **JMSReplyTo vers MQMD ReplyToQ, ReplyToQMgr, MQRFH2 :** Les noms Queue et QueueManager sont respectivement copiés dans les zones ReplyToQ et ReplyToQMgr du descripteur MQMD. Les informations relatives au suffixe destination (autres détails utiles conservés dans l'objet destination) sont copiées dans la zone MQRFH2. La zone MQRFH2 est codée en identicateur de ressource universel (URI) (pour plus d'informations sur la notation sous forme d'URI, reportez-vous à la section «uniform resource identifiers» à la page 181).
- **JMSDeliveryMode vers MQMD Persistence :** La valeur de JMSDeliveryMode est définie par la méthode send() ou publish() ou par la classe MessageProducer, sauf si elle est remplacée par l'objet destination. La valeur de JMSDeliveryMode est mappée comme suit vers la zone Persistence du descripteur MQMD :
  - la valeur JMS PERSISTENT est l'équivalent de MQPER_PERSISTENT,
  - la valeur JMS NON_PERSISTENT est l'équivalent de MQPER_NOT_PERSISTENT.

  Si JMSDeliveryMode ne contient pas de valeur par défaut, la valeur du mode de livraison est également codée dans l'en-tête MQRFH2.
- **JMSExpiration vers/à partir de MQMD Expiry, MQRFH2 :** JMSExpiration stocke le délai d'expiration (somme de l'âge actuel et de la durée de vie restante), tandis que MQMD contient la durée de vie restante. La valeur de JMSExpiration est exprimée en millisecondes et celle de MQMD.expiry, en centisecondes.
  - Si la méthode send() définit une durée de vie restante illimitée, MQMD Expiry prend pour valeur MQEI_UNLIMITED et JMSExpiration est codé dans l'en-tête MQRFH2.

  - Si la méthode send() définit une durée de vie restante inférieure à 214748364,7 secondes (environ 7 ans), la valeur correspondante est stockée dans le descripteur MQMD. L'expiration (Expiry) et le délai d'expiration (en millisecondes) sont codés sous forme de valeur i8 dans l'en-tête MQRFH2.
  - Si la méthode send() définit une durée de vie restante supérieure à 214748364,7 secondes, MQMD.Expiry prend pour valeur MQEI_UNLIMITED. Le délai d'expiration réel (en millisecondes) est codé sous forme de valeur i8 dans l'en-tête MQRFH2.
- **JMSPriority vers MQMD Priority :** Mappe directement la valeur JMSPriority (0 à 9) vers la valeur de priorité MQMD (0 à 9). Si JMSPriority ne contient pas de valeur par défaut, le niveau de priorité est également codé dans l'en-tête MQRFH2.
- **JMSMessageID à partir de MQMD MessageID :** Tous les messages envoyés à partir de JMS sont associés à des identificateurs de message uniques, affectés par MQSeries. La valeur attribuée est renvoyée dans la zone MQMD messageId après l'appel de MQPUT, puis retransmise à l'application dans la zone JMSMessageID. L'ID message (messageID) MQSeries est une valeur binaire de 24 octets, alors que l'ID message JMS (JMSMessageID) est une chaîne (String). L'ID message JMS est composé de la valeur binaire messageID convertie en séquence de 48 caractères hexadécimaux comportant le préfixe ″ID:″. JMS comporte une option permet de désactiver la génération d'identificateurs de message. Cette option est ignorée et un identificateur unique est affecté dans tous les cas. Toute valeur définie dans la zone JMSMessageId avant l'exécution de la méthode send() est remplacée.
- **JMSTimestamp à partir de MQMD PutDate, PutTime :** Après un envoi, la valeur attribuée à la zone JMSTimestamp est égale à la date et à l'heure fournies par les zones MQMD PutDate et PutTime. Toute valeur définie dans la zone JMSMessageId avant l'exécution de la méthode send() est remplacée.
- **JMSType vers MQRFH2 :** Cette chaîne est définie dans l'en-tête MQRFH2.
- **JMSCorrelationID vers MQMD CorrelId, MQRFH2 :** JMSCorrelationID peut contenir l'un des éléments suivants :
  - **ID message propre au fournisseur :** Identificateur de message provenant d'un message déjà envoyé ou reçu. Doit être une chaîne de 48 chiffres hexadécimaux comportant le préfixe ″'ID:″. Le préfixe est retiré, les caractères restants sont convertis en binaire, puis définis dans la zone CorrelId du descripteur MQMD. Aucun ID de corrélation (correlid) n'est codé dans l'en-tête MQRFH2.
  - **Valeur fournisseur en octet[]s :** Cette valeur est copiée dans la zone CorrelId du descripteur MQMD, puis complétée avec des valeurs NULL ou tronquée à 24 octets si nécessaire. Aucun ID de corrélation (correlid) n'est codé dans l'en-tête MQRFH2.
  - **Chaîne propre à l'application :** La valeur est copiée dans l'en-tête MQRFH2. Les 24 premiers octets de la chaîne (format UTF8) sont écrits dans la zone CorrelID du descripteur MQMD.

Mappage des zones de propriétés JMS

Les remarques ci-après concernent le mappage des zones de propriétés JMS dans les messages MQSeries :

- **JMSXUserID à partir de MQMD UserIdentifier :** JMSXUserID est défini lors du retour de l'appel send().
- **JMSXAppID à partir de MQMD PutApplName :** JSMXAppID est défini lors du retour de l'appel send ().

- **JMSXGroupID vers MQRFH2 (point à point) :** Dans le cas des messages point à point, JMSXGroupID est copié dans la zone GroupID du descripteur MQMD. Si JMSXGroupID commence par le préfixe ″ID:″, il est converti en binaire. Sinon, il est codé en chaîne UTF8. La valeur est complétée ou tronquée à 24 octets si nécessaire. L'indicateur MQF_MSG_IN_GROUP est défini.
- **JMSXGroupID vers MQRFH2 (publish/subscribe) :** Dans le cas des messages de publication/souscription (publish/subscribe), JMSXGroupID est copié sous forme de chaîne dans l'en-tête MQRFH2.
- **JMSXGroupSeq MQMD MsgSeqNumber (point à point) :** Dans le cas des messages point à point, JMSXGroupSeq est copié dans la zone MsgSeqNumber du descripteur MQMD. L'indicateur MQF_MSG_IN_GROUP est défini.
- **JMSXGroupSeq MQMD MsgSeqNumber (publish/subscribe) :** Dans le cas des messages de publication/souscription (publish/subscribe), JMSXGroupSeq est copié sous forme de valeur i4 dans l'en-tête MQRFH2.

Mappage des zones propres au fournisseur JMS

Les remarques ci-après concernent le mappage des zones propres au fournisseur JMS dans les messages MQSeries :

- **JMS_IBM_Report_<name> vers MQMD Report :** Une application JMS peut définir les options MQMD Report à l'aide des propriétés JMS_IBM_Report_XXX ci-après. Le descripteur MQMD unique est mappé vers plusieurs propriétés JMS_IBM_Report_XXX. L'application doit attribuer à ces propriétés une valeur égale à la valeur standard MQSeries MQRO_ constants (incluse dans com.ibm.mq.MQC). Par exemple, pour demander une notification COD (confirmation à la livraison) avec des données complètes, l'application doit attribuer à JMS_IBM_Report_COD la valeur MQC.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 vers MQMD MsgType :** La valeur établit une correspondance directe avec la zone MsgType du descripteur MQMD. Si l'application n'a pas défini de valeur explicite de type JMS_IBM_MsgType, une valeur par défaut est utilisée. Elle est déterminée comme suit :
  - si JMSReplyTo a pour valeur une destination de file d'attente MQSeries, MSGType prend la valeur MQMT_REQUEST,
  - si JMSReplyTo n'est pas défini ou que celui-ci comporte une valeur autre qu'une destination de file d'attente MQSeries, MsgType prend la valeur MQMT_DATAGRAM.
- **JMS_IBM_Feedback vers MQMD Feedback :** La valeur établit une correspondance directe avec la zone Feedback du descripteur MQMD.
- **JMS_IBM_Format vers MQMD Format :** La valeur établit une correspondance directe avec la zone Format du descripteur MQMD.
- **JMS_IBM_Encoding vers MQMD Encoding :** Lorsqu'elle est définie, cette propriété remplace le codage numérique de l'objet Destination Queue ou Topic.
- **JMS_IBM_Character_Set vers MQMD CodedCharacterSetId :** Lorsqu'elle est définie, cette propriété remplace le jeu de caractères codés de la file d'attente cible (Destination Queue) ou de la rubrique (Topic).

Mappage des zones MQSeries vers les zones JMS (messages entrants)

Le tableau 21 présente le mode selon lequel les zones d'en-tête et de propriété sont mappées vers des zones MQMD / MQRFH2 lors de l'exécution de la méthode send() ou publish() (envoi ou publication).

Tableau 21. Mappage des zones liées aux messages entrants

Zones JMS	**Extraction de**	
Nom	**Zone MQMD**	**MQRFH2**
En-têtes JMS		
JMSDestination		jms.Dst
JMSDeliveryMode	Persistence	
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MessageID	
JMSTimestamp	PutDate PutTime	
JMSCorrelationID	CorrelId	jms.Cid
JMSReplyTo	ReplyToQ ReplyToQMgr	jms.Rto
JMSType		mcd.Type
JMSRedelivered	BackoutCount	
Propriétés JMS		
JMSXUserID	UserIdentifier	

Tableau 21. Mappage des zones liées aux messages entrants (suite)

Zones JMS	**Extraction de**	
Nom	**Zone MQMD**	**MQRFH2**
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId	jms.Gid
JMSXGroupSeq	MsgSeqNumber	jms.Seq
Propre au fournisseur JMS		
JMS_IBM_Report_Exception	Report	
JMS_IBM_Report_Expiration	Report	
JMS_IBM_Report_COA	Report	
JMS_IBM_Report_COD	Report	
JMS_IBM_Report_PAN	Report	
JMS_IBM_Report_NAN	Report	
JMS_IBM_Report_ Pass_Msg_ID	Report	
JMS_IBM_Report_Pass_Correl_ID	Report	
JMS_IBM_Report_Discard_Msg	Report	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding[1]	Encoding	
JMS_IBM_Character_Set[1]	CodedCharacterSetId	
1. N'est défini que si le message entrant est un message de type Octets.		

Mappage de JMS vers une application MQSeries native

La présente section décrit les événements induits par l'envoi d'un message d'une application client JMS à une application MQSeries classique qui ignore les en-têtes MQRFH2. La figure 5 à la page 215 représente un diagramme du mappage.

L'administrateur indique que le client JMS dialogue avec cette application par l'attribution de la valeur JMSC.MQJMS_CLIENT_NONJMS_MQ à la zone TargetClient de la destination MQSeries. Cela signifie qu'aucune zone MQRFH2 ne doit être générée.

Le mappage JMS-MQMD ciblé au niveau d'une application MQSeries native est identique au mappage JMS-MQMD ciblé au niveau d'un client JMS authentique. Si JMS reçoit un message MQSeries avec une valeur autre que MQFMT_RFH2 dans la zone MQMD Format, cela signifie que les données reçues proviennent d'une application non JMS. Si la zone Format a pour valeur MQFMT_STRING, le message est reçu sous forme de message texte JMS. Sinon, il est reçu en tant que message JMS de type Octets. Il n'existe aucun en-tête MQRFH2 et, par conséquent, seules les propriétés JMS transmises dans le descripteur MQMD peuvent être restaurées.

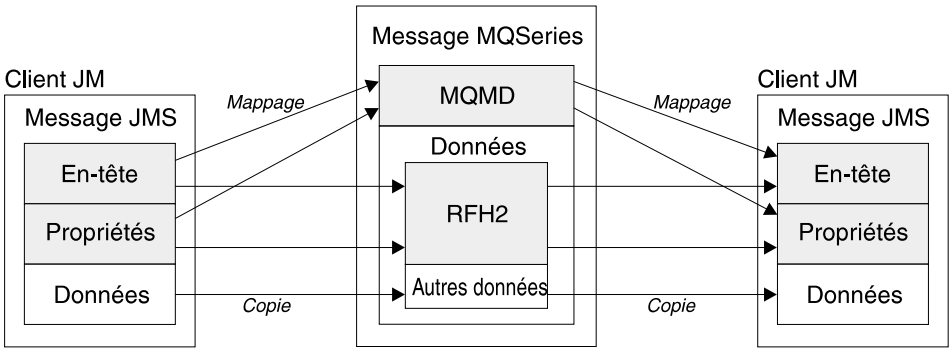


Figure 5. Modèle de mappage entre JMS et MQSeries

Corps de message

La présente section concerne le codage du corps de message. Ce codage dépend du type de message JMS :

ObjectMessage
Objet normalement sérialisé par le module d'exécution Java (Runtime).

TextMessage
Chaîne codée. Dans le cas d'un message sortant, la chaîne est codée avec le jeu de caractères indiqué par l'objet destination. Le codage par défaut est UTF8 (ce codage commence par le premier caractère du message qui n'est précédé d'aucune zone de longueur). Toutefois, il est possible d'indiquer des jeux de caractères pris en charge par MQ Java. Ceux-ci sont principalement employés lorsque vous envoyez un message à une application non JMS.

Dans le cas d'un jeu de caractères DBCS (y compris UTF16), la spécification de codage des entiers de l'objet destination 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 figurent dans l'en-tête MQSeries de droite (ou dans le descripteur MQMD en l'absence d'en-tête). Dans le cas des messages JMS, l'en-tête de droite est généralement MQRFH2.

BytesMessage
Par défaut, séquence d'octets conforme à la spécification JMS 1.0.2 et à la documentation Java associée.

Dans le cas d'un message sortant assemblé par l'application, la propriété codage de l'objet destination peut remplacer le codage des zones entier et virgule flottante contenues dans le message. Par exemple, vous pouvez demander que les valeurs en virgule flottante soient stockées au format IBM 390 plutôt qu'au format IEEE).

Un message entrant est interprété à l'aide du codage numérique spécifié dans le message lui-même. Cette spécification figure dans l'en-tête MQSeries de droite (ou dans le descripteur MQMD en l'absence d'en-tête). Dans le cas des messages JMS, l'en-tête de droite est généralement MQRFH2.

Si un message de type Octets (BytesMessage) est reçu et qu'il est renvoyé sans modification, le corps de message est transmis octet par octet, tel que. La propriété codage de l'objet destination n'a aucune incidence sur le corps de message. Une chaîne UTF8 constitue la seule entité de type chaîne pouvant être explicitement envoyée par un message d'octets. Cette chaîne est codée au format Java UTF8 et commence par une zone de longueur de 2 octets. La propriété jeu de caractères de l'objet destination n'a aucune incidence sur le codage d'un message d'octets sortant. La valeur du jeu de caractères d'un message MQSeries entrant n'a aucune incidence sur l'interprétation de ce message en tant que message JMS de type Octets.

Il est peu probable que les applications non Java puissent reconnaître le codage Java UTF8. Par conséquent, pour qu'une application JMS puisse envoyer un message d'octets qui comporte des données texte, elle doit convertir les chaînes de caractères en tableau d'octets, puis les écrire dans le message d'octets.

MapMessage

Chaîne contenant un jeu de triplets XML nom/type/valeur codés comme suit :

```
<map><elementName1 dt='datatype'>value</elementName1>
<elementName2 dt='datatype'>value</elementName2>.....
</map>
```

où :

`datatype` : une des valeurs décrites dans le tableau 18 à la page 207. `string` : type de données par défaut. Par conséquent, `dt='string'` est omis.

Le jeu de caractères, à l'aide duquel la chaîne XML qui constitue le corps du message MapMessage est codée ou interprétée, est déterminé par les règles applicables aux messages de type texte (TextMessage).

StreamMessage

Mappe sans les noms d'éléments :

```
<stream><elt dt='datatype'>value</elt>
<elt dt='datatype'>value</elt>.....</stream>
```

Chaque élément est envoyé sous le même nom de balise (elt). Le type par défaut est string (chaîne). Par conséquent `dt='string'` est omis pour les éléments de type chaîne.

Le jeu de caractères, à l'aide duquel la chaîne XML qui constitue le corps du message StreamMessage est codée ou interprétée, est déterminé par les règles applicables aux messages de type texte (TextMessage).

La zone MQRFH2.format prend la valeur suivante :

MQFMT_NONE

pour les messages objet (ObjectMessage), les messages de type Octets (BytesMessage) ou les messages sans corps de message.

MQFMT_STRING

pour les messages de texte (TextMessage), les messages de flot (StreamMessage) ou les messages de mappe (MapMessage).

Chapitre 13. Fonctions de serveur d'applications MQ JMS

MQ JMS version 5.2 prend en charge les fonctions de serveur d'applications (ASF) indiquées dans la spécification Java Message Service (JMS) 1.0.2 (consultez le site Java de Sun à l'adresse `http://java.sun.com`). Cette spécification identifie trois rôles dans le modèle de programmation :

- le **fournisseur JMS** propose des fonctionnalités client de connexion (ConnectionConsumer) et session avancée (Advanced Session),
- le **serveur d'applications** offre les fonctionnalités pool de sessions serveur (ServerSessionPool) et session serveur (ServerSession),
- l'**application client** utilise les fonctionnalités mises à disposition par le fournisseur JMS et le serveur d'applications.

Les sections suivantes contiennent des informations détaillées sur le mode selon lequel MQ JMS met en oeuvre les fonctions ASF :

- «Classes et fonctions ASF» : explique la manière dont MQ JMS met en oeuvre la classe et la fonctionnalité avancée ConnectionConsumer dans la classe Session.
- «Code exemple de serveur d'applications» à la page 224 : décrit le code exemple ServerSessionPool et ServerSession fourni avec MQ JMS.
- «Exemples d'utilisation des fonctions ASF» à la page 228 : présente les exemples ASF fournis et des exemples d'utilisation ASF côté application client.

Remarque : La spécification Java Message Service 1.0.2 pour fonctions ASF décrit également le support JMS destiné aux transactions réparties qui utilisent le protocole X/Open XA. Pour plus d'informations sur le support XA fourni par MQ JMS, reportez-vous à l'«Annexe E. Interface JMS JTA/XA avec WebSphere» à la page 371.

Classes et fonctions ASF

MQ JMS met en oeuvre la classe ConnectionConsumer et les fonctionnalités avancées dans la classe Session. Pour plus d'informations, reportez-vous aux sections suivantes :

- «MQPoolServices» à la page 129
- «MQPoolServicesEvent» à la page 130
- «MQPoolToken» à la page 132
- «MQPoolServicesEventListener» à la page 160
- «ConnectionConsumer» à la page 253
- «QueueConnection» à la page 299
- «Session» à la page 312
- «TopicConnection» à la page 330

ConnectionConsumer

La spécification JMS permet à un serveur d'applications de s'intégrer étroitement à une mise en oeuvre JMS par l'interface `ConnectionConsumer` (client de connexion). Cette fonction permet le traitement simultané des messages. De manière générale, un serveur d'applications crée un pool d'unités d'exécution et la mise en oeuvre JMS met des messages à la disposition de celles-ci. Ce dispositif permet à un

serveur d'applications compatible JMS d'offrir des fonctionnalités avancées de messagerie, telles que les beans de traitement des messages.

Les applications standard ne font pas appel à l'interface ConnectionConsumer, contrairement aux clients JMS experts. Pour ces clients, l'interface ConnectionConsumer fournit une méthode très performante de livraison simultanée de messages à un pool d'unités d'exécution. Lorsqu'un message entrant est destiné à une file d'attente ou à une rubrique, JMS sélectionne une unité d'exécution dans le pool, puis lui livre un lot de messages. Pour ce faire, il exécute une méthode `onMessage()` du programme d'écoute de messages (MessageListener) associé.

Vous pouvez obtenir le même résultat par la construction de plusieurs objets Session et MessageConsumer, chaque objet étant associé à un programme d'écoute MessageListener enregistré. Toutefois, l'interface ConnectionConsumer permet de meilleures performances, utilise moins de ressources et offre une plus grande souplesse d'utilisation. En particulier, elle nécessite moins d'objets Session.

Pour vous aider à développer des applications qui utilisent des objets ConnectionConsumer, MQ JMS fournit un exemple entièrement opérationnel de mise en oeuvre d'un pool d'unités d'exécution. Vous pouvez employer cette mise en oeuvre sans modification, ou l'adapter aux besoins spécifiques de l'application.

Planification d'une application

Principes généraux d'une application de messagerie point à point

Lorsqu'une application crée un client de connexion ConnectionConsumer à partir d'un objet QueueConnection, elle indique un objet JMS Queue et un sélecteur de messages de type chaîne. Le client ConnectionConsumer reçoit ensuite des messages (ou plus précisément, fournit des messages à des sessions du pool de sessions serveur associé). Les messages arrivent en file d'attente, puis s'ils correspondent au sélecteur, sont livrés aux sessions du pool de sessions serveur associé.

Pour MQSeries, l'objet Queue se rapporte à une file QLOCAL ou QALIAS du gestionnaire de files d'attente local. S'il s'agit d'une file QALIAS, celle-ci doit désigner une file QLOCAL. La file QLOCAL MQSeries entièrement résolue est connue en tant que file *QLOCAL sous-jacente*. Un client ConnectionConsumer est dit *actif* s'il n'est pas fermé et que son objet QueueConnection parent est lancé.

Plusieurs clients ConnectionConsumer, chacun doté de sélecteurs différents, peuvent s'exécuter pour la même file QLOCAL sous-jacente. Pour le maintien des performances, les messages inutiles ne doivent pas s'accumuler dans la file d'attente. Il s'agit des messages pour lesquels il n'existe aucune correspondance entre un client ConnectionConsumer actif et un sélecteur. Vous pouvez définir la fabrique QueueConnectionFactory pour retirer ces messages inutiles de la file d'attente (pour plus d'informations, reportez-vous à la section «Retrait des messages de la file d'attente» à la page 221). Vous pouvez configurer ce comportement de l'une des deux manières suivantes :

- A l'aide de l'outil d'administration JMS, attribuez la valeur MRET(NO) à la fabrique QueueConnectionFactory.
- Dans votre programme, utilisez :

  ```
  MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
  ```

Si vous ne modifiez pas cette configuration, le mode par défaut consiste à conserver dans la file d'attente les messages inutiles.

Des clients ConnectionConsumers ciblant la même file QLOCAL sous-jacente peuvent être créés à partir de plusieurs objets QueueConnection. Toutefois, pour des raisons de performances, il est préférable que plusieurs machines virtuelles Java (JVM) ne créent pas de client ConnectionConsumer pour la même file d'attente QLOCAL.

Lorsque vous configurez le gestionnaire de files d'attente MQSeries, tenez compte des points suivants :

- La file QLOCAL sous-jacente doit être activée pour les entrées partagées. Pour ce faire, utilisez la commande MQSC :

  `ALTER QLOCAL(`*`nom.qlocal.perso`*`) SHARE GET(ENABLED)`
- Votre gestionnaire de files d'attente doit comporter une file d'attente de rebut activée. En cas d'incident au niveau d'un client ConnectionConsumer lorsque celui-ci envoie un message en file d'attente de rebut, la livraison de messages de la file QLOCAL sous-jacente s'arrête. Pour définir une file d'attente de rebut, utilisez :

  `ALTER QMGR DEADQ(`*`nom.file.attente.rebut.perso`*`)`
- L'utilisateur qui exécute le client ConnectionConsumer doit détenir le droit d'exécuter MQOPEN avec MQOO_SAVE_ALL_CONTEXT et MQOO_PASS_ALL_CONTEXT. Pour plus d'informations, reportez-vous à la documentation MQSeries adaptée à la plateforme utilisée.
- La conservation des messages inutiles en file d'attente altère les performances du système. Par conséquent, planifiez les sélecteurs de messages pour que les clients ConnectionConsumers retirent tous les messages de la file d'attente.

Pour plus d'informations sur les commandes MQSC, reportez-vous au manuel *MQSeries - Guide de référence des commandes MQSC*.

Principes généraux d'une application de messagerie de publication/souscription (publish/subscribe)

Lorsqu'une application crée un client de connexion ConnectionConsumer à partir d'un objet TopicConnection, elle indique un objet Topic et un sélecteur de messages de type chaîne. Le client ConnectionConsumer reçoit ensuite dans cette rubrique (Topic) des messages qui correspondent au sélecteur.

Une application peut également créer un client ConnectionConsumer durable associé à un nom déterminé. Le client ConnectionConsumer durable reçoit des messages qui ont été publiés dans la rubrique depuis sa dernière activation. Les messages reçus dans la rubrique correspondent au sélecteur.

Dans le cas des souscriptions non durables, une file d'attente distincte est utilisée pour les souscriptions du client ConnectionConsumer. La propriété CCSUB (paramétrable) de la fabrique TopicConnectionFactory précise la file d'attente à utiliser. En général, la propriété CCSUB indique une seule file d'attente pour tous les clients ConnectionConsumer qui utilisent la même fabrique TopicConnectionFactory. Toutefois, chaque client ConnectionConsumer peut générer une file d'attente temporaire par l'indication d'un préfixe de nom de file suivi d'un astérisque '*'.

Dans le cas des souscriptions durables, la propriété CCDSUB de la rubrique précise la file d'attente à utiliser. A nouveau, il peut s'agir d'une file d'attente qui existe déjà ou d'un préfixe de nom de file suivi d'un astérisque '*'. Si vous indiquez une

file d'attente qui existe déjà, tous les clients ConnectionConsumer durables abonnés à la rubrique l'utilisent. Si vous entrez un préfixe de nom de file suivi d'un astérisque '*', une file d'attente est générée lorsqu'un premier client ConnectionConsumer durable est créé sous un nom donné. Cette file d'attente est réutilisée ultérieurement lorsqu'un autre client ConnectionConsumer durable est créé sous le même nom.

Lorsque vous configurez le gestionnaire de files d'attente MQSeries, tenez compte des points suivants :

- Votre gestionnaire de files d'attente doit comporter une file d'attente de rebut activée. En cas d'incident au niveau d'un client ConnectionConsumer lorsque celui-ci envoie un message en file d'attente de rebut, la livraison de messages de la file QLOCAL sous-jacente s'arrête. Pour définir une file d'attente de rebut, utilisez :

  `ALTER QMGR DEADQ(`*`nom.file.attente.rebut.perso`*`)`
- L'utilisateur qui exécute le client ConnectionConsumer doit détenir le droit d'exécuter MQOPEN avec MQOO_SAVE_ALL_CONTEXT et MQOO_PASS_ALL_CONTEXT. Pour plus d'informations, reportez-vous à la documentation MQSeries adaptée à la plateforme utilisée.
- Vous pouvez optimiser les performances individuelles d'un client ConnectionConsumer par la création d'une file d'attente distincte dédiée à celui-ci. Toutefois, cette option est plus coûteuse en ressources.

Traitement des messages nocifs

Un message mal formaté peut arriver dans une file d'attente. Un message de ce type peut provoquer l'échec de l'application destinataire et annuler la réception du message. Dans ce cas, le message peut être indéfiniment reçu puis renvoyé à la file d'attente. Il s'agit d'un *message nocif*. Le client ConnectionConsumer doit être capable de détecter les messages nocifs, puis de les rediriger vers une destination de remplacement.

Lorsqu'une application utilise des clients ConnectionConsumer, les conditions d'annulation d'un message dépendent de la session fournie par le serveur d'applications :

- Dans le cas d'une session non transactionnelle avec AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, un message n'est annulé que par une erreur système ou l'arrêt inattendu de l'application.
- Dans le cas d'une session non transactionnelle avec CLIENT_ACKNOWLEDGE, les messages non acquittés par un accusé de réception peuvent être annulés par le serveur d'applications à l'aide de la méthode `Session.recover()`.

  En général, l'implémentation client de MessageListener ou le serveur d'applications, via les appels `Message.acknowledge()`. `Message.acknowledge()`, accuse réception de tous les messages livrés à la session jusqu'ici.
- Dans le cas d'une session transactionnelle, le serveur d'applications valide (COMMIT) la session. S'il détecte une erreur, il peut annuler un ou plusieurs messages.
- Si le serveur d'applications fournit un objet XASession, les messages sont validés ou annulés en fonction d'une transaction répartie. Le serveur d'applications se charge de terminer l'exécution de la transaction.

Le gestionnaire de files d'attente MQSeries relève le nombre d'annulations dont chaque message fait l'objet. Lorsque ce nombre atteint un seuil pouvant être configuré, le client ConnectionConsumer redirige le message vers une file d'annulation désignée. En cas d'échec, le message est retiré de la file, puis envoyé

en file d'attente de rebut ou supprimé. Pour plus d'informations, reportez-vous à la section «Retrait des messages de la file d'attente».

Sur la plupart des plateformes, le seuil et la file de remise en attente sont des propriétés de la file QLOCAL de MQSeries. Dans le cas d'une application de messagerie point à point, il doit s'agir de la file QLOCAL sous-jacente. Dans le cas d'une application de messagerie de publication/souscription, il s'agit de la file CCSUB paramétrée dans la fabrique TopicConnectionFactory, ou de la file CCDSUB configurée dans la rubrique Topic. Pour définir les propriétés liées au seuil et à la file de remise en attente, lancez la commande MQSC suivante :

```
ALTER QLOCAL(nom.file.attente)
BOTHRESH(seuil)
BOQUEUE(nom.file.remise.en.attente)
```

Dans le cas d'une application de messagerie de publication/souscription, lorsque le système crée une file d'attente dynamique par souscription, ces paramètres sont dérivés de la file d'attente modèle MQ JMS. Pour les modifier, vous pouvez utiliser :

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE)
BOTHRESH(seuil)
BOQUEUE(nom.file.remise.en.attente)
```

Lorsque le seuil est égal à zéro, le traitement des messages nocifs est désactivé et ceux-ci restent dans la file d'entrée. Sinon, lorsque le nombre d'annulations atteint le seuil défini, le message est envoyé à la file de remise en attente désignée. Lorsque le nombre d'annulations atteint le seuil, mais que le message ne peut pas être envoyé à la file de remise en attente, le message est placé en file d'attente de rebut ou supprimé. Ce cas se produit lorsque la file de remise en attente n'est pas définie, ou que le client ConnectionConsumer ne peut pas envoyer le message dans la file de remise en attente. Sur certaines plateformes, vous ne pouvez pas préciser les propriétés liées au seuil et à la file de remise en attente. Sur ces plateformes, les messages sont envoyés en file d'attente de rebut ou supprimés après 20 annulations. Pour plus d'informations, reportez-vous à la section «Retrait des messages de la file d'attente».

Retrait des messages de la file d'attente

Lorsqu'une application utilise des clients ConnectionConsumer, JMS peut avoir besoin de retirer des messages de la file d'attente dans plusieurs cas :

Message mal formaté
: Message entrant que JMS ne peut pas analyser.

Message nocif
: Le message peut atteindre le seuil d'annulation, mais son envoi en file d'annulation par le client ConnectionConsumer échoue.

Aucun client ConnectionConsumer intéressé
: Dans le cas d'une application de messagerie point à point, lorsque la fabrique QueueConnectionFactory est configurée de manière à ne pas conserver les messages inutiles, le message entrant n'est utile à aucun client ConnectionConsumer.

Dans ces cas, le client ConnectionConsumer tente de retirer le message de la file d'attente. Les options de destination figurant dans la zone de rapport du descripteur MQMD associé au message définissent le comportement exact.

Ces options sont les suivantes :

MQRO_DEAD_LETTER_Q
Le message est envoyé dans la file d'attente de rebut du gestionnaire de files d'attente. C'est le mode par défaut.

MQRO_DISCARD_MSG
Le message est supprimé.

Le client ConnectionConsumer génère également un message de rapport, en fonction de la zone de rapport du descripteur MQMD associé au message. Ce message est envoyé à la file ReplyToQ correspondante du gestionnaire de files d'attente ReplyToQmgr. En cas d'erreur lors de l'envoi du message de rapport, le message est alors placé en file d'attente de rebut. Dans la zone de rapport du descripteur MQMD associé au message, les options de rapport d'exception définissent les détails relatifs au message de rapport. Ces options sont les suivantes :

MQRO_EXCEPTION
Génération d'un message de rapport contenant le descripteur MQMD du message d'origine. Ce message ne comporte aucune donnée de corps de message.

MQRO_EXCEPTION_WITH_DATA
Génération d'un message de rapport contenant le descripteur MQMD, des en-têtes MQ et 100 octets de données de corps de message.

MQRO_EXCEPTION_WITH_FULL_DATA
Génération d'un message de rapport contenant toutes les données du message d'origine.

default
Aucun message de rapport n'est généré.

Lorsque des messages de rapport sont générés, les options suivantes sont en vigueur :

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

Lorsqu'un client ConnectionConsumer ne peut pas suivre les options de destination ou les options de rapport d'exception qui figurent dans le descripteur MQMD du message, son action dépend de la persistance de ce message. Si le message n'est pas persistant, il est supprimé et aucun message de rapport n'est généré. Si le message est persistant, la livraison de tous les messages à partir de la file QLOCAL s'arrête.

Par conséquent, il est important de définir une file d'attente de rebut et de la vérifier régulièrement pour s'assurer qu'aucun incident ne se produit. En particulier, vérifiez que la file d'attente de rebut n'atteint pas sa longueur maximale et que la taille maximale de message convient pour tous les messages.

Lorsqu'un message est redirigié vers la file d'attente de rebut, il est précédé d'un en-tête de non-distribution MQSeries (MQDLH). Pour plus d'informations sur le format de l'en-tête MQDLH, reportez-vous au manuel *MQSeries Application Programming Reference*. Les zones suivantes permettent d'identifier les messages

qu'un client ConnectionConsumer a placés en file d'attente de rebut, ou les messages de rapport qu'un client ConnectionConsumer a générés :

- PutApplType a pour valeur `MQAT_JAVA (0x1C)`,
- PutApplName a pour valeur "`MQ JMS ConnectionConsumer`".

Ces zones figurent dans l'en-tête MQDLH des messages placés en file d'attente de rebut et dans le descripteur MQMD des messages de rapport. La zone Feedback (code retour) du descripteur MQMD et la zone Reason (code raison) de l'en-tête MQDLH contiennent un code qui décrit l'erreur. Pour plus d'informations sur ces codes, reportez-vous à la section «Traitement des erreurs». Les autres zones sont traitées dans le manuel *MQSeries Application Programming Reference*.

Traitement des erreurs

Reprise après erreur

En cas d'erreur importante au niveau du client ConnectionConsumer, la livraison de messages vers tous les clients ConnectionConsumers concernés par la même file QLOCAL s'arrête. C'est généralement le cas si le client ConnectionConsumer ne peut pas envoyer de message dans la file d'attente de rebut, ou qu'une erreur se produit lorsqu'il lit des messages à partir de la file QLOCAL.

Dans ce cas, l'application et le serveur d'applications sont avertis de la manière suivante :

- Une notification est adressée à tout programme d'écoute ExceptionListener connecté à l'objet Connection concerné.

Vous pouvez tenter d'identifier la cause de l'incident. L'administrateur système doit parfois intervenir pour le résoudre.

Deux méthodes permettent à une application d'effectuer une reprise après erreur :

- Appel de la méthode `close()` pour tous les clients ConnectionConsumers concernés. L'application ne peut créer des clients ConnectionConsumers qu'après la fermeture de tous les clients ConnectionConsumers concernés et la résolution des incidents système.
- Appel de la méthode `stop()` pour toutes les connexions concernées. Une fois les connexions arrêtées et les incidents système résolus, l'appel de la méthode `start()` par l'application pour toutes les connexions doit aboutir.

Codes raison et codes retour

Vous pouvez déterminer la cause d'une erreur à l'aide des informations suivantes :

- le code retour figurant dans tout message de rapport,
- le code raison apparaissant dans l'en-tête MQDLH de tout message placé en file d'attente de rebut.

Les clients ConnectionConsumer génèrent les codes raison suivants :

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Cause Le message atteint le seuil d'annulation paramétré dans la file QLOCAL, mais aucune file d'annulation n'est définie.

Sur les plateformes où vous ne pouvez pas configurer de file d'annulation, le message atteint le seuil fixé par JMS, c'est-à-dire 20 annulations.

Action Pour éviter cet incident, assurez-vous que les clients ConnectionConsumer qui utilisent la file d'attente comportent un jeu de sélecteurs traitant tous les messages, ou configurez la fabrique QueueConnectionFactory pour conserver les messages.

Vous pouvez également rechercher l'origine du message.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Cause Dans une application de messagerie point à point, un message ne correspond à aucun des sélecteurs associés aux clients ConnectionConsumer pour le contrôle de la file d'attente. Pour le maintien des performances, le message est redirigé vers la file d'attente de rebut.

Action Pour éviter cet incident, assurez-vous que les clients ConnectionConsumer qui utilisent la file d'attente comportent un jeu de sélecteurs traitant tous les messages, ou configurez la fabrique QueueConnectionFactory pour conserver les messages.

Vous pouvez également rechercher l'origine du message.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Cause JMS ne peut pas interpréter le message placé en file d'attente.

Action Recherchez l'origine du message. JMS fournit généralement des messages de format inattendu (par exemple, BytesMessage ou TextMessage). Un échec se produit parfois lorsque le message est très mal formaté.

Les autres codes apparaissent dans ces zones lorsqu'une tentative d'envoi d'un message en file d'annulation échoue. Dans ce cas, le code décrit la raison de l'échec de la remise en file d'attente. Pour établir un diagnostic sur la cause des erreurs de ce type, reportez-vous au manuel *MQSeries Application Programming Reference*.

Si le message de rapport ne peut pas être placé en file ReplyToQ, il est mis en file d'attente de rebut. Dans ce cas, la zone Feedback (code retour) du descripteur MQMD est remplie (reportez-vous aux descriptions précédentes). La zone Reason (code raison) de l'en-tête MQDLH explique la raison pour laquelle le message n'a pas été placé en file ReplyToQ.

Code exemple de serveur d'applications

La figure 6 à la page 225 constitue un récapitulatif des principes mis en oeuvre dans les fonctionnalités ServerSessionPool (pool de sessions serveur) et ServerSession (session serveur).

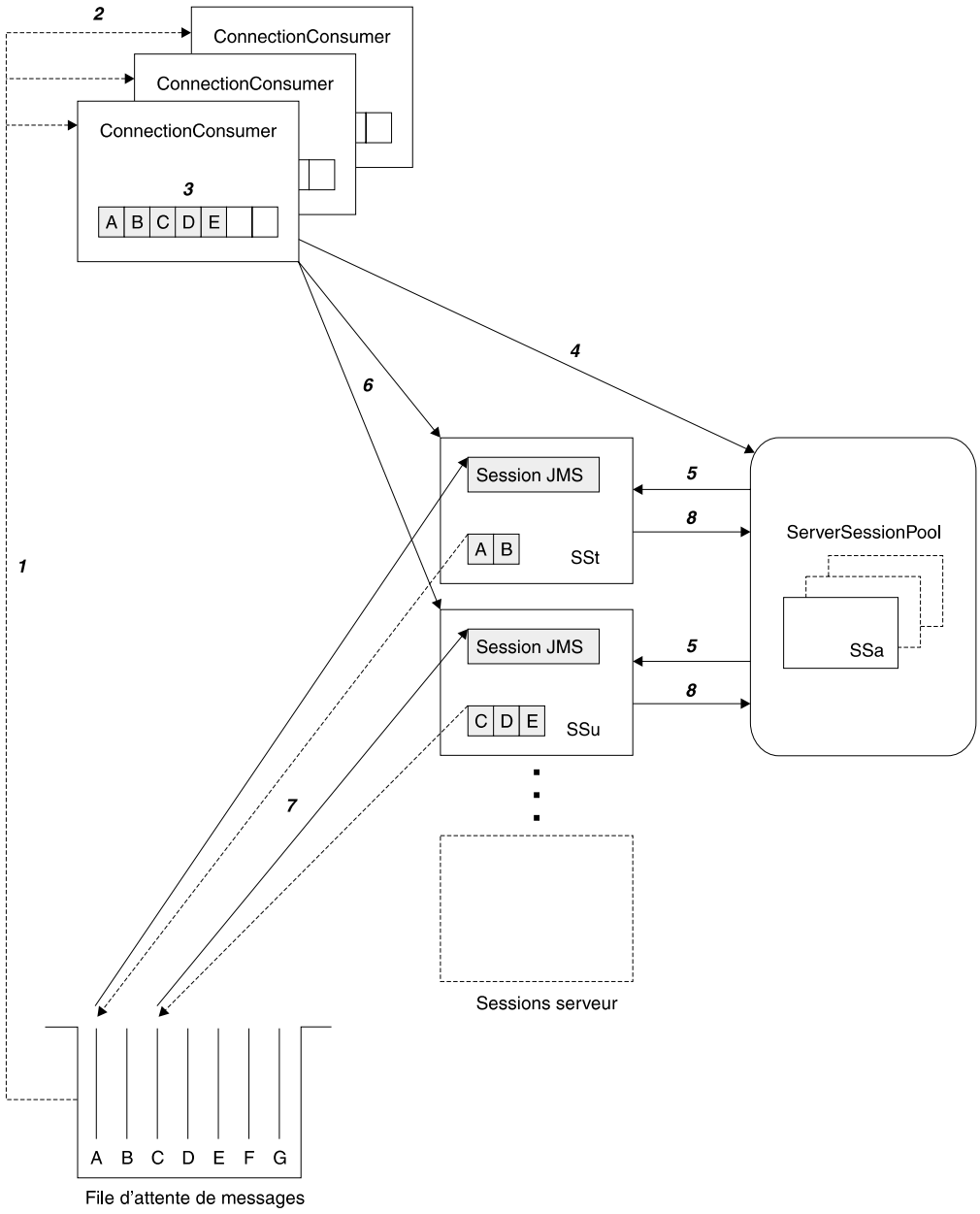


Figure 6. Fonctionnalités ServerSessionPool et ServerSession

1. Les clients de connexion ConnectionConsumers extraient de la file d'attente des références de message.
2. Chaque client ConnectionConsumer sélectionne certaines références de message.
3. La mémoire tampon ConnectionConsumer conserve les références de message sélectionnées.
4. Le client ConnectionConsumer demande un ou plusieurs objets ServerSessions de l'objet ServerSessionPool.
5. Les objets ServerSessions sont attribués à partir de l'objet ServerSessionPool.
6. Le client ConnectionConsumer affecte des références de message aux objets ServerSession et démarre les unités d'exécution correspondantes.

7. Chaque objet ServerSession extrait de la file d'attente ses messages référencés. Il les transmet à la méthode `onMessage` à partir du programme d'écoute MessageListener associé à l'objet Session JMS.
8. Une fois le traitement terminé, l'objet ServerSession est renvoyé dans le pool.

Le serveur d'applications assure les fonctionnalités ServerSessionPool et ServerSession. Toutefois, MQ JMS est fourni avec une mise en oeuvre simple de ces interfaces, avec le source du programme. Ces modèles se trouvent dans le répertoire suivant, où <rép_install> désigne le répertoire d'installation de MQ JMS:

```
<install_dir>/samples/jms/asf
```

Ces modèles permettent d'utiliser les fonctions ASF MQ JMS dans un environnement autonome (c'est-à-dire que vous n'avez pas besoin de serveur d'applications spécifique). Ils fournissent également des exemples de mise en oeuvre de ces interfaces et tirent parti des fonctions ASF MQ JMS. Ces exemples constituent une aide pour les utilisateurs MQ JMS et pour les fournisseurs d'autres serveurs d'applications.

MyServerSession.java

Cette classe met en oeuvre l'interface `javax.jms.ServerSession`. Sa fonction de base consiste à associer une unité d'exécution à une session JMS. Les instances de cette classe sont regroupées dans un pool de sessions serveur (objet ServerSessionPool) (reportez-vous à la section «MyServerSessionPool.java»). En tant que session serveur (objet ServerSession), elle doit mettre en oeuvre les deux méthodes suivantes :

- `getSession()`, qui renvoie la session JMS associée à cette session serveur,
- `start()`, qui démarre l'unité d'exécution de cette session serveur et a pour résultat l'appel de la méthode `run()` de la session JMS.

La classe MyServerSession met également en oeuvre l'interface `Runnable`. Par conséquent, la création de l'unité d'exécution de la session serveur peut découler de cette classe, sans qu'une classe distincte soit nécessaire.

La classe utilise un dispositif `wait()-notify()` basé sur les valeurs de deux indicateurs booléens, ready et quit. Ce dispositif signifie que la session serveur crée et démarre l'unité d'exécution associée en phase de construction. Cependant, elle n'exécute pas automatiquement le corps de la méthode `run()`. Le corps de la méthode `run()` n'est exécuté que lorsque la méthode `start()` attribue la valeur true à l'indicateur ready. L'interface ASF appelle la méthode `start()` lorsque des messages doivent être livrés à la session JMS associée.

La livraison fait appel à la méthode `run()` de la session JMS. L'interface ASF MQ JMS a déjà chargé des messages dans la méthode `run()`.

Une fois la livraison terminée, l'indicateur ready reprend la valeur false et le pool de sessions serveur propriétaire reçoit une notification de livraison. La session serveur reste en attente jusqu'à ce que la méthode `start()` soit appelée à nouveau, ou que la méthode `close()` soit sollicitée pour arrêter l'unité d'exécution associée.

MyServerSessionPool.java

Cette classe met en oeuvre l'interface `javax.jms.ServerSessionPool`. Elle a pour but de créer et de contrôler les accès à un pool de sessions serveur.

Dans cette mise en oeuvre simple, le pool est composé d'un tableau statique d'objets ServerSession créés pendant la construction du pool. Les quatre paramètres suivants sont transmis au constructeur :

- `javax.jms.Connection` *`connection`*

  Connexion utilisée pour la création des sessions JMS.

- `int` *`capacity`*

  Taille du tableau d'objets MyServerSession.

- `int` *`ackMode`*

  Mode d'accusé de réception obligatoire pour les sessions JMS.

- `MessageListenerFactory` *`mlf`*

  Fabrique MesssageListenerFactory chargée de créer le programme d'écoute de messages associé aux sessions JMS. Voir «MessageListenerFactory.java».

Avec ces paramètres, le constructeur du pool crée un tableau d'objets MyServerSession. La connexion fournie permet de créer des sessions JMS utilisant le mode d'accusé de réception défini et le domaine approprié (QueueSessions pour les applications de messagerie point à point et TopicSessions pour les applications de messagerie de publication/souscription). Les sessions sont fournies avec un programme d'écoute de messages. Enfin, les objets ServerSession basés sur les sessions JMS sont créés.

Cet exemple de mise en oeuvre constitue un modèle statique. Tous les objets ServerSessions du pool sont créés en même temps que le pool qui, par la suite, ne peut plus être agrandi ni réduit. Cette méthode est la plus simple. Un pool de sessions serveur peut, à l'aide d'un algorithme complexe, créer des objets ServerSessions en mode dynamique, selon les besoins.

La classe MyServerSessionPool effectue le suivi des objets ServerSessions en cours d'utilisation via la gestion d'un tableau de valeurs booléennes inUse. Ces valeurs booléennes sont initialisées sur la valeur false. Lorsque la méthode `getServerSession` est appelée et qu'elle demande une session serveur du pool, la première valeur false est recherchée dans le tableau inUse. Lorsque la recherche aboutit, la valeur booléenne prend la valeur true et la session serveur correspondante est renvoyée. Si aucune valeur false n'est détectée dans le tableau inUse, la méthode `getServerSession` doit attendre (`wait()`) jusqu'à la réception d'une notification.

La notification est envoyée dans les cas suivants :

- La méthode `close()` du pool est appelée, indiquant que le pool doit être fermé.
- Une session serveur en cours termine sa charge de travail et appelle la méthode `serverSessionFinished`. La méthode `serverSessionFinished` renvoie la session serveur au pool et attribue la valeur false à l'indicateur inUse correspondant. La session serveur est ensuite réutilisable.

MessageListenerFactory.java

Dans cet exemple, un objet MessageListenerFactory est associé à chaque instance ServerSessionPool. La classe MessageListenerFactory représente une interface très simple qui permet d'extraire l'instance d'une classe chargée de mettre en oeuvre l'interface `javax.jms.MessageListener`. Elle contient une seule méthode :

```
javax.jms.MessageListener createMessageListener();
```

Code exemple de serveur d'applications

Une mise en oeuvre de cette interface est fournie lors de la construction de l'objet ServerSessionPool. Cet objet permet de créer des programmes d'écoute de messages pour les sessions JMS individuelles qui constituent une copie de secours des sessions serveur du pool. Cette architecture signifie que chaque mise en oeuvre de l'interface `MessageListenerFactory` doit posséder son propre objet ServerSessionPool.

MQ JMS comporte un exemple de mise en oeuvre MessageListenerFactory traité à la section «CountingMessageListenerFactory.java» à la page 229.

Exemples d'utilisation des fonctions ASF

Un jeu de classes accompagnées de leur source se trouve dans le répertoire `<rép_install>/samples/jms/asf` (où <rép_install> désigne le répertoire d'installation de MQ JMS). Ces classes utilisent les fonctions de serveur d'applications (ASF) MQ JMS décrites à la section «Classes et fonctions ASF» à la page 217), dans l'environnement du serveur d'applications autonome présenté à la section «Code exemple de serveur d'applications» à la page 224.

Les exemples fournis illustrent trois cas d'utilisation des fonctions ASF côté application client :

- Un exemple simple d'application de messagerie point à point utilise les classes suivantes :
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- Un exemple plus complexe d'application de messagerie point à point utilise les classes suivantes :
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- Un exemple simple d'application de messagerie de publication/souscription utilise les classes suivantes :
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- Un exemple plus complexe d'application de messagerie de publication/souscription utilise les classes suivantes :
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java

Les sections suivantes décrivent chaque classe.

Load1.java

Cette classe est une application JMS générique simple qui charge un nombre déterminé de messages dans une file d'attente donnée, puis prend fin. Elle peut extraire les objets gérés obligatoires à partir d'un espace annuaire JNDI, ou les créer explicitement, à l'aide des classes MQ JMS de mise en oeuvre. Les objets

gérés obligatoires sont QueueConnectionFactory et Queue. Les options de ligne de commande permettent de définir le nombre de messages à charger dans la file d'attente et le délai de veille entre deux insertions de message.

Pour cette application, deux versions de syntaxe sont possibles depuis la ligne de commande.

Syntaxe avec l'interface JNDI :

```
java Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Syntaxe sans l'interface JNDI :

```
java Load1 -nojndi [-qm qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

Le tableau 22 décrit les arguments et fournit leurs valeurs par défaut.

Tableau 22. Arguments et valeurs par défaut de la classe Load1

Argument	Signification	Valeur par défaut
jndiICF	Classe de fabrique de contexte initial utilisée pour JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Adresse URL du fournisseur utilisée pour JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	Clé de recherche JNDI utilisée pour l'objet QueueConnectionFactory	cn=qcf
qLookup	Clé de recherche JNDI utilisée pour l'objet Queue	cn=q
qMgrName	Gestionnaire de files d'attente auquel vous devez vous connecter	"" (utilisation du gestionnaire de files d'attente par défaut)
qName	Nom de la file d'attente à utiliser	SYSTEM.DEFAULT.LOCAL.QUEUE
sleepTime	Durée de la pause (en millisecondes) entre les insertions de message	0 (aucune pause)
numMsgs	Nombre de messages à insérer	1000

En cas d'erreur, un message d'erreur s'affiche et l'application s'arrête.

Avec cette application, vous pouvez simuler le chargement de messages dans la file d'attente MQSeries. Le chargement de message peut ensuite déclencher les applications compatibles ASF décrites dans les sections ci-après. Les messages placés en file d'attente sont des objets TextMessage JMS simples. Ces objets ne contiennent aucune propriété de message définie par l'utilisateur, alors que ce type de propriété peut être utile pour faire appel à divers programmes d'écoute de messages. Le code source est fourni pour vous permettre de modifier si nécessaire l'application de chargement.

CountingMessageListenerFactory.java

Ce fichier contient la définition des deux classes suivantes :

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener constitue une mise en oeuvre très simple de l'interface `javax.jms.MessageListener`. Elle enregistre le nombre d'appels de la méthode `onMessage` associée, mais n'a aucune incidence sur les messages transmis.

CountingMessageListenerFactory constitue la classe de fabrique de CountingMessageListener. C'est une mise en oeuvre de l'interface `MessageListenerFactory` décrite à la section «MessageListenerFactory.java» à la page 227. Cette fabrique enregistre tous les programmes d'écoute de messages qu'elle génère. Elle contient également la méthode `printStats()` qui affiche les statistiques d'utilisation de chaque programme d'écoute.

ASFClient1.java

Cette application sert de client pour l'ASF MQ JMS. Elle définit un seul objet ConnectionConsumer pour l'utilisation des messages dans une file d'attente MQSeries unique. Elle affiche des statistiques de débit par programme d'écoute utilisé et s'arrête après un délai d'une minute.

L'application peut extraire les objets gérés obligatoires à partir d'un espace annuaire JNDI, ou les créer explicitement, à l'aide des classes MQ JMS de mise en oeuvre. Les objets gérés obligatoires sont QueueConnectionFactory et Queue.

Pour cette application, deux versions de syntaxe sont possibles à partir de la ligne de commande.

Syntaxe avec l'interface JNDI :

```
java ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

Syntaxe sans l'interface JNDI :

```
java ASFClient1 -nojndi [-qm qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

Le tableau 23 décrit les arguments et fournit leurs valeurs par défaut.

Tableau 23. Arguments et valeurs par défaut de la classe ASFClient1

Argument	Signification	Valeur par défaut
jndiICF	Classe de fabrique de contexte initial utilisée pour JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Adresse URL du fournisseur utilisée pour JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	Clé de recherche JNDI utilisée pour l'objet QueueConnectionFactory	cn=qcf
qLookup	Clé de recherche JNDI utilisée pour l'objet Queue	cn=q
qMgrName	Gestionnaire de files d'attente auquel vous devez vous connecter	"" (utilisation du gestionnaire de files d'attente par défaut)
qName	Nom de la file d'attente à utiliser	SYSTEM.DEFAULT.LOCAL.QUEUE
poolSize	Nombre d'objets ServerSession créés dans l'objet ServerSessionPool utilisé	5

Tableau 23. Arguments et valeurs par défaut de la classe ASFClient1 (suite)

Argument	Signification	Valeur par défaut
batchSize	Nombre maximal de messages pouvant être simultanément attribués à un objet ServerSession	10

L'application obtient un objet QueueConnection de la fabrique QueueConnectionFactory.

Un objet ServerSessionPool est construit sous forme d'objet MyServerSessionPool à l'aide des éléments suivants :

- l'objet QueueConnection créé précédemment,
- la taille de pool requise (paramètre poolSize),
- le mode d'accusé de réception AUTO_ACKNOWLEDGE,
- une instance de la fabrique CountingMessageListenerFactory (reportez-vous à la section «CountingMessageListenerFactory.java» à la page 229).

La méthode `createConnectionConsumer` associée à la connexion est ensuite appelée, avec la transmission des éléments suivants :

- l'objet Queue (file d'attente) obtenu précédemment,
- un sélecteur de messages ayant pour valeur NULL (indiquant que tous les messages doivent être acceptés),
- l'objet ServerSessionPool qui vient d'être créé,
- la taille de lot requise (paramètre batchSize).

La consommation des messages est ensuite lancée par l'appel de la méthode `start()` associée à la connexion.

L'application client affiche toutes les 10 secondes des statistiques de débit par programme d'écoute de messages utilisé. Après un délai d'une minute, la connexion est fermée, le pool de sessions serveur est arrêté et l'application prend fin.

Load2.java

Cette classe est une application JMS qui charge un nombre déterminé de messages dans une file d'attente donnée, puis prend fin, comme la classe Load1.java. La syntaxe possible depuis la ligne de commande est également similaire à celle de la classe Load1.java (remplacez Load1 par Load2 dans la syntaxe). Pour plus d'informations, reportez-vous à la section «Load1.java» à la page 228.

La différence entre les deux classes est que chaque message contient une propriété utilisateur appelée value, qui prend une valeur entière aléatoire comprise entre 0 et 100. Cette propriété signifie que vous pouvez appliquer des sélecteurs aux messages. Par conséquent, les messages peuvent être partagés entre les deux objets ConnectionConsumer créés dans l'application client décrite à la section «ASFClient2.java» à la page 232.

LoggingMessageListenerFactory.java

Ce fichier contient la définition des deux classes suivantes :

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener constitue une mise en oeuvre de l'interface `javax.jms.MessageListener`. Elle prend les messages qui lui sont transmis et écrit une entrée dans le fichier journal. Le fichier journal par défaut est `./ASFClient2.log`. Dans le fichier journal, vous pouvez rechercher les messages envoyés à l'objet ConnectionConsumer qui utilise le programme d'écoute de messages indiqué.

LoggingMessageListenerFactory représente la classe de fabrique de LoggingMessageListener. C'est une mise en oeuvre de l'interface `MessageListenerFactory` décrite à la section «MessageListenerFactory.java» à la page 227.

ASFClient2.java

ASFClient2.java est une application client sensiblement plus complexe que l'application ASFClient1.java. Elle crée deux clients de connexion (objets ConnectionConsumer) qui alimentent la même file d'attente, mais appliquent des sélecteurs de messages différents. L'application utilise la fabrique CountingMessageListenerFactory pour l'un des deux objets ConnectionConsumer et la fabrique LoggingMessageListenerFactory, pour l'autre. L'utilisation de deux fabriques de programmes d'écoute de messages signifie que chaque objet ConnectionConsumer doit posséder son propre pool de sessions serveur.

L'application affiche à l'écran les statistiques relatives à l'un des deux objets ConnectionConsumer et consigne celles relatives à l'autre objet ConnectionConsumer dans un fichier journal.

La syntaxe possible depuis la ligne de commande est similaire à celle de la classe «ASFClient1.java» à la page 230 (remplacez ASFClient1 par ASFClient2 dans la syntaxe). Chaque pool de sessions serveur contient le nombre d'objets ServerSession défini par le paramètre poolSize (taille de pool).

Normalement, la répartition des messages est irrégulière. Les messages chargés dans la file d'attente source par Load2 contiennent une propriété utilisateur, dont la valeur doit être comprise entre 0 et 100, répartie de manière régulière et aléatoire. Les sélecteurs de messages `value>75` et `value≤75` sont appliqués respectivement à l'objet highConnectionConsumer et à l'objet normalConnectionConsumer. Les messages de l'objet highConnectionConsumer (environ 25 % de la charge totale) sont envoyés au programme d'écoute LoggingMessageListener. Les messages de l'objet normalConnectionConsumer (environ 75 % de la charge totale) sont envoyés au programme d'écoute CountingMessageListener.

Lors de l'exécution de l'application client, les statistiques relatives à l'objet normalConnectionConsumer et aux fabriques CountingMessageListenerFactory associées sont affichées à l'écran toutes les 10 secondes. Les statistiques relatives à l'objet highConnectionConsumer et aux fabriques LoggingMessageListenerFactory associées sont consignées dans le fichier journal.

Vous pouvez rechercher à l'écran et dans le fichier journal la destination réelle des messages. Additionnez les totaux de chaque programme d'écoute CountingMessageListener. Si l'application client ne s'arrête pas avant la consommation de tous les messages, cette somme doit représenter environ 75 % de la charge. Le nombre d'entrées de fichier journal doit constituer le reste de la charge. (Si l'application client s'arrête avant la consommation de tous les messages, vous pouvez augmenter le délai d'expiration de l'application.)

TopicLoad.java

Cette classe est une application JMS représentant la version publish/subscribe (publication/souscription) du chargeur de file d'attente Load2 décrit à la section «Load2.java» à la page 231. Elle publie le nombre requis de messages sous la rubrique indiquée, puis prend fin. Chaque message contient une propriété utilisateur appelée value, qui prend une valeur entière aléatoire comprise entre 0 et 100.

Pour pouvoir utiliser cette application, assurez-vous que le courtier est en service et que la configuration requise a été effectuée. Pour plus d'informations, reportez-vous à la section «Configuration supplémentaire pour le mode Publication/Souscription» à la page 22.

Pour cette application, deux versions de syntaxe sont possibles depuis la ligne de commande.

Syntaxe avec l'interface JNDI :

```
java TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

Syntaxe sans l'interface JNDI :

```
java TopicLoad -nojndi [-qm qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

Le tableau 24 décrit les arguments et fournit leurs valeurs par défaut.

Tableau 24. Arguments et valeurs par défaut de la classe TopicLoad

Argument	Signification	Valeur par défaut
jndiICF	Classe de fabrique de contexte initial utilisée pour JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Adresse URL du fournisseur utilisée pour JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	Clé de recherche JNDI utilisée pour l'objet TopicConnectionFactory	cn=tcf
tLookup	Clé de recherche JNDI utilisée pour l'objet Topic	cn=t
qMgrName	Gestionnaire de files d'attente auquel vous devez vous connecter et gestionnaire de files d'attente courtier à utiliser pour la publication des messages	"" (utilisation du gestionnaire de files d'attente par défaut)
tName	Nom de la rubrique cible pour la publication	MQJMS/ASF/TopicLoad
sleepTime	Durée de la pause (en millisecondes) entre les insertions de message	0 (aucune pause)
numMsgs	Nombre de messages à insérer	200

En cas d'erreur, un message d'erreur s'affiche et l'application s'arrête.

ASFClient3.java

ASFClient3.java est une application client représentant la version publish/subscribe (publication/souscription) de la classe «ASFClient1.java» à la page 230. Elle définit un seul objet ConnectionConsumer pour l'utilisation des messages publiés dans une rubrique (Topic) unique. Elle affiche des statistiques de débit par programme d'écoute utilisé et s'arrête après un délai d'une minute.

Pour cette application, deux versions de syntaxe sont possibles depuis la ligne de commande.

Syntaxe avec l'interface JNDI :

```
java ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

Syntaxe sans l'interface JNDI :

```
java ASFClient3 -nojndi [-qm qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

Le tableau 25 décrit les arguments et fournit leurs valeurs par défaut.

Tableau 25. Arguments et valeurs par défaut de la classe ASFClient3

Argument	Signification	Valeur par défaut
jndiICF	Classe de fabrique de contexte initial utilisée pour JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Adresse URL du fournisseur utilisée pour JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	Clé de recherche JNDI utilisée pour l'objet TopicConnectionFactory	cn=tcf
tLookup	Clé de recherche JNDI utilisée pour l'objet Topic	cn=t
qMgrName	Gestionnaire de files d'attente auquel vous devez vous connecter et gestionnaire de files d'attente courtier à utiliser pour la publication des messages	"" (utilisation du gestionnaire de files d'attente par défaut)
tName	Nom de la rubrique à utiliser	MQJMS/ASF/TopicLoad
poolSize	Nombre d'objets ServerSession créés dans l'objet ServerSessionPool utilisé	5
batchSize	Nombre maximal de messages pouvant être simultanément attribués à un objet ServerSession	10

A l'instar de la classe ASFClient1, l'application client affiche toutes les 10 secondes des statistiques de débit par programme d'écoute de messages utilisé. Après un délai d'une minute, la connexion est fermée, le pool de sessions serveur est arrêté et l'application prend fin.

ASFClient4.java

ASFClient4.java est une application client de publication/souscription plus complexe. Elle crée trois objets ConnectionConsumer qui utilisent la même rubrique, mais appliquent des sélecteurs de messages différents.

Les deux premiers objets ConnectionConsumer utilisent les sélecteurs de messages 'high' et 'normal', comme l'application «ASFClient2.java» à la page 232. Le troisième objet ConnectionConsumer n'utilise aucun sélecteur de messages. L'application fait appel à deux fabriques CountingMessageListenerFactory pour les deux objets ConnectionConsumer avec sélecteurs de messages et une fabrique LoggingMessageListenerFactory pour le troisième objet ConnectionConsumer. L'application recourt à des fabriques de programmes d'écoute de messages différentes et, par conséquent, chaque objet ConnectionConsumer doit posséder son propre pool de sessions serveur.

L'application affiche à l'écran les statistiques relatives aux deux objets ConnectionConsumer avec sélecteurs de messages. Elle consigne les statistiques relatives au troisième objet ConnectionConsumer dans un fichier journal.

La syntaxe possible à partir de la ligne de commande est similaire à celle de la classe «ASFClient3.java» à la page 234 (remplacez ASFClient3 par ASFClient3 dans la syntaxe). Chacun des trois pools de sessions serveur contient le nombre d'objets ServerSession défini par le paramètre poolSize (taille de pool).

Lors de l'exécution de l'application client, les statistiques relatives aux objets normalConnectionConsumer et highConnectionConsumer, ainsi qu'aux fabriques CountingMessageListenerFactory associées, sont affichées à l'écran toutes les 10 secondes. Les statistiques relatives au troisième objet ConnectionConsumer et aux fabriques LoggingMessageListenerFactory associées sont consignées dans le fichier journal.

Vous pouvez rechercher à l'écran et dans le fichier journal la destination réelle des messages. Additionnez les totaux de chaque programme d'écoute CountingMessageListener et examinez le nombre d'entrées de fichier journal.

La répartition des messages est normalement différente de celle obtenue avec la version point à point de la même application (ASFClient2.java). En effet, dans un domaine de publication/souscription, chaque client d'une rubrique obtient une copie personnelle de tous les messages publiés dans cette rubrique. Dans cette application, les objets ConnectionConsumer 'high' et 'normal' reçoivent respectivement environ 25 % et 75 % de la charge publiée pour une rubrique donnée. Le troisième objet ConnectionConsumer reçoit 100 % de la charge. Par conséquent, le nombre total de messages reçus dépasse 100 % de la charge initialement publiée pour la rubrique.

Chapitre 14. Interfaces et classes JMS

Les classes MQSeries pour Java Message Service consistent en un certain nombre de classes et d'interfaces JMS fondées sur le module Sun javax.jms d'interfaces et de classes. Les modules client doivent être écrits à l'aide des interfaces et des classes Sun indiquées ci-dessous et décrites en détail dans les sections suivantes. Les noms des objets MQSeries qui implémentent les interfaces et les classes Sun ont le préfixe 'MQ' (sauf indication contraire dans la description de l'objet). Les descriptions comprennent les caractéristiques des objets MQSeries correspondant aux définitions JMS standards. Ces différences sont indiquées par le signe '*'.

Classes et interfaces Sun Java Message Service

Les tableaux suivants répertorient les objets JMS contenus dans le module **javax.jms**. Les interfaces signalées par le signe '*****' sont mises en oeuvre par des applications. Les interfaces signalées par le signe '******' sont mises en oeuvre par des serveurs d'applications.

Tableau 26. Récapitulatif des interfaces

Interface	Description
BytesMessage	BytesMessage est utilisé pour envoyer un message contenant une chaîne d'octets non interprétés.
Connexion	Une connexion JMS désigne la connexion active d'un client à son fournisseur JMS.
ConnectionConsumer	Pour les serveurs d'applications, Connections fournit une fonction spéciale permettant de créer un objet ConnectionConsumer.
ConnectionFactory	Une classe ConnectionFactory regroupe un ensemble de paramètres de configuration de connexion définis au niveau administrateur.
ConnectionMetaData	ConnectionMetaData fournit des informations sur la connexion.
DeliveryMode	Modes de livraison pris en charge par JMS.
Destination	Interface parent pour Queue et Topic.
ExceptionListener*	Un module d'écoute d'exceptions reçoit les exceptions déclenchées par les unités d'exécution de livraison asynchrone de Connections.
MapMessage	MapMessage envoie un ensemble de paires nom-valeur pour lesquelles les noms sont des chaînes et les valeurs, des types de Java primitives.
Message	Message est l'interface racine de tous les messages JMS.
MessageConsumer	Interface parent de tous les clients de messages.
MessageListener*	MessageListener permet de recevoir les messages en livraison asynchrone.
MessageProducer	Un client utilise un MessageProducer pour envoyer des messages vers une Destination.
ObjectMessage	ObjectMessage envoie un message contenant un objet Java sérialisable.

Tableau 26. Récapitulatif des interfaces (suite)

Interface	Description
Queue	Un objet Queue contient un nom de file d'attente spécifique à un fournisseur.
QueueBrowser	Un client utilise un QueueBrowser pour consulter des messages dans une file d'attente sans les retirer.
QueueConnection	QueueConnection est une connexion active avec un fournisseur JMS point à point.
QueueConnectionFactory	Un client utilise un objet QueueConnectionFactory pour créer un objet QueueConnections avec un fournisseur JMS point à point.
QueueReceiver	Un client utilise un QueueReceiver pour recevoir des messages placés en file d'attente.
QueueSender	Un client utilise un QueueSender pour envoyer des messages vers une file d'attente.
QueueSession	QueueSession fournit des méthodes de création de QueueReceivers, QueueSenders, QueueBrowsers et TemporaryQueues.
ServerSession **	ServerSession est un objet mis en oeuvre par un serveur d'applications.
ServerSessionPool **	ServerSessionPool est un objet mis en oeuvre par un serveur d'applications pour fournir un groupe d'interfaces ServerSessions de traitement des messages d'une interface ConnectionConsumer.
Session	Une session JMS est un contexte unique conçu avec des unités d'exécution pour produire et traiter des messages.
StreamMessage	StreamMessage envoie une chaîne de Java primitives.
TemporaryQueue	TemporaryQueue est un objet Queue unique créé pour la durée de QueueConnection.
TemporaryTopic	TemporaryTopic est un objet Topic unique créé pour la durée de TopicConnection.
TextMessage	TextMessage envoie un message contenant `java.lang.String`.
Topic	Un objet Topic contient une rubrique spécifique à un fournisseur.
TopicConnection	TopicConnection est une connexion active avec un fournisseur JMS de publication/souscription.
TopicConnectionFactory	Un client utilise une TopicConnectionFactory pour créer des connexions TopicConnections avec un fournisseur JMS de publication/souscription.
TopicPublisher	Un client utilise un TopicPublisher pour publier des messages relatifs à une rubrique.
TopicSession	TopicSession fournit des méthodes de création de TopicPublishers, TopicSubscribers et TemporaryTopics.
TopicSubscriber	Un client utilise un TopicSubscriber pour recevoir des messages publiés dans une rubrique.
XAConnection	XAConnection accroît la capacité de Connection grâce à XASession.

Tableau 26. Récapitulatif des interfaces (suite)

Interface	Description
XAConnectionFactory	Certains serveurs d'applications offrent une assistance pour l'utilisation des ressources Java Transaction Service (JTS) dans les transactions distribuées.
XAQueueConnection	L'interface XAQueueConnection offre les mêmes options de création que QueueConnection.
XAQueueConnectionFactory	L'interface XAQueueConnectionFactory offre les mêmes options de création que QueueConnectionFactory.
XAQueueSession	L'interface XAQueueSession fournit une interface QueueSession classique permettant de créer des objets QueueReceivers, QueueSenders et QueueBrowsers.
XASession	XASession accroît la capacité de Session en créant un accès à l'assistance Java Transaction API (JTA) du fournisseur JMS.
XATopicConnection	L'interface XATopicConnection offre les mêmes options de création que TopicConnection.
XATopicConnectionFactory	L'interface XATopicConnectionFactory offre les mêmes options de création que TopicConnectionFactory.
XATopicSession	L'interface XATopicSession fournit une interface TopicSession classique permettant de créer des objets TopicSubscribers et TopicPublishers.

Tableau 27. Récapitulatif des classes

Classe	Description
QueueRequestor	JMS fournit une classe d'aide programmable QueueRequestor pour simplifier les demandes de service.
TopicRequestor	JMS fournit une classe d'aide programmable TopicRequestor pour simplifier les demandes de service.

Classes MQSeries JMS

Les tableaux suivants répertorient les modules **com.ibm.mq.jms** et **com.ibm.jms** qui contiennent les classes MQSeries qui mettent en oeuvre les interfaces Sun.

Tableau 28. Récapitulatif des classes du module 'com.ibm.mq.jms'

Classe	Met en oeuvre
MQConnection	Connection
MQConnectionConsumer	ConnectionConsumer
MQConnectionFactory	ConnectionFactory
MQConnectionMetaData	ConnectionMetaData
MQDestination	Destination
MQMessageConsumer	MessageConsumer
MQMessageProducer	MessageProducer
MQQueue	File d'attente
MQQueueBrowser	QueueBrowser
MQQueueConnection	QueueConnection
MQQueueConnectionFactory	QueueConnectionFactory
MQQueueEnumeration	java.util.Enumération de QueueBrowser
MQQueueReceiver	QueueReceiver
MQQueueSender	QueueSender
MQQueueSession	QueueSession
MQSession	Session
MQTemporaryQueue	TemporaryQueue
MQTemporaryTopic	TemporaryTopic
MQTopic	Rubrique
MQTopicConnection	TopicConnection
MQTopicConnectionFactory	TopicConnectionFactory
MQTopicPublisher	TopicPublisher
MQTopicSession	TopicSession
MQTopicSubscriber	TopicSubscriber
MQXAConnection	XAConnection
MQXAConnectionFactory	XAConnectionFactory
MQXAQueueConnection	XAQueueConnection
MQXAQueueConnectionFactory	XAQueueConnectionFactory
MQXAQueueSession	XAQueueSession
MQXASession	XASession
MQXATopicConnection	XATopicConnection
MQXATopicConnectionFactory	XATopicConnectionFactory
MQXATopicSession	XATopicSession

Tableau 29. Récapitulatif des classes du module 'com.ibm.jms'

Classe	**Met en oeuvre**
JMSBytesMessage	BytesMessage
JMSMapMessage	MapMessage
JMSMessage	Message
JMSObjectMessage	ObjectMessage
JMSStreamMessage	StreamMessage
JMSTextMessage	TextMessage

Un exemple de mise en oeuvre des interfaces JMS suivantes est fourni dans cette édition des classes MQSeries pour Java Message Service.

- ServerSession
- ServerSessionPool

Voir «Code exemple de serveur d'applications» à la page 224.

BytesMessage

public interface **BytesMessage**
extends **Message**

Classe MQSeries : **JMSBytesMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

BytesMessage est utilisé pour envoyer un message contenant une chaîne d'octets non interprétés. Il reçoit un **Message** et ajoute un corps de message. Le réceptionnaire du message fournit l'interprétation des octets.

Remarque : Ce type de message est destiné au codage client de formats de message existants. Si possible, l'un des autres types de message auto-défini doit être utilisé à la place.

Voir aussi : **MapMessage**, **Message**, **ObjectMessage**, **StreamMessage** et **TextMessage**.

Méthodes

readBoolean

```
public boolean readBoolean() throws JMSException
```

Lit un booléen dans le message d'octets.

Renvoie :
la valeur booléenne lue.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageEOFException - en cas de fin des octets du message.

readByte

```
public byte readByte() throws JMSException
```

Lit dans le message d'octets une valeur signée à huit bits.

Renvoie :
l'octet suivant du message d'octets sous la forme d'un octet signé à 8 bits.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readUnsignedByte

```
public int readUnsignedByte() throws JMSException
```

Lit dans le message d'octets un nombre non signé à huit bits.

Renvoie :

l'octet suivant du message d'octets, interprété comme un nombre non signé à 8 bits.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readShort

```
public short readShort() throws JMSException
```

Lit dans le message d'octets une valeur signée à seize bits.

Renvoie :

les deux octets suivants du message d'octets, interprétés comme un nombre signé à 16 bits.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readUnsignedShort

```
public int readUnsignedShort() throws JMSException
```

Lit dans le message d'octets un nombre non signé à seize bits.

Renvoie :

les deux octets suivants du message d'octets, interprétés comme un entier non signé à 16 bits.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readChar

```
public char readChar() throws JMSException
```

Lit un caractère unicode dans le message d'octets.

Renvoie :

les deux octets suivants du message d'octets sous la forme d'un caractère unicode.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readInt

```
public int readInt() throws JMSException
```

Lit dans le message d'octets un entier signé à trente-deux bits.

Renvoie :

les quatre octets suivants du message d'octets, interprétés comme un entier.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readLong

```
public long readLong() throws JMSException
```

Lit dans le message d'octets un entier signé à soixante-quatre bits.

Renvoie :

les huit octets suivants du message d'octets, interprétés comme un entier long.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readFloat

```
public float readFloat() throws JMSException
```

Lit dans le message d'octets une variable flottante.

Renvoie :

les quatre octets suivants du message d'octet, interprétés comme une variable flottante.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readDouble

```
public double readDouble() throws JMSException
```

Lit un double dans le message d'octets.

Renvoie :
les huit octets suivants du message d'octets, interprétés comme un double.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readUTF

```
public java.lang.String readUTF() throws JMSException
```

Lit dans le message d'octets une chaîne codée à l'aide du format UTF-8. Les deux premiers octets sont interprétés comme une zone de deux octets.

Renvoie :
une chaîne unicode du message d'octets.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- MessageEOFException - en cas de fin des octets du message.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readBytes

```
public int readBytes(byte[] value) throws JMSException
```

Lit dans le message d'octets un tableau d'octets. Si le flot contient suffisamment d'octets, la mémoire tampon est remplie en totalité. Dans le cas contraire, la mémoire est partiellement remplie.

Arguments :
value - la mémoire tampon dans laquelle les données sont lues.

Renvoie :
le nombre total d'octets lus dans la mémoire tampon, ou -1 en l'absence de données lorsque la fin des octets a été atteinte.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

readBytes

```
public int readBytes(byte[] value, int length)
                                        throws JMSException
```

Lit une partie du message d'octets.

Arguments :

- value - la mémoire tampon dans laquelle les données sont lues.
- length - le nombre d'octets à lire.

Renvoie :

le nombre total d'octets lus dans la mémoire tampon, ou -1 en l'absence de données lorsque la fin des octets a été atteinte.

Génère :

- MessageNotReadableException - si le message est en mode écriture seule.
- IndexOutOfBoundsException - si length est une valeur négative ou inférieure à la longueur du tableau value.
- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

Ecrit un booléen dans le message d'octets sous la forme d'une valeur d'un octet. La valeur true (vrai) est exprimée par la valeur (byte)1 et la valeur false (faux) est exprimée par la valeur (byte)0.

Arguments :

value - la valeur booléenne à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeByte

```
public void writeByte(byte value) throws JMSException
```

Ecrit un octet dans le message d'octets sous la forme d'une valeur d'un octet.

Arguments :

value - la valeur octet à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeShort

```
public void writeShort(short value) throws JMSException
```

Ecrit une valeur courte dans le message d'octets sous forme de deux octets.

Arguments :

value - la valeur courte à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeChar

```
public void writeChar(char value) throws JMSException
```

Ecrit un caractère dans le message d'octets sous la forme d'une valeur à deux octets, en indiquant l'octet de poids fort en premier.

Arguments :

value - la valeur de type caractère à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeInt

```
public void writeInt(int value) throws JMSException
```

Ecrit un entier dans le message d'octets sous forme de quatre octets.

Arguments :

value - l'entier à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeLong

```
public void writeLong(long value) throws JMSException
```

Ecrit une valeur longue dans le message d'octets sous forme de huit octets.

Arguments :

value - la valeur longue à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeFloat

```
public void writeFloat(float value) throws JMSException
```

Convertit l'argument flottant en un entier à l'aide de la méthode floatToIntBits de la classe Float, puis écrit cette valeur entière dans le message d'octets sous forme de 4 octets.

Arguments :

value - la valeur flottante à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeDouble

```
public void writeDouble(double value) throws JMSException
```

Convertit l'argument double en une valeur longue à l'aide de la méthode doubleToLongBits de la classe Double, puis écrit cette valeur longue dans le message d'octets sous forme d'une valeur à huit octets.

Arguments :

value - la valeur double à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeUTF

```
public void writeUTF(java.lang.String value)
                              throws JMSException
```

Ecrit une chaîne dans le message d'octets à l'aide du codage UTF-8 de manière indépendante de la machine. La chaîne UTF-8 écrite dans la mémoire tampon commence par une zone de deux octets.

Arguments :

value - la chaîne à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

Ecrit un tableau d'octets dans le message d'octets.

Arguments :

value - le tableau d'octets à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeBytes

```
public void writeBytes(byte[] value,
                       int length) throws JMSException
```

Ecrit une partie d'un tableau d'octets dans le message d'octets.

Arguments :

- `value` - la valeur du tableau d'octets à écrire.
- `offset` - le décalage initial dans le tableau d'octets.
- `length` - le nombre d'octets à utiliser.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

writeObject

```
public void writeObject(java.lang.Object value)
                                    throws JMSException
```

Ecrit un objet Java dans le message d'octets.

Remarque : Cette méthode ne s'applique qu'aux types d'objets primitifs (tel que Entier, Double et Long), aux chaînes et aux tableaux d'octets.

Arguments :

`value` - l'objet Java à écrire.

Génère :

- MessageNotWriteableException - si le message est en mode lecture seule.
- MessageFormatException - si l'objet est de type incorrect.
- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

reset

```
public void reset() throws JMSException
```

Attribue au corps de message le mode lecture seule et repositionne les octets au début.

Génère :

- JMSException - si JMS ne parvient pas à réinitialiser le message du fait d'une erreur interne JMS.
- MessageFormatException - si le format du message est incorrect.

Connection

public interface **Connection**
Interfaces secondaires : **QueueConnection**, **TopicConnection**, **XAQueueConnection** et **XATopicConnection**

Classe MQSeries : **MQConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

Une connexion JMS désigne une connexion active du client à son fournisseur JMS.

Voir aussi : **QueueConnection**, **TopicConnection**, **XAQueueConnection** et **XATopicConnection**

Méthodes

getClientID

```
public java.lang.String getClientID()
throws JMSException
```

Extrait l'identificateur client pour cette connexion. Celui-ci peut être prédéfini par l'administrateur dans un objet ConnectionFactory ou à l'aide de la méthode setClientId.

Renvoie :
l'identificateur client unique.

Génère :
JMSException - si JMS ne parvient pas à renvoyer l'identificateur client pour cette connexion du fait d'une erreur interne.

setClientID

```
public void setClientID(java.lang.String clientID)
throws JMSException
```

Définit l'identificateur client pour cette connexion.

Remarque : L'identificateur client est ignoré pour les connexions point à point.

Arguments
clientID - l'identificateur client unique.

Génère :

- JMSException - si JMS ne parvient pas à définir l'identificateur client pour cette connexion du fait d'une erreur interne.
- InvalidClientIDException - si le client JMS spécifie un identificateur client non valide ou existant.
- IllegalStateException - si l'identificateur client d'une connexion est défini à un moment inapproprié ou s'il a été configuré par l'administrateur.

getMetaData

```
public ConnectionMetaData getMetaData()throws JMSException
```

Extrait les métadonnées de cette connexion.

Renvoie :
les métadonnées de la connexion.

Génère :
JMSException - exception générale si JMS ne parvient pas à extraire les métadonnées de la connexion.

Voir aussi :
ConnectionMetaData

getExceptionListener

```
public ExceptionListener getExceptionListener()
throws JMSException
```

Extrait le programme d'écoute d'exception pour cette connexion.

Renvoie :
le programme d'écoute d'exception pour cette connexion

Génère :
JMSException - exception générale si JMS ne parvient pas à extraire le programme d'écoute d'exception de cette connexion.

setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
throws JMSException
```

Définit un programme d'écoute d'exception pour cette connexion.

Arguments
handler - programme d'écoute d'exception.

Génère :
JMSException - exception générale si JMS ne parvient pas à définir le programme d'écoute d'exception de cette connexion.

start

```
public void start() throws JMSException
```

Démarre (ou redémarre) la réception des messages pour cette connexion. Le démarrage d'une session déjà ouverte est ignoré.

Génère :
JMSException - si JMS ne parvient pas à démarrer la réception des messages du fait d'une erreur interne.

Voir aussi :
stop

stop

```
public void stop() throws JMSException
```

Interrompt provisoirement la réception des messages pour cette connexion. La réception peut être redémarrée à l'aide de la méthode `start`. Une fois la réception interrompue, la transmission des messages est bloquée. Les réceptions synchrones sont interrompues et les messages ne sont plus adressés à leurs destinataires.

L'arrêt d'une session n'a pas d'incidence sur l'envoi de messages. L'arrêt d'une session déjà arrêtée est ignoré.

Génère :

JMSException - si JMS ne parvient pas à interrompre la réception des messages du fait d'une erreur interne.

Voir aussi :

start

close

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt. Il est inutile de fermer les sessions, les producteurs et les clients d'une connexion interrompue.

L'arrêt d'une connexion peut engendrer l'annulation des transactions en cours de la session. Si un travail de session est coordonné par un gestionnaire de transactions externe et si un XASession est utilisé, les méthodes de validation et d'annulation de session ne sont pas utilisées et le résultat du travail de session fermé est déterminé ultérieurement par le gestionnaire de transactions. La fermeture d'une connexion n'impose pas d'accuser réception des sessions clientes ayant déjà fait l'objet d'un accusé de réception.

MQ JMS conserve un pool d'objets MQSeries hConns disponibles en vue de leur utilisation par des sessions. Dans certains cas, Connection.close() supprime ce pool. Si une application utilise plusieurs connexions successivement, il est possible de forcer le pool à rester actif entre deux connexions JMS. Pour ce faire, enregistrez un objet MQPoolToken à l'aide de com.ibm.mq.MQEnvironment afin de prolonger la durée de vie de votre application JMS. Pour plus de détails, reportez-vous aux sections «Définition d'un pool de connexion» à la page 66 et «MQEnvironment» à la page 93.

Génère :

JMSException - si JMS ne parvient pas à fermer la connexion du fait d'une erreur interne (incident lors de la libération des ressources ou de la fermeture d'une connexion socket, par exemple).

ConnectionConsumer

public interface **ConnectionConsumer**

Classe MQSeries : **MQConnectionConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionConsumer
```

Pour les serveurs d'applications, Connections fournit une fonction spéciale permettant de créer un objet ConnectionConsumer. Un objet Destination et un objet Property Selector indiquent les messages à traiter. De même, un objet ServerSessionPool doit être affecté au ConnectionConsumer afin que les messages puissent être traités.

Voir aussi : **QueueConnection** et **TopicConnection**.

Méthodes

close()

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

Génère :

JMSException - si JMS ne parvient pas à libérer les ressources pour ConnectionConsumer, ou s'il ne parvient pas à arrêter le client de la connexion.

getServerSessionPool()

```
public ServerSessionPool getServerSessionPool()
                                          throws JMSException
```

Extrait la session serveur associée au client de la connexion.

Renvoie :

le pool de sessions serveur utilisé par le client de la connexion.

Génère :

JMSException - si JMS ne parvient pas à extraire le pool de sessions serveur associé au client de la connexion du fait d'une erreur interne.

ConnectionFactory

public interface **ConnectionFactory**
Interfaces secondaires : **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** et **XATopicConnectionFactory**

Classe MQSeries : **MQConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

Une classe ConnectionFactory encapsule un ensemble de paramètres de configuration de connexion définis par l'administrateur. Cette classe permet à un client de créer une connexion à l'aide d'un fournisseur JMS.

Voir aussi : **QueueConnectionFactory**, **TopicConnectionFactory**, **XAQueueConnectionFactory** et **XATopicConnectionFactory**

Constructeur QSeries

MQConnectionFactory

```
public MQConnectionFactory()
```

Méthodes

setDescription *

```
public void setDescription(String x)
```

Brève description de l'objet.

getDescription *

```
public String getDescription()
```

Extrait la description de l'objet.

setTransportType *

```
public void setTransportType(int x) génère JMSException
```

Définit le type de transfert à utiliser : JMSC.MQJMS_TP_BINDINGS_MQ ou JMSC.MQJMS_TP_CLIENT_MQ_TCPIP.

getTransportType *

```
public int getTransportType()
```

Extrait le type de transfert.

setClientId *

```
public void setClientId(String x)
```

Définit l'identificateur client à utiliser pour toutes les connexions créées à l'aide de Connection.

getClientId *

```
public String getClientId()
```

Extrait l'identificateur client utilisé pour toutes les connexions créées à l'aide de ConnectionFactory.

setQueueManager *

```
public void setQueueManager(String x) génère JMSException
```

Définit le nom du gestionnaire de files d'attente à connecter.

getQueueManager *

```
public String getQueueManager()
```

Extrait le nom du gestionnaire de files d'attente.

setHostName *

```
public void setHostName(String hostname)
```

Définit le nom de l'hôte à connecter, pour les clients uniquement.

getHostName *

```
public String getHostName()
```

Extrait le nom de l'hôte.

setPort *

```
public void setPort(int port) génère JMSException
```

Définit le numéro de port d'une connexion cliente.

Arguments
port - la nouvelle valeur à utiliser.

Génère :
JMSException - si le numéro de port est négatif.

getPort *

```
public int getPort()
```

Extrait le numéro de port, pour les connexions clientes uniquement.

setChannel *

```
public void setChannel(String x) génère JMSException
```

Définit le canal à utiliser, pour les clients uniquement.

getChannel *

```
public String getChannel()
```

Extrait le canal utilisé, pour les clients uniquement.

setCCSID *

```
public void setCCSID(int x) génère JMSException
```

Définit le jeu de caractères à utiliser lors de la connexion au gestionnaire de files d'attente. Reportez-vous au tableau 13 à la page 111 pour consulter la liste des valeurs autorisées. Nous vous recommandons d'utiliser la valeur par défaut (819) dans la majorité des cas.

getCCSID *

```
public int getCCSID()
```

Extrait le jeu de caractères du gestionnaire de files d'attente.

setReceiveExit *

```
public void setReceiveExit(String receiveExit)
```

Nom de la classe qui met en œuvre un exit de réception.

getReceiveExit *

```
public String getReceiveExit()
```

Extrait le nom de la classe de l'exit de réception.

setReceiveExitInit *

```
public void setReceiveExitInit(String x)
```

Chaîne d'initialisation transmise au constructeur de la classe de l'exit de réception.

getReceiveExitInit *

```
public String getReceiveExitInit()
```

Extrait la chaîne d'initialisation transmise à la classe de l'exit de réception.

setSecurityExit *

```
public void setSecurityExit(String securityExit)
```

Nom de la classe qui met en œuvre un exit de sécurité.

getSecurityExit *

```
public String getSecurityExit()
```

Extrait le nom de la classe de l'exit de sécurité.

setSecurityExitInit *

```
public void setSecurityExitInit(String x)
```

Chaîne d'initialisation transmise au constructeur de la classe de l'exit de sécurité.

getSecurityExitInit *

```
public String getSecurityExitInit()
```

Extrait la chaîne d'initialisation de l'exit de sécurité.

setSendExit *

```
public void setSendExit(String sendExit)
```

Nom de la classe qui met en œuvre un exit d'envoi.

getSendExit *

```
public String getSendExit()
```

Extrait le nom de la classe de l'exit d'envoi.

setSendExitInit *

```
public void setSendExitInit(String x)
```

Chaîne d'initialisation transmise au constructeur de l'exit d'envoi.

getSendExitInit *

```
public String getSendExitInit()
```

Extrait la chaîne d'initialisation de l'exit d'envoi.

ConnectionMetaData

public interface **ConnectionMetaData**

Classe MQSeries : **MQConnectionMetaData**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

ConnectionMetaData fournit des informations sur la connexion.

Constructeur MQSeries

MQConnectionMetaData

```
public MQConnectionMetaData()
```

Méthodes

getJMSVersion

```
public java.lang.String getJMSVersion() throws JMSException
```

Extrait la version de JMS.

Renvoie :
la version de JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getJMSMajorVersion

```
public int getJMSMajorVersion() throws JMSException
```

Extrait le numéro de version principal de JMS.

Renvoie :
le numéro de version principal de JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getJMSMinorVersion

```
public int getJMSMinorVersion() throws JMSException
```

Extrait le numéro de version secondaire de JMS.

Renvoie :
le numéro de version secondaire de JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getJMSXPropertyNames

```
public java.util.Enumeration getJMSXPropertyNames()
throws JMSException
```

Extrait la liste des propriétés JMSX prises en charge par cette connexion.

Renvoie :
la liste des propriétés JMSX.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des noms de propriétés.

getJMSProviderName

```
public java.lang.String getJMSProviderName()
throws JMSException
```

Extrait le nom du fournisseur JMS.

Renvoie :
le nom du fournisseur JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getProviderVersion

```
public java.lang.String getProviderVersion()
throws JMSException
```

Extrait la version du fournisseur JMS.

Renvoie :
la version du fournisseur JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getProviderMajorVersion

```
public int getProviderMajorVersion() throws JMSException
```

Extrait le numéro de version principal du fournisseur JMS.

Renvoie :
le numéro de version principal du fournisseur JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

getProviderMinorVersion

```
public int getProviderMinorVersion() throws JMSException
```

Extrait le numéro de version secondaire du fournisseur JMS.

Renvoie :
le numéro de version secondaire du fournisseur JMS.

Génère :
JMSException - en cas d'erreur JMS interne pendant l'extraction des métadonnées.

toString *

```
public String toString()
```

Remplace :
toString dans la classe Object.

DeliveryMode

public interface **DeliveryMode**

Modes de livraison pris en charge par JMS.

Zones

NON_PERSISTENT

```
public static final int NON_PERSISTENT
```

Il s'agit du mode de livraison sollicitant le moins le système car il ne nécessite pas le stockage du message dans un emplacement de stockage stable.

PERSISTENT

```
public static final int PERSISTENT
```

Dans le cadre de l'opération d'envoi du client, ce mode demande au fournisseur JMS de consigner le message dans un emplacement de stockage stable.

Destination

public interface **Destination**
Interfaces secondaires : **Queue**, **TemporaryQueue**, **TemporaryTopic** et **Topic**

Classe MQSeries : **MQDestination**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

L'objet Destination contient les adresses propres au fournisseur.

Voir aussi : **Queue**, **TemporaryQueue**, **TemporaryTopic** et **Topic**

Constructeurs MQSeries

MQDestination

```
public MQDestination()
```

Méthodes

setDescription *

```
public void setDescription(String x)
```

Brève description de l'objet.

getDescription *

```
public String getDescription()
```

Extrait la description de l'objet.

setPriority *

```
public void setPriority(int priority) throws JMSException
```

Utilisé pour modifier le niveau de priorité de tous les messages envoyés vers cette destination.

getPriority *

```
public int getPriority()
```

Extrait la priorité modifiée.

setExpiry *

```
public void setExpiry(int expiry) throws JMSException
```

Utilisé pour modifier le délai d'expiration de tous les messages envoyés vers cette destination.

getExpiry *

```
public int getExpiry()
```

Extrait le délai d'expiration associé à cette destination.

setPersistence *

```
public void setPersistence(int persistence)
                                    throws JMSException
```

Utilisé pour modifier l'état de persistance de tous les messages envoyés vers cette destination.

getPersistence *

```
public int getPersistence()
```

Extrait l'état de persistance associé à cette destination.

setTargetClient *

```
public void setTargetClient(int targetClient)
                                    throws JMSException
```

Indique si l'application éloignée est compatible JMS ou pas.

getTargetClient *

```
public int getTargetClient()
```

Extrait l'indicateur de compatibilité JMS.

setCCSID *

```
public void setCCSID(int x) throws JMSException
```

Jeu de caractères à utiliser pour coder les chaînes de texte dans les messages envoyés vers cette destination. Reportez-vous au tableau 13 à la page 111 pour consulter la liste des valeurs autorisées. La valeur par défaut est 1208 (UTF8).

getCCSID *

```
public int getCCSID()
```

Extrait le nom du jeu de caractères utilisé par cette destination.

setEncoding *

```
public void setEncoding(int x) throws JMSException
```

Indique le codage à utiliser pour les zones numériques des messages envoyés vers cette destination. Reportez-vous au tableau 13 à la page 111 pour consulter la liste des valeurs autorisées.

getEncoding *

```
public int getEncoding()
```

Extrait le codage utilisé pour cette destination.

ExceptionListener

public interface **ExceptionListener**

Si un fournisseur JMS détecte un incident grave au niveau d'une connexion, il en informe le programme d'écoute ExceptionListener de la connexion si un programme de ce type a été enregistré. Pour cela, il appelle la méthode onException() du programme d'écoute et lui transmet une exception JMSException décrivant l'incident.

Cette opération permet à un client d'être informé d'un incident en mode asynchrone. Certaines connexions n'utilisent que des messages. Elles ne disposent donc pas d'autres moyens pour être averties d'un échec de leur connexion.

Des exceptions sont déclenchées dans les cas suivants :
- Echec de la réception d'un message en mode asynchrone
- Envoi par un message d'une exception d'exécution

Méthodes

onException

```
public void onException(JMSException exception)
```

Informe un utilisateur de l'arrivée d'une exception JMS.

Arguments

exception - exception JMS. Ce type d'exception résulte de la livraison d'un message en mode asynchrone. Généralement, ce type d'exception signale un incident lors de la réception d'un message provenant du gestionnaire de files d'attente ou une erreur interne de mise en oeuvre de JMS.

MapMessage

public interface **MapMessage**
extends **Message**

Classe MQSeries : **JMSMapMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

MapMessage est utilisé pour envoyer un ensemble de paires nom-valeur dans lesquelles les chaînes et les valeurs sont de type Java primitive. L'accès aux éléments peut être séquentiel ou par nom. L'ordre des entrées n'est pas défini.

Voir aussi : **BytesMessage**, **Message**, **ObjectMessage**, **StreamMessage** et **TextMessage**.

Méthodes

getBoolean

```
public boolean getBoolean(java.lang.String name)
                                throws JMSException
```

Renvoie la valeur booléenne correspondant au nom indiqué.

Arguments
name - le nom de la valeur booléenne

Renvoie :
la valeur booléenne correspondant au nom indiqué

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getByte

```
public byte getByte(java.lang.String name)
                                     throws JMSException
```

Renvoie la valeur d'octet correspondant au nom indiqué.

Arguments
name - le nom de l'octet

Renvoie :
la valeur d'octet correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getShort

```
public short getShort(java.lang.String name) throws JMSException
```

Renvoie la valeur de la variable courte correspondant au nom indiqué.

Arguments

name - le nom de la variable courte

Renvoie :

la valeur de la variable courte correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getChar

```
public char getChar(java.lang.String name)
                            throws JMSException
```

Renvoie la valeur du caractère Unicode correspondant au nom indiqué.

Arguments

name - le nom du caractère Unicode

Renvoie :

la valeur du caractère Unicode correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getInt

```
public int getInt(java.lang.String name)
                            throws JMSException
```

Renvoie la valeur de l'entier correspondant au nom indiqué.

Arguments

name - le nom de l'entier

Renvoie :

la valeur de l'entier correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getLong

```
public long getLong(java.lang.String name)
                            throws JMSException
```

Renvoie la valeur de la variable longue correspondant au nom indiqué.

Arguments

name - le nom de la variable longue

Renvoie :
la valeur de la variable longue correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getFloat

```
public float getFloat(java.lang.String name) throws JMSException
```

Renvoie la valeur de la variable flottante correspondant au nom indiqué.

Arguments
name - le nom de la variable flottante

Renvoie :
la valeur de la variable flottante correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getDouble

```
public double getDouble(java.lang.String name) throws JMSException
```

Renvoie la valeur double correspondant au nom indiqué.

Arguments
name - le nom de la valeur double

Renvoie :
la valeur double correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getString

```
public java.lang.String getString(java.lang.String name)
                           throws JMSException
```

Renvoie la valeur de la chaîne correspondant au nom indiqué.

Arguments
name - le nom de la chaîne

Renvoie :
la valeur de la chaîne correspondant au nom indiqué. Si aucun élément ne porte ce nom, une valeur nulle est renvoyée.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

- MessageFormatException - si la conversion de ce type n'est pas valide.

getBytes

```
public byte[] getBytes(java.lang.String name) throws JMSException
```

Renvoie la valeur de tableau d'octets correspondant au nom indiqué.

Arguments

name : nom du tableau d'octets

Renvoie :

une valeur de tableau d'octets correspondant au nom indiqué. Si aucun élément ne porte ce nom, une valeur nulle est renvoyée.

Génère :

- JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.
- MessageFormatException - si la conversion de ce type n'est pas valide.

getObject

```
public java.lang.Object getObject(java.lang.String name)
                           throws JMSException
```

Renvoie la valeur de l'objet Java correspondant au nom indiqué. Cette méthode renvoie, au format objet, une valeur enregistrée dans Map à l'aide de l'appel de la méthode setObject ou de la méthode set primitive équivalente.

Arguments

name - le nom de l'objet Java.

Renvoie :

une copie de la valeur de l'objet Java correspondant au nom indiqué, au format objet (renvoie un entier en cas de définition en tant qu'entier). Si aucun élément ne porte ce nom, une valeur nulle est renvoyée.

Génère :

JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

getMapNames

```
public java.util.Enumeration getMapNames() throws JMSException
```

Renvoie une énumération de tous les noms du message Map.

Renvoie :

une énumération de tous les noms de ce message Map.

Génère :

JMSException - si JMS ne parvient pas à lire le message du fait d'une erreur interne JMS.

setBoolean

```
public void setBoolean(java.lang.String name,
                       boolean value) throws JMSException
```

Définit dans Map une valeur booléenne correspondant au nom indiqué.

Arguments

- name - nom de la valeur booléenne.
- value - valeur Booléenne à définir dans Map

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setByte

```
public void setByte(java.lang.String name,
                    byte value) throws JMSException
```

Définit dans Map une valeur d'octet correspondant au nom indiqué.

Arguments

- name - le nom de l'octet
- value - la valeur d'octet à définir dans Map

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setShort

```
public void setShort(java.lang.String name,
                     short value) throws JMSException
```

Définit dans Map une valeur courte correspondant au nom indiqué

Arguments

- name - le nom de la valeur courte.
- value - la valeur courte à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS
- MessageNotWriteableException - si le message est en mode lecture seule.

setChar

```
public void setChar(java.lang.String name,
                    char value) throws JMSException
```

Définit dans Map une valeur de caractère Unicode correspondant au nom indiqué.

Arguments

- name - le nom du caractère Unicode
- value - la valeur du caractère Unicode à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.

- MessageNotWriteableException - si le message est en mode lecture seule.

setInt

```
public void setInt(java.lang.String name,
                   int value) throws JMSException
```

Définit dans Map une valeur d'entier correspondant au nom indiqué.

Arguments

- name - le nom de l'entier
- value - la valeur de l'entier à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setLong

```
public void setLong(java.lang.String name,
                    long value) throws JMSException
```

Définit dans Map une valeur de variable longue correspondant au nom indiqué.

Arguments

- name - le nom de la variable longue
- value - la valeur de la variable longue à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setFloat

```
public void setFloat(java.lang.String name,
                     float value) throws JMSException
```

Définit dans Map une valeur de variable flottante correspondant au nom indiqué.

Arguments

- name - le nom de la variable flottante
- value - la valeur de la variable flottante à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setDouble

```
public void setDouble(java.lang.String name,
                      double value) throws JMSException
```

Définit dans Map une valeur double correspondant au nom indiqué.

Arguments

- name - le nom de la valeur double
- value - la valeur double à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setString

```
public void setString(java.lang.String name,
                      java.lang.String value) throws JMSException
```

Définit dans Map une valeur de chaîne correspondant au nom indiqué.

Arguments

- name - le nom de la chaîne
- value - la valeur de la chaîne à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value) throws JMSException
```

Définit dans Map une valeur de tableau d'octets correspondant au nom indiqué.

Arguments

- name - le nom du tableau d'octets
- value - la valeur du tableau d'octets à définir dans Map

  Le tableau est copié, si bien que la valeur de la mappe n'est pas modifiée en cas de modifications ultérieures apportées au tableau.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setBytes

```
public void setBytes(java.lang.String name,
                     byte[] value,
                     int offset,
                     int length) throws JMSException
```

Définit dans Map une partie de la valeur du tableau d'octets correspondant au nom indiqué.

Le tableau est copié, si bien que la valeur de la mappe n'est pas modifiée en cas de modifications ultérieures apportées au tableau.

Arguments

- name - le nom du tableau d'octets
- value - la valeur du tableau d'octets à définir dans Map
- offset - le décalage initial dans le tableau d'octets.
- length - le nombre d'octets à copier.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

setObject

```
public void setObject(java.lang.String name,
                      java.lang.Object value) throws JMSException
```

Définit dans Map une valeur d'objet Java correspondant au nom indiqué. Cette méthode ne s'applique qu'aux objets de type primitive (tels que entier, double et variable longue), aux chaînes et aux tableaux d'octets.

Arguments

- name - le nom de l'objet Java.
- value - la valeur de l'objet Java à définir dans Map.

Génère :

- JMSException - si JMS ne parvient pas à écrire le message du fait d'une erreur interne JMS.
- MessageFormatException - si l'objet est incorrect.
- MessageNotWriteableException - si le message est en mode lecture seule.

itemExists

```
public boolean itemExists(java.lang.String name)
                                    throws JMSException
```

Vérifie si un élément existe dans MapMessage.

Arguments

name - le nom de l'élément à tester.

Renvoie :

la valeur vrai si l'élément existe.

Génère :

JMSException - en cas d'erreur JMS.

Message

public interface **Message**
Interfaces secondaires : **BytesMessage**, **MapMessage**, **ObjectMessage**, **StreamMessage** et **TextMessage**

Classe MQSeries : **JMSMessage**

```
java.lang.Object
  |
  +----com.ibm.jms.MQJMSMessage
```

L'interface Message est l'interface racine de tous les messages JMS. Elle définit l'en-tête JMS et la méthode d'accusé de réception utilisée pour tous les messages.

Zones

DEFAULT_DELIVERY_MODE

```
public static final int DEFAULT_DELIVERY_MODE
```

Valeur du mode de communication des messages par défaut.

DEFAULT_PRIORITY

```
public static final int DEFAULT_PRIORITY
```

Valeur de la priorité par défaut.

DEFAULT_TIME_TO_LIVE

```
public static final long DEFAULT_TIME_TO_LIVE
```

Valeur de la durée de vie par défaut.

Méthodes

getJMSMessageID

```
public java.lang.String getJMSMessageID()
                                    throws JMSException
```

Extrait l'ID message.

Renvoie :
L'ID message.

Génère :
JMSException - si JMS ne réussit pas à extraire l'ID message en raison d'une erreur interne JMS.

Voir aussi :
setJMSMessageID()

setJMSMessageID

```
public void setJMSMessageID(java.lang.String id)
                                          throws JMSException
```

Définit l'ID message.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Arguments
id - ID message.

Génère :
JMSException - si JMS ne réussit pas à extraire l'ID message en raison d'une erreur interne JMS.

Voir aussi :
getJMSMessageID()

getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException
```

Extrait l'heure de consignation du message.

Renvoie :
L'heure de consignation du message.

Génère :
JMSException - si JMS ne réussit pas à extraire l'heure de consignation en raison d'une erreur interne JMS.

Voir aussi :
setJMSTimestamp()

setJMSTimestamp

```
public void setJMSTimestamp(long timestamp)
                                        throws JMSException
```

Définit l'heure de consignation du message.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Arguments
timestamp - heure de consignation du message.

Génère :
JMSException - si JMS ne réussit pas à définir l'heure de consignation en raison d'une erreur interne JMS.

Voir aussi :
getJMSTimestamp()

getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes()
                                                    throws JMSException
```

Extrait l'ID de corrélation du message sous la forme d'un tableau d'octets.

Renvoie :
L'ID de corrélation d'un message sous forme de tableau d'octets.

Génère :
JMSException - si JMS ne réussit pas à extraire l'ID de corrélation en raison d'une erreur interne JMS.

Voir aussi :
setJMSCorrelationID(), getJMSCorrelationID(), setJMSCorrelationIDAsBytes()

setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[]
                                       correlationID)
                                              throws JMSException
```

Définit l'ID de corrélation du message sous la forme d'un tableau d'octets. Un client peut utiliser cet appel pour définir un ID de corrélation égal à l'ID message d'un message précédent ou à une chaîne spécifique d'une application. Les chaînes spécifiques ne doivent pas commencer par les ID caractère.

Arguments

correlationID - ID de corrélation sous forme de chaîne ou ID message d'un message auquel il est fait référence.

Génère :

JMSException - si JMS ne réussit pas à définir l'ID de corrélation en raison d'une erreur interne JMS.

Voir aussi :

setJMSCorrelationID(), getJMSCorrelationID(), getJMSCorrelationIDAsBytes()

getJMSCorrelationID

```
public java.lang.String getJMSCorrelationID()
                                          throws JMSException
```

Extrait l'ID de corrélation du message.

Renvoie :

L'ID de corrélation d'un message sous forme de chaîne.

Génère :

JMSException - si JMS ne réussit pas à extraire l'ID de corrélation en raison d'une erreur interne JMS.

Voir aussi :

setJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

setJMSCorrelationID

```
public void setJMSCorrelationID
                (java.lang.String correlationID)
                                       throws JMSException
```

Définit l'ID de corrélation du message.

Un client peut utiliser la zone d'en-tête JMSCorrelationID pour lier un message à un autre. On l'utilise généralement pour lier un message de réponse à un message de demande.

Remarque : L'utilisation d'une valeur de type octet[] pour JMSCorrelationID n'est pas portable.

Arguments

correlationID - ID message d'un message auquel il est fait référence.

Génère :

JMSException - si JMS ne réussit pas à définir l'ID de corrélation en raison d'une erreur interne JMS.

Voir aussi :
getJMSCorrelationID(), getJMSCorrelationIDAsBytes(), setJMSCorrelationIDAsBytes()

getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException
```

Extrait la destination de la réponse à un message.

Renvoie :
La destination de la réponse à un message.

Génère :
JMSException - si JMS ne réussit pas à extraire la destination de la réponse en raison d'une erreur interne JMS.

Voir aussi :
setJMSReplyTo()

setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo)
                                        throws JMSException
```

Définit la destination de la réponse à un message.

Arguments
replyTo - Destination de la réponse à un message. Une valeur NULL indique qu'aucune réponse n'est attendue.

Génère :
JMSException - si JMS ne réussit pas à définir la destination de la réponse en raison d'une erreur interne JMS.

Voir aussi :
getJMSReplyTo()

getJMSDestination

```
public Destination getJMSDestination() throws JMSException
```

Extrait la destination du message.

Renvoie :
La destination du message.

Génère :
JMSException - si JMS ne réussit pas à extraire la destination JMS en raison d'une erreur interne JMS.

Voir aussi :
setJMSDestination()

setJMSDestination

```
public void setJMSDestination(Destination destination)
                                              throws JMSexception
```

Définit la destination du message.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Arguments
destination - Destination du message.

Génère :
JMSException - si JMS ne réussit pas à définir la destination JMS en raison d'une erreur interne JMS.

Voir aussi :
getJMSDestination()

getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException
```

Extrait le mode de communication du message.

Renvoie :
Le mode de communication du message.

Génère :
JMSException - si JMS ne réussit pas à extraire le mode de communication JMS en raison d'une erreur interne JMS.

Voir aussi :
setJMSDeliveryMode(), DeliveryMode

setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode)
                                          throws JMSException
```

Définit le mode de communication du message.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Pour modifier le mode de communication d'un message lors de son envoi, utilisez la méthode setDeliveryMode au niveau du QueueSender ou du TopicPublisher (cette méthode provient de MessageProducer).

Arguments
deliveryMode - Mode de communication du message.

Génère :
JMSException - si JMS ne réussit pas à définir le mode de communication JMS en raison d'une erreur interne JMS.

Voir aussi :
getJMSDeliveryMode(), DeliveryMode

getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException
```

Extrait des informations indiquant si ce message fait l'objet d'une seconde communication.

Lorsqu'un client reçoit un message comportant un indicateur de seconde communication, cela signifie généralement que ce message a déjà été envoyé au client mais que ce dernier n'en a pas accusé réception.

Renvoie :
La condition TRUE si ce message fait l'objet d'une seconde communication.

Génère :

JMSException - si JMS ne réussit pas à extraire l'indicateur de seconde communication JMS en raison d'une erreur interne JMS.

Voir aussi :

setJMSRedelivered()

setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered)
                                                throws JMSException
```

Définit des informations indiquant si ce message fait l'objet d'une seconde communication.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Arguments

redelivered - Indique si ce message fait l'objet d'une seconde communication.

Génère :

JMSException - si JMS ne réussit pas à définir un indicateur JMS de seconde communication en raison d'une erreur interne JMS.

Voir aussi :

getJMSRedelivered()

getJMSType

```
public java.lang.String getJMSType() throws JMSException
```

Extrait le type du message.

Renvoie :

Le type du message.

Génère :

JMSException - si JMS ne réussit pas à extraire le type du message JMS en raison d'une erreur interne JMS.

Voir aussi :

setJMSType()

setJMSType

```
public void setJMSType(java.lang.String type)
                                          throws JMSException
```

Définit le type du message.

Les clients JMS doivent affecter une valeur au type du message, que l'application l'utilise ou pas. Cela permet de s'assurer que le type du message est correctement défini pour les fournisseurs qui en ont besoin.

Arguments

type - Classe du message.

Génère :

JMSException - si JMS ne réussit pas à définir le type du message JMS en raison d'une erreur interne JMS.

Voir aussi :
getJMSType()

getJMSExpiration

```
public long getJMSExpiration() throws JMSException
```

Extrait la valeur du délai d'expiration du message.

Renvoie :
Le délai d'expiration du message. Il s'agit de la somme de la durée de vie spécifiée par le client et de l'heure GMT au moment de l'envoi.

Génère :
JMSException - si JMS ne réussit pas à extraire le délai d'expiration du message JMS en raison d'une erreur interne JMS.

Voir aussi :
setJMSExpiration()

setJMSExpiration

```
public void setJMSExpiration(long expiration)
                                        throws JMSException
```

Définit la valeur du délai d'expiration du message.

Toute valeur définie via cette méthode est ignorée lors de l'envoi du message, mais cette méthode peut être utilisée pour modifier la valeur dans un message reçu.

Arguments
expiration - Délai d'expiration du message.

Génère :
JMSException - si JMS ne réussit pas à définir le délai d'expiration JMS du message en raison d'une erreur interne JMS.

Voir aussi :
getJMSExpiration()

getJMSPriority

```
public int getJMSPriority() throws JMSException
```

Extrait la priorité du message.

Renvoie :
La priorité du message.

Génère :
JMSException - si JMS ne réussit pas à extraire la priorité JMS du message en raison d'une erreur interne JMS.

Voir aussi :
setJMSPriority() pour les niveaux de priorité

setJMSPriority

```
public void setJMSPriority(int priority)
                                    throws JMSException
```

Définit la priorité du message.

JMS définit une priorité à dix niveaux, pouvant aller de 0 (priorité la plus faible) à 9 (priorité la plus élevée). En outre, les clients doivent tenir compte de la priorité normale (0 à 4) et de la priorité majeure (5 à 9).

Arguments
priority - Priorité du message.

Génère :
JMSException - si JMS ne réussit pas à définir la priorité JMS du message en raison d'une erreur interne JMS.

Voir aussi :
getJMSPriority()

clearProperties

```
public void clearProperties() throws JMSException
```

Efface les propriétés du message. Les zones d'en-tête et le corps du message ne sont pas effacés.

Génère :
JMSException - si JMS ne réussit pas à effacer les propriétés JMS du message en raison d'une erreur interne JMS.

propertyExists

```
public boolean propertyExists(java.lang.String name)
                                                throws JMSException
```

Vérifie l'existence d'une valeur de propriété.

Arguments
name - Nom de la propriété à tester.

Renvoie :
La condition TRUE si la propriété existe.

Génère :
JMSException - si JMS ne réussit pas à contrôler l'existence d'une propriété en raison d'une erreur interne JMS.

getBooleanProperty

```
public boolean getBooleanProperty(java.lang.String name)
                                                    throws JMSException
```

Renvoie la valeur de la propriété booléenne correspondant au nom indiqué.

Arguments
name - Nom de la propriété booléenne.

Renvoie :
La valeur de la propriété booléenne correspondant au nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getByteProperty

```
public byte getByteProperty(java.lang.String name)
                                                throws JMSException
```

Renvoie la valeur de la propriété de type BYTE sous le nom indiqué.

Arguments

name - Nom de la propriété de type BYTE.

Renvoie :

La valeur de la propriété de type BYTE sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getShortProperty

```
public short getShortProperty(java.lang.String name)
                                                 throws JMSException
```

Renvoie la valeur de la propriété de type SHORT sous le nom indiqué.

Arguments

name - Nom de la propriété de type SHORT.

Renvoie :

La valeur de la propriété de type SHORT sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getIntProperty

```
public int getIntProperty(java.lang.String name)
                                              throws JMSException
```

Renvoie la valeur de la propriété de type INTEGER sous le nom indiqué.

Arguments

name - Nom de la propriété de type INTEGER.

Renvoie :

La valeur de la propriété de type INTEGER sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getLongProperty

```
public long getLongProperty(java.lang.String name)
                                                throws JMSException
```

Renvoie la valeur de la propriété de type LONG sous le nom indiqué.

Arguments

name - Nom de la propriété de type LONG.

Renvoie :

La valeur de la propriété de type LONG sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getFloatProperty

```
public float getFloatProperty(java.lang.String name)
                                                throws JMSException
```

Renvoie la valeur de la propriété de type FLOAT sous le nom indiqué.

Arguments

name - Nom de la propriété de type FLOAT.

Renvoie :

La valeur de la propriété de type FLOAT sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getDoubleProperty

```
public double getDoubleProperty(java.lang.String name)
                                                 throws JMSException
```

Renvoie la valeur de la propriété de type DOUBLE sous le nom indiqué.

Arguments

name - Nom de la propriété de type DOUBLE.

Renvoie :

La valeur de la propriété de type DOUBLE sous le nom indiqué.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getStringProperty

```
public java.lang.String getStringProperty (java.lang.String name)
                                                              throws JMSException
```

Renvoie la valeur de la propriété de type STRING sous le nom indiqué.

Arguments

name - Nom de la propriété de type STRING.

Renvoie :

La valeur de la propriété de type STRING sous le nom indiqué. Si aucune propriété ne porte ce nom, la valeur NULL est renvoyée.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si cette conversion de type est incorrecte.

getObjectProperty

```
public java.lang.Object getObjectProperty (java.lang.String name)
                                                    throws JMSException
```

Renvoie la valeur de la propriété de type objet Java sous le nom indiqué.

Arguments

name - Nom de la propriété de type objet Java.

Renvoie :

La valeur de la propriété de type objet Java sous le nom indiqué, au format objet (par exemple, si elle est définie au format int, un entier (Integer) est renvoyé). Si aucune propriété ne porte ce nom, la valeur NULL est renvoyée.

Génère :

JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.

getPropertyNames

```
public java.util.Enumeration getPropertyNames()
                                          throws JMSException
```

Renvoie une énumération de tous les noms de propriété.

Renvoie :

Une énumération de tous les noms de propriété.

Génère :

JMSException - si JMS ne réussit pas à extraire les noms de propriété en raison d'une erreur interne JMS.

setBooleanProperty

```
public void setBooleanProperty(java.lang.String name,
                               boolean value) throws JMSException
```

Définit la valeur de la propriété booléenne sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété booléenne.
- value - Valeur de la propriété booléenne à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setByteProperty

```
public void setByteProperty(java.lang.String name,
                            byte value) throws JMSException
```

Définit la valeur de la propriété de type BYTE sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type BYTE.
- value - Valeur de la propriété de type BYTE à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setShortProperty

```
public void setShortProperty(java.lang.String name,
                             short value) throws JMSException
```

Définit la valeur de la propriété de type SHORT sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type SHORT.
- value - Valeur de la propriété de type SHORT à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setIntProperty

```
public void setIntProperty(java.lang.String name,
                           int value) throws JMSException
```

Définit la valeur de la propriété de type INTEGER sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type INTEGER.
- value - Valeur de la propriété de type INTEGER à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setLongProperty

```
public void setLongProperty(java.lang.String name,
                            long value) throws JMSException
```

Définit la valeur de la propriété de type LONG sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type LONG.
- value - Valeur de la propriété de type LONG à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setFloatProperty

```
public void setFloatProperty(java.lang.String name,
                             float value) throws JMSException
```

Définit la valeur de la propriété de type FLOAT sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type FLOAT.
- value - Valeur de la propriété de type FLOAT à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setDoubleProperty

```
public void setDoubleProperty(java.lang.String name,
                              double value) throws JMSException
```

Définit la valeur de la propriété de type DOUBLE sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type DOUBLE.
- value - Valeur de la propriété de type DOUBLE à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setStringProperty

```
public void setStringProperty(java.lang.String name,
                              java.lang.String value) throws JMSException
```

Définit la valeur de la propriété de type STRING sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type STRING.

- value - Valeur de la propriété de type STRING à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à extraire la propriété en raison d'une erreur interne JMS.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

setObjectProperty

```
public void setObjectProperty(java.lang.String name,
                              java.lang.Object value) throws JMSException
```

Définit la valeur de la propriété sous le nom indiqué dans le message.

Arguments

- name - Nom de la propriété de type objet Java.
- value - Valeur de la propriété de type objet Java à définir dans le message.

Génère :

- JMSException - si JMS ne réussit pas à définir la propriété en raison d'une erreur interne JMS.
- MessageFormatException - si l'objet est incorrect.
- MessageNotWriteableException - si les propriétés sont en lecture seule.

acknowledge

```
public void acknowledge() throws JMSException
```

Accuse réception de ce message et de tous les précédents messages reçus par la session.

Génère :

JMSException - si JMS ne réussit pas à émettre un accusé de réception en raison d'une erreur interne JMS.

clearBody

```
public void clearBody() throws JMSException
```

Efface le corps du message. Toutes les autres parties du message restent inchangées.

Génère :

JMSException - si JMS ne réussit pas à effacer le corps du message en raison d'une erreur interne JMS.

MessageConsumer

public interface **MessageConsumer**
Sous-interfaces : **QueueReceiver** et **TopicSubscriber**

Classe MQSeries : **MQMessageConsumer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

Interface parent de tous les clients de messages. Un client utilise un client de message pour recevoir des messages en provenance d'une destination.

Méthodes

getMessageSelector

```
public java.lang.String getMessageSelector()
                                throws JMSException
```

Extrait l'expression du sélecteur de message du client de message.

Renvoie :
le sélecteur de message du client de message.

Génère :
JMSException - si JMS ne parvient pas à extraire le sélecteur de message du fait d'une erreur JMS.

getMessageListener

```
public MessageListener getMessageListener()
                                     throws JMSException
```

Extrait l'objet MessageListener du client de message.

Renvoie :
le programme d'écoute du client de message ou la valeur NULL si aucun programme d'écoute n'est défini.

Génère :
JMSException - si JMS ne parvient pas à extraire le programme d'écoute de message du fait d'une erreur JMS.

Voir aussi :
setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                            throws JMSException
```

Définit l'objet MessageListener du client de message.

Arguments
messageListener - les messages sont distribués à ce programme d'écoute.

Génère :
JMSException - si JMS ne parvient pas à définir le programme d'écoute de message du fait d'une erreur JMS.

Voir aussi :

getMessageListener

receive

```
public Message receive() throws JMSException
```

Reçoit le message suivant produit pour ce client de message.

Renvoie :

le message suivant produit pour ce client de message.

Génère :

JMSException - si JMS ne parvient pas à recevoir le message suivant du fait d'une erreur.

receive

```
public Message receive(long timeOut) throws JMSException
```

Reçoit le message suivant arrivé dans le délai indiqué. Un délai d'expiration de zéro provoque l'attente indéfinie d'un message.

Arguments

timeout - la valeur du délai d'expiration (en millisecondes).

Renvoie :

le message suivant produit pour ce client de message ou la valeur NULL en l'absence de message.

Génère :

JMSException - si JMS ne parvient pas à recevoir le message suivant du fait d'une erreur.

receiveNoWait

```
public Message receiveNoWait() throws JMSException
```

Reçoit le message suivant s'il en existe un de disponible.

Renvoie :

le message suivant produit pour ce client de message ou la valeur NULL en l'absence de message.

Génère :

JMSException - si JMS ne parvient pas à recevoir le message suivant du fait d'une erreur.

close

```
public void close() throws JMSException
```

Du fait qu'un fournisseur est susceptible d'attribuer des ressources en dehors de JVM pour le compte de l'objet MessageConsumer, les clients doivent fermer les ressources lorsqu'elles ne sont plus nécessaires. Vous ne pouvez pas compter sur la récupération de place pour récupérer ces ressources à la fin du traitement, car celle-ci peut survenir tardivement.

Cet appel est bloqué jusqu'à la fin d'une réception ou d'un programme d'écoute en cours.

Génère :

JMSException - si JMS ne parvient pas à fermer le client du fait d'une erreur.

MessageListener

public interface **MessageListener**

MessageListener reçoit les messages transmis en mode de livraison asynchrone.

Méthodes

onMessage

```
public void onMessage(Message message)
```

Transmet un message au programme d'écoute.

Arguments

message - le message transmis au programme d'écoute.

Voir aussi

Session.setMessageListener

MessageProducer

public interface **MessageProducer**
Sous-interfaces : **QueueSender** et **TopicPublisher**

Classe MQSeries : **MQMessageProducer**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

Un client utilise un producteur de messages pour envoyer des messages à une destination.

Constructeurs MQSeries

MQMessageProducer

```
public MQMessageProducer()
```

Méthodes

setDisableMessageID

```
public void setDisableMessageID(boolean value)
                                          throws JMSException
```

Indique si les ID message sont désactivés.

Par défaut, les ID message sont activés.

Remarque : Cette méthode est ignorée dans la mise en oeuvre des classes MQSeries pour Java Message Service.

Arguments
value - indique si les ID message sont désactivés.

Génère :
JMSException - si JMS ne parvient pas à définir l'ID message désactivé du fait d'une erreur interne.

getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException
```

Extrait une indication de l'état des ID message (désactivés ou non).

Renvoie :
une indication de l'état des ID message (désactivés ou non).

Génère :
JMSException - si JMS ne parvient pas à extraire l'ID message désactivé du fait d'une erreur interne.

setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value)
                                                 throws JMSException
```

Indique si les données d'horodatage de message sont désactivées.

Par défaut, les données d'horodatage de message sont activées.

Remarque : Cette méthode est ignorée dans la mise en oeuvre des classes MQSeries pour Java Message Service.

Arguments

value - indique si les données d'horodatage de message sont désactivées.

Génère :

JMSException - si JMS ne parvient pas à définir les données d'horodatage de message désactivées du fait d'une erreur interne.

getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp()
                                        throws JMSException
```

Extrait une indication de l'état des données d'horodatage de message (désactivées.ou non).

Renvoie :

une indication de l'état des ID message (désactivés ou non).

Génère :

JMSException - si JMS ne parvient pas à extraire les données d'horodatage de messages désactivées du fait d'une erreur interne.

setDeliveryMode

```
public void setDeliveryMode(int deliveryMode)
                                         throws JMSException
```

Définit le mode de communication des messages par défaut du producteur.

Le mode de communication est PERSISTENT par défaut.

Arguments

deliveryMode - le mode de communication des messages pour ce producteur de messages.

Génère :

JMSException - si JMS ne parvient pas à définir le mode de communication des messages du fait d'une erreur interne.

Voir aussi :

getDeliveryMode

getDeliveryMode

```
public int getDeliveryMode() throws JMSException
```

Extrait le mode de communication des messages par défaut du producteur.

Renvoie :

le mode de communication des messages pour ce producteur de messages.

Génère :

JMSException - si JMS ne parvient pas à extraire le mode de communication des messages du fait d'une erreur interne.

Voir aussi :

setDeliveryMode

setPriority

```
public void setPriority(int priority) throws JMSException
```

Définit la priorité par défaut du producteur.

Par défaut, le niveau de priorité est 4.

Arguments
priority - la priorité des messages pour ce producteur de messages.

Génère :
JMSException - si JMS ne parvient pas à définir la priorité du fait d'une erreur interne.

Voir aussi :
getPriority

getPriority

```
public int getPriority() throws JMSException
```

Extrait la priorité par défaut du producteur.

Renvoie :
la priorité des messages pour ce producteur de messages.

Génère :
JMSException - si JMS ne parvient pas à extraire la priorité du fait d'une erreur interne.

Voir aussi :
setPriority

setTimeToLive

```
public void setTimeToLive(long timeToLive)
                                    throws JMSException
```

Définit la durée par défaut, en millisecondes, depuis l'heure de distribution, pour la rétention d'un message par le système de messages.

Par défaut, la durée de vie est définie à zéro.

Arguments
timeToLive - la durée de vie d'un message en millisecondes ; zéro correspond à une durée de vie illimitée.

Génère :
JMSException - si JMS ne parvient pas à définir la durée de vie du fait d'une erreur interne.

Voir aussi :
getTimeToLive

getTimeToLive

```
public long getTimeToLive() throws JMSException
```

Extrait la durée par défaut, en millisecondes, depuis l'heure de distribution, pour la rétention d'un message par le système de messages.

Renvoie :
La durée de vie d'un message en millisecondes ; zéro correspond à une durée de vie illimitée.

Génère :
JMSException - si JMS ne parvient pas à extraire la durée de vie du fait d'une erreur interne.

Voir aussi :

setTimeToLive

close

```
public void close() throws JMSException
```

Du fait qu'un fournisseur est susceptible d'attribuer des ressources en dehors de JVM pour le compte d'un MessageProducer, les clients doivent fermer les ressources lorsqu'elles ne sont plus nécessaires. Vous ne pouvez pas compter sur la récupération de place pour récupérer ces ressources à la fin du traitement, car celle-ci peut survenir tardivement.

Génère :

JMSException - si JMS ne parvient pas à fermer le producteur du fait d'une erreur.

MQQueueEnumeration *

public class **MQQueueEnumeration**
extends **Object**
implements **Enumeration**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

Enumération des messages d'une file d'attente. Cette classe n'est pas définie dans la spécification JMS, elle est créée en appelant la méthode getEnumeration de MQQueueBrowser. Cette classe comprend une instance de base de MQQueue pour contenir le curseur d'exploration. La file d'attente est fermée une fois que le curseur a atteint la fin de la file d'attente.

Il est impossible de restaurer une instance de cette classe ; il s'agit d'un mécanisme à ″essai unique″.

Voir aussi : **MQQueueBrowser**

Méthodes

hasMoreElements

```
public boolean hasMoreElements()
```

Indique si un autre message peut être renvoyé.

nextElement

```
public Object nextElement() throws NoSuchElementException
```

Renvoie le message en cours.

Si `hasMoreElements()` renvoie la valeur 'true' (vraie), `nextElement()` renvoie toujours un message. Le message renvoyé peut transmettre sa date d'expiration entre les appels de `hasMoreElements()` et de `nextElement`.

ObjectMessage

public interface **ObjectMessage**
extends **Message**

Classe MQSeries : **JMSObjectMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

Un ObjectMessage permet d'envoyer un message contenant un objet Java sérialisable. Il hérite du Message et y ajoute un corps contenant une référence Java unique. Seuls les objets Java sérialisables peuvent être utilisés.

Voir aussi : **BytesMessage**, **MapMessage**, **Message**, **StreamMessage** et **TextMessage**

Méthodes

setObject

```
public void setObject(java.io.Serializable object)
                                        throws JMSException
```

Définit l'objet sérialisable qui contient les données du message. ObjectMessage contient une image instantanée de l'objet au moment où setObject() est appelé. Les modifications ultérieures de l'objet sont sans effet sur le corps de ObjectMessage.

Arguments
object - données du message.

Génère :

- JMSException - si JMS ne réussit pas à définir l'objet en raison d'une erreur interne JMS.
- MessageFormatException - si la sérialisation de l'objet échoue.
- MessageNotWriteableException - si le message est en mode lecture seule.

getObject

```
public java.io.Serializable getObject()
                              throws JMSException
```

Extrait l'objet sérialisable contenant les données du message. La valeur par défaut est NULL.

Renvoie :
l'objet sérialisable contenant les données du message.

Génère :

- JMSException - si JMS ne réussit pas à extraire l'objet en raison d'une erreur interne JMS.
- MessageFormatException - si la désérialisation de l'objet échoue.

Queue

public interface **Queue**
extends **Destination**
Interfaces secondaires : **TemporaryQueue**

Classe MQSeries : **MQQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

Un objet Queue contient un nom de file d'attente spécifique à un fournisseur. C'est comme cela qu'un client spécifie l'identité d'une file d'attente pour les méthodes JMS.

Constructeurs MQSeries

MQQueue *

```
public MQQueue()
```

Constructeur par défaut devant être utilisé par l'outil d'administration.

MQQueue *

```
public MQQueue(String URIqueue)
```

Crée une nouvelle instance MQQueue. La chaîne nécessite un format URI, comme cela a été décrit à la page 181.

MQQueue *

```
public MQQueue(String queueManagerName,
               String queueName)
```

Méthodes

getQueueName

```
public java.lang.String getQueueName()
                                    throws JMSException
```

Extrait le nom de cette file d'attente.

Les clients dépendant de ce nom ne sont pas transférables.

Renvoie :
le nom de la file d'attente.

Génère :
JMSException - si la mise en oeuvre de JMS pour Queue ne parvient pas à renvoyer le nom de la file d'attente en raison d'une erreur interne.

toString

```
public java.lang.String toString()
```

Renvoie une version imprimée du nom de la file d'attente.

Renvoie :
Les valeurs d'identité spécifiques au fournisseur pour cette file d'attente.

Remplace :
toString dans la classe java.lang.Object

getReference *

```
public Reference getReference() throws NamingException
```

Crée une référence pour cette file d'attente.

Renvoie :
une référence pour cet objet

Génère :
NamingException

setBaseQueueName *

```
public void setBaseQueueName(String x) throws JMSException
```

Définie la valeur du nom de la file d'attente MQSeries.

Remarque : Cette méthode devrait uniquement être utilisée par l'outil d'administration. Elle n'essaie pas de décoder les chaînes au format queue:qmgr:queue.

getBaseQueueName *

```
public String getBaseQueueName()
```

Renvoie :
la valeur du nom de la file d'attente MQSeries.

setBaseQueueManagerName *

```
public void setBaseQueueManagerName(String x) throws JMSException
```

Définit la valeur du nom du gestionnaire de files d'attente MQSeries.

Remarque : Cette méthode devrait uniquement être utilisée par l'outil d'administration.

getBaseQueueManagerName *

```
public String getBaseQueueManagerName()
```

Renvoie :
la valeur du nom du gestionnaire de files d'attente MQSeries.

QueueBrowser

public interface **QueueBrowser**

Classe MQSeries : **MQQueueBrowser**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

Un client utilise un QueueBrowser pour consulter les messages d'une file d'attente sans les supprimer.

Remarque : Le curseur d'exploration se trouve dans la classe MQSeries **MQQueueEnumeration**.

Voir aussi : **QueueReceiver**

Méthodes

getQueue

```
public Queue getQueue() throws JMSException
```

Extrait la file d'attente associée à l'afficheur de file d'attente.

Renvoie :
La file d'attente.

Génère :
JMSException - si JMS ne réussit pas à extraire la file d'attente associée à cet afficheur en raison d'une erreur interne JMS.

getMessageSelector

```
public java.lang.String getMessageSelector() throws JMSException
```

Extrait le sélecteur de message de cet afficheur de file d'attente.

Renvoie :
Le sélecteur de message de cet afficheur de file d'attente.

Génère :
JMSException - si JMS ne réussit pas à extraire le sélecteur de message associé à cet afficheur en raison d'une erreur interne JMS.

getEnumeration

```
public java.util.Enumeration getEnumeration() throws JMSException
```

Extrait une énumération permettant de parcourir les messages de la file d'attente en cours dans leur ordre de réception.

Renvoie :
Une énumération permettant de parcourir les messages.

Génère :
JMSException - si JMS ne réussit pas à extraire l'énumération associée à cet afficheur en raison d'une erreur interne JMS.

Remarque : Si l'afficheur a été créé pour une file d'attente qui n'existe pas, cette absence de file d'attente ne sera détectée que lors du premier appel de getEnumeration.

close

```
public void close() throws JMSException
```

Etant donné qu'un fournisseur peut allouer certaines ressources en dehors de JVM pour le compte d'un QueueBrowser, les clients doivent fermer les ressources lorsqu'ils ne les utilisent pas. Vous ne pouvez pas compter sur une récupération de place pour réutiliser ces ressources car cela risque de prendre trop de temps.

Génère :

JMSException - si JMS ne réussit pas à fermer cet afficheur en raison d'une erreur interne JMS.

QueueConnection

public interface **QueueConnection**
extends **Connection**
Interfaces secondaires : **XAQueueConnection**

Classe MQSeries : **MQQueueConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

Une QueueConnection est une connexion active à un fournisseur point à point JMS. Un client utilise une QueueConnection pour créer une ou plusieurs QueueSessions afin de créer et de traiter des messages.

Voir aussi : **Connection**, **QueueConnectionFactory** et **XAQueueConnection**

Méthodes

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

Crée une QueueSession.

Arguments

- transacted - si cet argument a pour valeur TRUE, la session fait l'objet d'une transaction.
- acknowledgeMode - indique si le consommateur ou le client accusera réception des messages qu'il reçoit. Les valeurs possibles sont les suivantes :
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Cet argument n'est pas pris en compte si la session fait l'objet d'une transaction.

Renvoie :
Une nouvelle session de file d'attente.

Génère :
JMSException - si JMS Connection ne réussit pas à créer une session en raison d'une erreur interne, ou en cas de non prise en charge des transactions spécifiques et du mode accusé de réception.

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                                       (Queue queue,
                                        java.lang.String messageSelector,
                                        ServerSessionPool sessionPool,
                                        int maxMessages)
                                                    throws JMSException
```

Crée un consommateur de connexion pour cette connexion. Il s'agit d'une fonction expert qui n'est pas utilisée par les clients JMS classiques.

Arguments

- queue - file d'attente à laquelle il faut accéder.
- messageSelector - seuls les messages possédant des propriétés correspondant à l'expression du sélecteur de message sont communiqués.
- sessionPool - pool de sessions serveur à associer au consommateur de connexion.
- maxMessages - nombre maximal de messages pouvant être affectés simultanément à une session serveur.

Renvoie :

Le consommateur de connexion.

Génère :

- JMSException - si JMS Connection ne réussit pas à créer un consommateur de connexion en raison d'une erreur interne, ou en cas d'arguments incorrects pour sessionPool et messageSelector.
- InvalidSelectorException - si le sélecteur de message est incorrect.

Voir aussi :

ConnectionConsumer

close *

```
public void close() throws JMSException
```

Remplace :

close dans la classe MQConnection.

QueueConnectionFactory

public interface **QueueConnectionFactory**
extends **ConnectionFactory**
Interfaces secondaires : **XAQueueConnectionFactory**

Classe MQSeries : **MQQueueConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

Un client utilise une interface QueueConnectionFactory pour créer des QueueConnections avec un fournisseur point à point JMS.

Voir aussi : **ConnectionFactory** et **XAQueueConnectionFactory**

Constructeur MQSeries

MQQueueConnectionFactory

```
public MQQueueConnectionFactory()
```

Méthodes

createQueueConnection

```
public QueueConnection createQueueConnection()
                                          throws JMSException
```

Crée une connexion à une file d'attente avec un ID utilisateur par défaut. Cette connexion est créée en mode Arrêt. Aucun message n'est transmis tant que la méthode Connection.start n'a pas été explicitement appelée.

Renvoie :
Une nouvelle connexion de file d'attente.

Génère :

- JMSException - si le fournisseur JMS ne réussit pas à créer une connexion à la file d'attente en raison d'une erreur interne JMS.
- JMSSecurityException - si l'authentification du client échoue en raison d'un nom d'utilisateur et/ou d'un mot de passe incorrect(s).

createQueueConnection

```
public QueueConnection createQueueConnection
                                (java.lang.String userName,
                                 java.lang.String password)
                                                  throws JMSException
```

Crée une connexion à une file d'attente avec l'ID utilisateur spécifié.

Remarque : Cette méthode ne peut être utilisée qu'avec le type de transfert JMSC.MQJMS_TP_CLIENT_MQ_TCPIP (voir ConnectionFactory). La connexion est créée en mode Arrêt. Aucun message n'est transmis tant que la méthode Connection.start n'a pas été explicitement appelée.

Arguments

- userName - nom d'utilisateur de l'appelant.
- password - mot de passe de l'appelant.

Renvoie :

Une nouvelle connexion de file d'attente.

Génère :

- JMSException - si le fournisseur JMS ne réussit pas à créer une connexion à la file d'attente en raison d'une erreur interne JMS.
- JMSSecurityException - si l'authentification du client échoue en raison d'un nom d'utilisateur et/ou d'un mot de passe incorrect(s).

setTemporaryModel *

```
public void setTemporaryModel(String x) throws JMSException
```

getTemporaryModel *

```
public String getTemporaryModel()
```

getReference *

```
public Reference getReference() throws NamingException
```

Crée une référence associée à la fabrique de connexions de file d'attente.

Renvoie :

Une référence pour cet objet.

Génère :

L'exception NamingException.

setMessage Retention*

```
public void setMessageRetention(int x) throws JMSException
```

Définit la méthode associée à l'attribut messageRetention.

Arguments

Les valeurs possibles sont les suivantes :

- JMSC.MQJMS_MRET_YES - les messages inutiles restent dans la file d'attente en entrée.
- JMSC.MQJMS_MRET_NO - les messages inutiles sont traités en fonction de leurs options de destination.

getMessage Retention*

```
public void getMessageRetention()
```

Extrait la méthode associée à l'attribut messageRetention.

Renvoie :

- JMSC.MQJMS_MRET_YES - les messages inutiles restent dans la file d'attente en entrée.
- JMSC.MQJMS_MRET_NO - les messages inutiles sont traités en fonction de leurs options de destination.

QueueReceiver

public interface **QueueReceiver**
extends **MessageConsumer**

Classe MQSeries : **MQQueueReceiver**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

Un client utilise un QueueReceiver pour recevoir des messages placés dans une file d'attente.

Voir aussi : **MessageConsumer**

Cette classe hérite des méthodes suivantes de **MQMessageConsumer** :
- receive
- receiveNoWait
- close
- getMessageListener
- setMessageListener

Méthodes

getQueue

```
public Queue getQueue() throws JMSException
```

Extrait la file d'attente associée au destinataire de la file d'attente.

Renvoie :
la file d'attente.

Génère :
JMSException - si JMS ne parvient pas à extraire la file d'attente pour ce destinataire de file d'attente en raison d'une erreur interne.

QueueRequestor

public class **QueueRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.QueueRequestor
```

JMS fournit cette classe d'aide QueueRequestor pour simplifier les demandes de service. Une session de file d'attente non transactionnelle et une file d'attente cible sont attribuées au constructeur QueueRequestor. Ce dernier crée une file d'attente temporaire (TemporaryQueue) pour les réponses, et il fournit une méthode request() qui envoie le message de demande et attend la réponse. Les utilisateurs peuvent créer des versions plus évoluées.

Voir aussi : **TopicRequestor**

Constructeurs

QueueRequestor

```
public QueueRequestor(QueueSession session,
                      Queue queue)
                              throws JMSException
```

Cette mise en œuvre suppose que l'argument session a pour valeur AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, et qu'il n'est pas transactionnel.

Arguments :

- session - la session de file d'attente à laquelle appartient la file d'attente.
- queue - la file d'attente dans laquelle le message de demande/réponse doit être traité.

Génère :

JMSException - si une erreur JMS se produit.

Méthodes

requête

```
public Message requête(Message message)
                                    throws JMSException
```

Envoie une demande et attend la réponse. La file d'attente temporaire est utilisée pour répondre, et une seule réponse par demande est attendue.

Arguments :

message - le message à envoyer.

Renvoie :

le message de réponse.

Génère :

JMSException - si une erreur JMS se produit.

close

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM pour le compte d'un QueueRequestor, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

Remarque : Cette méthode permet de fermer l'objet Session transmis au constructeur QueueRequestor.

Génère :

JMSException - si une erreur JMS se produit.

QueueSender

public interface **QueueSender**
extends **MessageProducer**

Classe MQSeries : **MQQueueSender**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

Un client utilise un QueueSender pour envoyer des messages dans une file d'attente.

Un QueueSender est normalement associé à une file d'attente spécifique. Toutefois, il est possible de créer un QueueSender non identifié qui n'est pas associé à une file d'attente définie.

Voir aussi : **MessageProducer**

Méthodes

getQueue

```
public Queue getQueue() throws JMSException
```

Extrait la file d'attente associée à l'émetteur de la file d'attente.

Renvoie :
la file d'attente.

Génère :
JMSException - si JMS ne parvient pas à extraire la file d'attente pour cet émetteur de file d'attente en raison d'une erreur interne.

send

```
public void send(Message message) throws JMSException
```

Envoie un message dans la file d'attente. Utilise le mode de livraison, la durée de vie et la priorité par défaut de QueueSender.

Arguments :
message - le message à envoyer.

Génère :

- JMSException - si JMS ne parvient pas à envoyer le message en raison d'une erreur.
- MessageFormatException - si un message incorrect est spécifié.
- InvalidDestinationException - si un client utilise cette méthode avec un émetteur de file d'attente associé à une file d'attente non valide.

send

```
public void send(Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Envoie vers la file d'attente un message spécifiant le mode de livraison, la priorité, et la durée de vie du message.

Arguments :

- message - le message à envoyer.
- deliveryMode - le mode de distribution à utiliser.
- priority - la priorité du message.
- timeToLive - la durée de vie du message (en millisecondes).

Génère :

- JMSException - si JMS ne parvient pas à envoyer le message en raison d'une erreur interne.
- MessageFormatException - si un message incorrect est spécifié.
- InvalidDestinationException - si un client utilise cette méthode avec un émetteur de file d'attente associé à une file d'attente non valide.

send

```
public void send(Queue queue,
                 Message message) throws JMSException
```

Envoie un message à la file d'attente spécifiée avec le mode de livraison, la durée de vie et la priorité par défaut de QueueSender.

Remarque : Cette méthode ne peut être utilisée qu'avec des QueueSenders non identifiés.

Arguments :

- queue - la file d'attente vers laquelle ce message doit être envoyé.
- message - le message à envoyer.

Génère :

- JMSException - si JMS ne parvient pas à envoyer le message en raison d'une erreur interne.
- MessageFormatException - si un message incorrect est spécifié.
- InvalidDestinationException - si un client utilise cette méthode avec une file d'attente non valide.

send

```
public void send(Queue queue,
                 Message message,
                 int deliveryMode,
                 int priority,
                 long timeToLive) throws JMSException
```

Envoie un message à la file d'attente spécifiée avec le mode de livraison, la priorité et la durée de vie.

Remarque : Cette méthode ne peut être utilisée qu'avec des QueueSenders non identifiés.

Arguments :

- queue - la file d'attente vers laquelle ce message doit être envoyé.

- message - le message à envoyer.
- deliveryMode - le mode de distribution à utiliser.
- priority - la priorité du message.
- timeToLive - la durée de vie du message (en millisecondes).

Génère :

- JMSException - si JMS ne parvient pas à envoyer le message en raison d'une erreur interne.
- MessageFormatException - si un message incorrect est spécifié.
- InvalidDestinationException - si un client utilise cette méthode avec une file d'attente non valide.

close *

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM pour le compte d'un QueueSender, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

Génère :

JMSException si JMS ne parvient pas à fermer le producteur en raison d'une erreur.

Remplace :

close dans la classe MQMessageProducer.

QueueSession

public interface **QueueSession**
extends **Session**

Classe MQSeries : **MQQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

Une QueueSession fournit des méthodes de création de QueueReceivers, QueueSenders, QueueBrowsers et TemporaryQueues.

Voir aussi : **Session**

Les méthodes suivantes sont héritées de **MQSession** :
- close
- commit
- rollback
- recover

Méthodes

createQueue

```
public Queue createQueue(java.lang.String queueName)
                                                throws JMSException
```

Crée une file d'attente dotée d'un nom de file d'attente. Cela permet la création d'une file d'attente avec le nom spécifique d'un fournisseur. La chaîne nécessite un format URI, comme décrit page 181.

Remarque : Les clients dépendant de cette capacité ne sont pas transférables.

Arguments :
queueName - le nom de la file d'attente.

Renvoie :
une file d'attente portant le nom indiqué.

Génère :
JMSException - si une session ne parvient pas à créer une file d'attente en raison d'une erreur JMS.

createReceiver

```
public QueueReceiver createReceiver(Queue queue)
                                                throws JMSException
```

Crée un QueueReceiver pour recevoir des messages de la file d'attente spécifiée.

Arguments :
queue - la file d'attente à laquelle il faut accéder.

Génère :

- JMSException - si une session ne parvient pas à créer un réceptionnaire en raison d'une erreur JMS.
- InvalidDestinationException - si une file d'attente incorrecte est spécifiée.

createReceiver

```
public QueueReceiver createReceiver(Queue queue,
                                    java.lang.String messageSelector)
                                    throws JMSException
```

Crée un QueueReceiver pour recevoir des messages de la file d'attente spécifiée.

Arguments :

- queue - la file d'attente à laquelle il faut accéder.
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis.

Génère :

- JMSException - si une session ne parvient pas à créer un réceptionnaire en raison d'une erreur JMS.
- InvalidDestinationException - si une file d'attente incorrecte est spécifiée.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

createSender

```
public QueueSender createSender(Queue queue)
                                        throws JMSException
```

Crée un QueueSender pour envoyer des messages vers la file d'attente spécifiée.

Arguments :

queue - la file d'attente à laquelle il faut accéder, ou la valeur NULL s'il s'agit d'un producteur non identifié.

Génère :

- JMSException - si une session ne parvient pas à créer un émetteur en raison d'une erreur JMS.
- InvalidDestinationException - si une file d'attente incorrecte est spécifiée.

createBrowser

```
public QueueBrowser createBrowser(Queue queue)
                                      throws JMSException
```

Crée un QueueBrowser pour jeter un coup d'oeil aux messages de la file d'attente spécifiée.

Arguments :

queue - la file d'attente à laquelle il faut accéder.

Génère :

- JMSException - si une session ne parvient pas à créer un navigateur en raison d'une erreur JMS.

- InvalidDestinationException - si une file d'attente incorrecte est spécifiée.

createBrowser

```
public QueueBrowser createBrowser(Queue queue,
                                  java.lang.String messageSelector)
                                  throws JMSException
```

Crée un QueueBrowser pour jeter un coup d'oeil aux messages de la file d'attente spécifiée.

Arguments :

- queue - la file d'attente à laquelle il faut accéder.
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis.

Génère :

- JMSException - si une session ne parvient pas à créer un navigateur en raison d'une erreur JMS.
- InvalidDestinationException - si une file d'attente incorrecte est spécifiée.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue()
                                        throws JMSException
```

Crée une file d'attente temporaire. Sa durée de vie sera celle de la QueueConnection à moins qu'elle ne soit supprimée avant.

Renvoie :

une file d'attente temporaire.

Génère :

JMSException - si une session ne parvient pas à créer une file d'attente temporaire en raison d'une erreur JMS.

Session

public interface **Session**
extends **java.lang.Runnable**
Interfaces secondaires : **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** et **XATopicSession**

Classe MQSeries : **MQSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

Une session JMS est un contexte unique conçu avec des unités d'exécution pour produire et traiter des messages.

Voir aussi : **QueueSession**, **TopicSession**, **XAQueueSession**, **XASession** et **XATopicSession**

Zones

AUTO_ACKNOWLEDGE

```
public static final int AUTO_ACKNOWLEDGE
```

Avec ce mode d'accusé de réception, la session accuse automatiquement réception d'un message lorsqu'un appel de réception a abouti, ou lorsque le programme d'écoute des messages appelé pour traiter le message aboutit.

CLIENT_ACKNOWLEDGE

```
public static final int CLIENT_ACKNOWLEDGE
```

Avec ce mode d'accusé de réception, le client accuse réception d'un message en appelant une méthode d'accusé de réception de message.

DUPS_OK_ACKNOWLEDGE

```
public static final int DUPS_OK_ACKNOWLEDGE
```

Ce mode d'accusé de réception indique à la session d'accuser réception dela livraison des messages de façon indolente.

Méthodes

createBytesMessage

```
public BytesMessage createBytesMessage()
                                throws JMSException
```

Crée un BytesMessage. Un BytesMessage est utilisé pour envoyer un message contenant une chaîne d'octets non interprétés.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createMapMessage

```
public MapMessage createMapMessage() throws JMSException
```

Crée un MapMessage. Un MapMessage est utilisé pour envoyer des paires nom-valeur auto-définies, dans lesquelles les noms sont des chaînes et les valeurs sont de type Java primitive.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createMessage

```
public Message createMessage() throws JMSException
```

Crée un Message. L'interface Message est l'interface racine de tous les messages JMS. Elle contient toutes les informations d'en-tête de message standard. Elle peut être envoyée lorsqu'un message comprenant uniquement des informations d'en-tête est suffisant.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createObjectMessage

```
public ObjectMessage createObjectMessage()
                              throws JMSException
```

Crée un ObjectMessage. Un ObjectMessage est utilisé pour envoyer un message qui comprend un objet Java sérialisable.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createObjectMessage

```
public ObjectMessage createObjectMessage
                             (java.io.Serializable object)
                              throws JMSException
```

Crée un ObjectMessage initialisé. Un ObjectMessage est utilisé pour envoyer un message qui comprend un objet Java sérialisable.

Arguments :

object - l'objet à utiliser pour initialiser ce message.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createStreamMessage

```
public StreamMessage createStreamMessage()
                              throws JMSException
```

Crée un StreamMessage. Un StreamMessage permet d'envoyer un flot de Java primitives auto-défini.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

Session

createTextMessage

```
public TextMessage createTextMessage() throws JMSException
```

Crée un TextMessage. Un TextMessage est utilisé pour envoyer un message contenant une Chaîne.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

createTextMessage

```
public TextMessage createTextMessage
                        (java.lang.String string)
                        throws JMSException
```

Crée un TextMessage initialisé. Un TextMessage est utilisé pour envoyer un message contenant une Chaîne.

Arguments :

string - la chaîne utilisée pour initialiser ce message.

Génère :

JMSException - si JMS ne parvient pas à créer le message du fait d'une erreur interne.

getTransacted

```
public boolean getTransacted() throws JMSException
```

Est-ce que la session est en mode transactionnel ?

Renvoie :

la valeur true (vrai) si la session est en mode transactionnel.

Génère :

JMSException - si JMS ne parvient pas à renvoyer le mode transactionnel du fait d'une erreur interne du fournisseur JMS.

commit

```
public void commit() throws JMSException
```

Valide tous les messages réalisés au cours de cette transaction et libère tous les verrous en cours.

Génère :

- JMSException - si JMS ne parvient pas à valider la transaction du fait d'une erreur interne.
- TransactionRolledBackException - si la transaction est annulée suite à une erreur interne au cours de la validation.

rollback

```
public void rollback() throws JMSException
```

Annule tous les messages réalisés au cours de cette transaction et libère tous les verrous en cours.

Génère :

JMSException - si JMS ne parvient pas à annuler la transactiondu fait d'une erreur interne.

close

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM pour le compte d'une Session, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

La fermeture d'une session transactionnelle annule toutes les transactions en cours. La fermeture d'une session ferme automatiquement ses producteurs et ses consommateurs de messages, il n'est donc pas nécessaire de les fermer individuellement.

Génère :

JMSException - si JMS ne parvient pas à fermer la session du fait d'une erreur interne

recover

```
public void recover() throws JMSException
```

Arrête la livraison des messages dans cette session, et redémarre l'envoi des messages avec le message le plus ancien n'ayant pas fait l'objet d'un accusé de réception.

Génère :

JMSException - si JMS ne parvient pas à interrompre la livraison des messages et à redémarrer l'envoi des messages du fait d'une erreur interne.

getMessageListener

```
public MessageListener getMessageListener()
                                  throws JMSException
```

Renvoie le programme d'écoute des messages de la session.

Renvoie :

le programme d'écoute des messages associé à cette session.

Génère :

JMSException - si JMS ne parvient pas à utiliser le programme d'écoute des messages du fait d'une erreur interne du fournisseur JMS.

Voir aussi :

setMessageListener

setMessageListener

```
public void setMessageListener(MessageListener listener)
                                                 throws JMSException
```

Définit le programme d'écoute des messages de la session. Lorsque cette méthode est définie, aucune autre forme de réception des messages dans cette session ne peut être utilisée. Toutefois, toutes les formes d'envoi des messages sont toujours prises en charge.

Il s'agit une fonction expert qui n'est pas utilisée par les clients JMS classiques.

Arguments :

listener - le programme d'écoute des messages à associer à cette session.

Génère :

JMSException - si JMS ne parvient pas à définir le programme d'écoute des messages du fait d'une erreur interne du fournisseur JMS.

Voir aussi :

getMessageListener, ServerSessionPool, ServerSession

run

```
public void run()
```

Cette méthode est uniquement prévue pour être utilisée par des serveurs d'application.

Spécifié par :

run dans l'interface java.lang.Runnable

Voir aussi :

ServerSession

StreamMessage

public interface **StreamMessage**
extends **Message**

Classe MQSeries : **JMSStreamMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

StreamMessage est utilisé pour envoyer un flot de ″Java primitives″.

Voir aussi : **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** et **TextMessage**

Méthodes

readBoolean

```
public boolean readBoolean() throws JMSException
```

Lit une valeur booléenne dans le message de flot.

Renvoie :
La valeur booléenne lue.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readByte

```
public byte readByte() throws JMSException
```

Lit une valeur de type BYTE (octet) dans le message de flot.

Renvoie :
L'octet suivant du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readShort

```
public short readShort() throws JMSException
```

Lit un nombre de 16 bits dans le message de flot.

Renvoie :

Un nombre de 16 bits du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readChar

```
public char readChar() throws JMSException
```

Lit une valeur de type UNICODE CHARACTER (caractère unicode) dans le message de flot.

Renvoie :

Un caractère unicode du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readInt

```
public int readInt() throws JMSException
```

Lit un entier de 32 bits dans le message de flot.

Renvoie :

Un entier de 32 bits du message de flot, interprété comme une valeur de type INT.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readLong

```
public long readLong() throws JMSException
```

Lit un entier de 64 bits dans le message de flot.

Renvoie :

Un entier de 64 bits du message de flot, interprété comme une valeur de type LONG.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readFloat

```
public float readFloat() throws JMSException
```

Lit une valeur de type FLOAT (variable flottante) dans le message de flot.

Renvoie :

Une valeur de type FLOAT du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readDouble

```
public double readDouble() throws JMSException
```

Lit une valeur de type DOUBLE dans le message de flot.

Renvoie :

Une valeur de type DOUBLE du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readString

```
public java.lang.String readString() throws JMSException
```

Lit une valeur de type STRING dans le message de flot.

Renvoie :

Une chaîne unicode du message de flot.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readBytes

```
public int readBytes(byte[] value)
             throws JMSExceptioneam message.
```

Lit une zone de tableau d'octets dans le message de flot qui se trouve dans l'objet byte[] (octet) spécifié (mémoire tampon en lecture). Si la taille de la mémoire tampon est inférieure ou égale à celle des données de la zone de message, une application doit effectuer des appels supplémentaires de cette méthode pour extraire les données restantes. Une fois le premier appel de la méthode readBytes effectué sur la valeur d'une zone byte[], la totalité de la valeur de cette zone doit être lue pour que la lecture de la zone suivante puisse être effectuée. En cas de tentative de lecture de la zone suivante avant la fin de la lecture de la zone en cours, une exception MessageFormatException est envoyée.

Arguments

value - mémoire tampon dans laquelle les données sont lues.

Renvoie :

Le nombre total d'octets lus dans la mémoire tampon, ou -1 s'il ne reste plus aucune donnée à lire car la fin de la zone de type BYTE a été atteinte.

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- MessageFormatException - si cette conversion de type est incorrecte.
- MessageNotReadableException - si le message est en mode écriture seule.

readObject

```
public java.lang.Object readObject() throws JMSException
```

Lit un objet Java dans le message de flot.

Renvoie :

Un objet Java du message de flot, au format objet (par exemple, en cas de définition au format int, un entier (Integer) est renvoyé).

Génère :

- JMSException - si JMS ne réussit pas à lire le message en raison d'une erreur interne JMS.
- MessageEOFException - en cas de réception d'une fin de flot de message.
- NotReadableException - si le message est en mode écriture seule.

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException
```

Ecrit une valeur booléenne dans le message de flot.

Arguments

value - valeur booléenne à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeByte

```
public void writeByte(byte value) throws JMSException
```

Ecrit un octet dans le message de flot.

Arguments

value - valeur de type BYTE (octet) à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeShort

```
public void writeShort(short value) throws JMSException
```

Ecrit une valeur de type SHORT dans le message de flot.

Arguments

value - valeur de type SHORT à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeChar

```
public void writeChar(char value) throws JMSException
```

Ecrit une valeur de type CHAR dans le message de flot.

Arguments

value - valeur de type CHAR à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeInt

```
public void writeInt(int value) throws JMSException
```

Ecrit une valeur de type INT dans le message de flot.

Arguments

value - valeur de type INT à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeLong

```
public void writeLong(long value) throws JMSException
```

Ecrit une valeur de type LONG dans le message de flot.

Arguments

value - valeur de type LONG à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeFloat

```
public void writeFloat(float value) throws JMSException
```

Ecrit une valeur de type FLOAT (variable flottante) dans le message de flot.

Arguments

value - valeur de type FLOAT à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeDouble

```
public void writeDouble(double value) throws JMSException
```

Ecrit une valeur de type DOUBLE dans le message de flot.

Arguments

value - valeur de type DOUBLE à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeString

```
public void writeString(java.lang.String value)
                                          throws JMSException
```

Ecrit une valeur de type STRING (chaîne) dans le message de flot.

Arguments

value - valeur de type STRING à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeBytes

```
public void writeBytes(byte[] value) throws JMSException
```

Ecrit un tableau d'octets dans le message de flot.

Arguments

value - tableau d'octets à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeBytes

```
public void writeBytes(byte[] value,
                       int offset,
                       int length) throws JMSException
```

Ecrit une portion de tableau d'octets dans le message de flot.

Arguments

- value - valeur du tableau d'octets à écrire.
- offset - décalage initial dans le tableau d'octets.
- length - nombre d'octets à utiliser.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

writeObject

```
public void writeObject(java.lang.Object value)
                                         throws JMSException
```

Ecrit un objet Java dans le message de flot. Cette méthode ne fonctionne qu'avec les types primitifs d'objet (Integer, Double, Long, par exemple), les chaînes et les tableaux d'octets.

Arguments

value - objet Java à écrire.

Génère :

- JMSException - si JMS ne réussit pas à écrire le message en raison d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.
- MessageFormatException - si l'objet est incorrect.

reset

```
public void reset() throws JMSException
```

Place le message en mode lecture seule et replace le flot à son début.

Génère :

- JMSException - si JMS ne réussit pas à réinitialiser le message en raison d'une erreur interne JMS.
- MessageFormatException - si le format du message est incorrect.

TemporaryQueue

public interface **TemporaryQueue**
extends **Queue**

Classe MQSeries : **MQTemporaryQueue**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

Un objet TemporaryQueue est un objet Queue unique créé pour la durée d'une connexion QueueConnection.

Méthodes

delete

```
public void delete() throws JMSException
```

Supprime cette file d'attente temporaire. S'il existe toujours des émetteurs ou des réceptionnaires qui l'utilisent, une exception JMSException sera générée.

Génère :

JMSException - si JMS ne parvient pas à supprimer une TemporaryQueue en raison d'une erreur interne.

TemporaryTopic

public interface **TemporaryTopic**
extends **Topic**

Classe MQSeries : **MQTemporaryTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

Un TemporaryTopic est un objet Topic unique créé pour la durée d'une connexion TopicConnection et il peut uniquement être utilisé par les consommateurs de cette connexion.

Constructeur QSeries

MQTemporaryTopic

```
MQTemporaryTopic() throws JMSException
```

Méthodes

delete

```
public void delete() throws JMSException
```

Supprime cette rubrique temporaire. S'il existe toujours des diffuseurs de publication ou des souscripteurs qui l'utilisent, une exception JMSException sera générée.

Génère :

JMSException - si JMS ne parvient pas à supprimer un TemporaryTopic du fait d'une erreur interne.

TextMessage

public interface **TextMessage**
extends **Message**

Classe MQSeries : **JMSTextMessage**

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

Un TextMessage permet d'envoyer un message contenant un java.lang.String. Il hérite d'un message et y ajoute un texte de corps de message.

Voir aussi : **BytesMessage**, **MapMessage**, **Message**, **ObjectMessage** et **StreamMessage**

Méthodes

setText

```
public void setText(java.lang.String string)
                                          throws JMSException
```

Définit la chaîne contenant les données de ce message.

Arguments :

string - la chaîne contenant les données du message.

Génère :

- JMSException - si JMS ne parvient pas à définir le texte du fait d'une erreur interne JMS.
- MessageNotWriteableException - si le message est en mode lecture seule.

getText

```
public java.lang.String getText() throws JMSException
```

Extrait la chaîne contenant les données de ce message. La valeur par défaut est NULL.

Renvoie :

la chaîne contenant les données du message.

Génère :

JMSException - si JMS ne parvient pas à extraire le texte du fait d'une erreur interne JMS.

Topic

public interface **Topic**
extends **Destination**
Interfaces secondaires : **TemporaryTopic**

Classe MQSeries : **MQTopic**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

Un objet Topic contient un nom de rubrique spécifique à un fournisseur. C'est comme cela qu'un client spécifie l'identité d'une rubrique pour les méthodes JMS.

Voir aussi : **Destination**

Constructeur QSeries

MQTopic

```
public MQTopic()
public MQTopic(string URItopic)
```

Voir **TopicSession.createTopic**.

Méthodes

getTopicName

```
public java.lang.String getTopicName() throws JMSException
```

Extrait le nom de cette rubrique au format URI. (Le format URI est décrit dans la section «Création de rubriques en phase d'exécution» à la page 191.)

Remarque : Les clients dépendant de ce nom ne sont pas transférables.

Renvoie :
le nom de la rubrique.

Génère :
JMSException - si la mise en oeuvre de JMS pour Topic ne parvient pas à renvoyer le nom de la rubrique suite à une erreur interne.

toString

```
public String toString()
```

Renvoie une version imprimée du nom de la rubrique.

Renvoie :
les valeurs d'identité spécifiques du fournisseur pour cette rubrique.

Remplace :
toString dans la classes Object.

getReference *

```
public Reference getReference()
```

Crée une référence pour cette rubrique.

Renvoie :
une référence pour cet objet.

Génère :
NamingException.

setBaseTopicName *

```
public void setBaseTopicName(String x)
```

définit la méthode pour le nom de la rubrique MQSeries sous-jacente.

getBaseTopicName *

```
public String getBaseTopicName()
```

extrait la méthode pour le nom de la rubrique MQSeries sous-jacente.

setBrokerDurSubQueue *

```
public void setBrokerDurSubQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerDurSubQueue.

Arguments :
brokerDurSubQueue - le nom de la file d'attente des souscriptions de longue durée à utiliser.

getBrokerDurSubQueue *

```
public String getBrokerDurSubQueue()
```

Extrait la méthode pour l'attribut brokerDurSubQueue.

Renvoie :
le nom de la file d'attente des souscriptions de longue durée (le brokerDurSubQueue) à utiliser.

setBrokerCCDurSubQueue *

```
public void setBrokerCCDurSubQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerCCDurSubQueue.

Arguments :
brokerCCDurSubQueue - le nom de la file d'attente des souscriptions de longue durée à utiliser pour un ConnectionConsumer.

getBrokerCCDurSubQueue *

```
public String getBrokerCCDurSubQueue()
```

Extrait la méthode pour l'attribut brokerCCDurSubQueue.

Renvoie :
le nom de la file d'attente des souscriptions de longue durée (le brokerCCDurSubQueue) à utiliser pour un ConnectionConsumer.

TopicConnection

public interface **TopicConnection**
extends **Connection**
Interfaces secondaires : **XATopicConnection**

Classe MQSeries : **MQTopicConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

Une TopicConnection est une connexion active avec un fournisseur JMS de publication/souscription.

Voir aussi : **Connection**, **TopicConnectionFactory** et **XATopicConnection**

Méthodes

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                       throws JMSException
```

Crée une TopicSession.

Arguments :

- transacted - si la valeur est true, la session est transactionnelle.
- acknowledgeMode - une des valeurs suivantes :
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indique si le consommateur ou le client accusera réception des messages qu'il recevra. Ce paramètre est ignoré si la session est transactionnelle.

Renvoie :
une session de rubrique nouvellement créée.

Génère :
JMSException - si JMS ne parvient pas à créer une session en raison d'une erreur interne, ou à cause d'un manque de prise en charge de la transaction particulière et du mode d'accusé de réception.

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer
                            (Topic topic,
                             java.lang.String messageSelector,
                             ServerSessionPool sessionPool,
                             int maxMessages)
                                      throws JMSException
```

Crée un consommateur de connexion pour cette connexion. Il s'agit d'une fonction expert qui n'est pas utilisée par les clients JMS classiques.

Arguments :

- topic - la rubrique à laquelle il faut accéder.
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis.
- sessionPool - le pool de sessions serveur à associer à ce consommateur de connexion.
- maxMessages - le nombre maximal de messages qui peuvent être attribués simultanément à une session serveur.

Renvoie :

le consommateur de la connexion.

Génère :

- JMSException - si JMS ne parvient pas à créer un consommateur de connexion en raison d'une erreur interne, ou à cause d'arguments incorrects pour SessionPool.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

Voir aussi :

ConnectionConsumer

createDurableConnectionConsumer

```
public ConnectionConsumer createDurableConnectionConsumer
                                (Topic topic,
                                 java.lang.String subscriptionName
                                 java.lang.String messageSelector,
                                 ServerSessionPool sessionPool,
                                 int maxMessages)
                                          throws JMSException
```

Crée un consommateur de connexion durable pour cette connexion. Il s'agit d'une fonction expert qui n'est pas utilisée par les clients JMS classiques.

Arguments :

- topic - la rubrique à laquelle il faut accéder.
- subscriptionName - nom de la souscription de longue durée.
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis.
- sessionPool - le pool de sessions serveur à associer à ce consommateur de connexion de longue durée.
- maxMessages - le nombre maximal de messages qui peuvent être attribués simultanément à une session serveur.

Renvoie :

le consommateur de connexion de longue durée.

Génère :

- JMSException - si JMS ne parvient pas à créer un consommateur de connexion en raison d'une erreur interne, ou à cause d'arguments incorrects pour SessionPool et messageSelector.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

TopicConnection

Voir aussi :

ConnectionConsumer

TopicConnectionFactory

public interface **TopicConnectionFactory**
extends **ConnectionFactory**
Interfaces secondaires : **XATopicConnectionFactory**

Classe MQSeries : **MQTopicConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

Un client utilise TopicConnectionFactory pour créer des TopicConnections avec un fournisseur JMS de publication/souscription.

Voir aussi : **ConnectionFactory** et **XATopicConnectionFactory**

Constructeur MQSeries

MQTopicConnectionFactory

```
public MQTopicConnectionFactory()
```

Méthodes

createTopicConnection

```
public TopicConnection createTopicConnection()
                                        throws JMSException
```

Crée une connexion de rubrique avec un ID utilisateur par défaut. La connexion est créée en mode arrêt. Aucun message ne sera livré avant que la méthode Connection.start soit explicitement appelée.

Renvoie :
une connexion de rubrique nouvellement créée.

Génère :

- JMSException - si le fournisseur JMS ne parvient pas à créer une connexion de rubrique en raison d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

createTopicConnection

```
public TopicConnection createTopicConnection
                                        (java.lang.String userName,
                                        java.lang.String password)
                                        throws JMSException
```

Crée une connexion de rubrique avec un ID utilisateur spécifié. La connexion est créée en mode arrêt. Aucun message ne sera livré avant que la méthode Connection.start soit explicitement appelée.

Remarque : Cette méthode est uniquement valide pour le type de transfert IBM_JMS_TP_CLIENT_MQ_TCPIP. Voir ConnectionFactory.

Arguments :

- userName - le nom d'utilisateur de l'appelant
- password - le mot de passe de l'appelant

Renvoie :

une connexion de rubrique nouvellement créée.

Génère :

- JMSException - si le fournisseur JMS ne parvient pas à créer une connexion de rubrique en raison d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

setBrokerControlQueue *

```
public void setBrokerControlQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerControlQueue.

Arguments :

brokerControlQueue - le nom de la file d'attente de contrôle du courtier.

getBrokerControlQueue *

```
public String getBrokerControlQueue()
```

Extrait la méthode pour l'attribut brokerControlQueue.

Renvoie :

le nom de la file d'attente de contrôle du courtier.

setBrokerQueueManager *

```
public void setBrokerQueueManager(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerQueueManager.

Arguments :

brokerQueueManager - le nom du gestionnaire de files d'attente du courtier.

getBrokerQueueManager *

```
public String getBrokerQueueManager()
```

Extrait la méthode pour l'attribut brokerQueueManager.

Renvoie :

le nom du gestionnaire de files d'attente du courtier.

setBrokerPubQueue *

```
public void setBrokerPubQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerPubQueue.

Arguments :

brokerPubQueue - le nom de la file d'attente de publication du courtier.

getBrokerPubQueue *

```
public String getBrokerPubQueue()
```

Extrait la méthode pour l'attribut brokerPubQueue.

Renvoie :
le nom de de la file d'attente de publication du courtier.

setBrokerSubQueue *

```
public void setBrokerSubQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerSubQueue.

Arguments :
brokerSubQueue - le nom de la file d'attente des souscriptions temporaires à utiliser.

getBrokerSubQueue *

```
public String getBrokerSubQueue()
```

Extrait la méthode pour l'attribut brokerSubQueue.

Renvoie :
le nom de la file d'attente des souscriptions temporaires à utiliser.

setBrokerCCSubQueue *

```
public void setBrokerCCSubQueue(String x) throws JMSException
```

Définit la méthode pour l'attribut brokerCCSubQueue.

Arguments :
brokerSubQueue - le nom de la file d'attente des souscriptions temporaires à utiliser pour un ConnectionConsumer.

getBrokerCCSubQueue *

```
public String getBrokerCCSubQueue()
```

Extrait la méthode pour l'attribut brokerCCSubQueue.

Renvoie :
le nom de la file d'attente des souscriptions temporaires à utiliser pour un ConnectionConsumer.

setBrokerVersion *

```
public void setBrokerVersion(int x) throws JMSException
```

Définit la méthode pour l'attribut brokerVersion.

Arguments :
brokerVersion - le numéro de version du courtier.

getBrokerVersion *

```
public int getBrokerVersion()
```

Extrait la méthode pour l'attribut brokerVersion.

Renvoie :
le numéro de version du courtier.

getReference *

```
public Reference getReference()
```

Renvoie une référence pour cette fabrique de connexion de rubriques.

Renvoie :
une référence pour cette fabrique de connexion de rubriques.

Génère :
NamingException.

TopicPublisher

public interface **TopicPublisher**
extends **MessageProducer**

Classe MQSeries : **MQTopicPublisher**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

TopicPublisher permet à un client de publier les messages d'une rubrique. Il s'agit de la variante publication/souscription d'un producteur de messages JMS.

Méthodes

getTopic

```
public Topic getTopic() throws JMSException
```

Extrait la rubrique associée au diffuseur de publications.

Renvoie :
la rubrique du diffuseur de publications

Génère :
JMSException - si JMS ne parvient pas à extraire la rubrique du diffuseur de publications du fait d'une erreur interne.

publish

```
public void publish(Message message) throws JMSException
```

Publie un message sur la rubrique. Utilise le mode de distribution, la durée de vie et la priorité par défaut de la rubrique.

Arguments
message - le message à publier.

Génère :

- JMSException - si JMS ne parvient pas à publier le message du fait d'une erreur interne.
- MessageFormatException - si le message spécifié n'est pas valide.
- InvalidDestinationException - si un client utilise cette méthode avec un diffuseur de publications associé à une rubrique non valide.

publish

```
public void publish(Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Publie un message sur la rubrique en indiquant son mode de distribution, sa priorité et sa durée de vie.

Arguments

- message - le message à publier.
- deliveryMode - le mode de distribution à utiliser.
- priority - la priorité du message.
- timeToLive - la durée de vie du message (en millisecondes).

Génère :

- JMSException - si JMS ne parvient pas à publier le message du fait d'une erreur interne.
- MessageFormatException - si le message spécifié n'est pas valide.
- InvalidDestinationException - si un client utilise cette méthode avec un diffuseur de publications associé à une rubrique non valide.

publish

```
public void publish(Topic topic,
                    Message message) throws JMSException
```

Publie un message sur une rubrique pour un producteur de messages non identifié. Utilise le mode de distribution, la durée de vie et la priorité par défaut de la rubrique.

Arguments

- topic - la rubrique dans laquelle publier ce message.
- message - le message à envoyer.

Génère :

- JMSException - si JMS ne parvient pas à publier le message du fait d'une erreur interne.
- MessageFormatException - si le message spécifié n'est pas valide.
- InvalidDestinationException - si un client utilise cette méthode avec une rubrique non valide.

publish

```
public void publish(Topic topic,
                    Message message,
                    int deliveryMode,
                    int priority,
                    long timeToLive) throws JMSException
```

Publie un message sur une rubrique pour un producteur de messages non identifié, en indiquant son mode de distribution, sa priorité et sa durée de vie.

Arguments

- topic - la rubrique dans laquelle publier ce message.
- message - le message à envoyer.
- deliveryMode - le mode de distribution à utiliser.
- priority - la priorité du message.
- timeToLive - la durée de vie du message (en millisecondes).

Génère :

- JMSException - si JMS ne parvient pas à publier le message du fait d'une erreur interne.
- MessageFormatException - si le message spécifié n'est pas valide.
- InvalidDestinationException - si un client utilise cette méthode avec une rubrique non valide.

close *

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM pour le compte d'un TopicPublisher, les clients doivent fermer les ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

Génère :

JMSException - si JMS ne parvient pas à fermer le producteur du fait d'une erreur.

Remplace :

close dans la classe MQMessageProducer.

TopicRequestor

public class **TopicRequestor**
extends **java.lang.Object**

```
java.lang.Object
    |
    +----javax.jms.TopicRequestor
```

TopicRequestor vous permet de formuler des demandes de maintenance.

Un TopicSession non transactionnel et un Topic cible sont attribués au constructeur TopicRequestor. Ce dernier crée un TemporaryTopic pour les réponses et fournit une méthode request() qui envoie le message de demande et attend la réponse. Les utilisateurs peuvent créer des versions plus évoluées.

Constructeurs

TopicRequestor

```
public TopicRequestor(TopicSession session,
                      Topic topic) throws JMSException
```

Constructeur de la classe TopicRequestor. Cette mise en œuvre suppose que l'argument session a pour valeur AUTO_ACKNOWLEDGE ou DUPS_OK_ACKNOWLEDGE, et qu'il n'est pas transactionnel.

Arguments

- session - la session à laquelle appartient la rubrique.
- topic - la rubrique faisant l'objet de la demande.

Génère :

JMSException - si une erreur JMS se produit.

Méthodes

request

```
public Message request(Message message) throws JMSException
```

Envoie une demande et attend la réponse.

Arguments

message - le message à envoyer.

Renvoie :

le message de réponse.

Génère :

JMSException - si une erreur JMS se produit.

close

```
public void close() throws JMSException
```

Le fournisseur pouvant allouer certaines ressources sans passer par JVM pour le compte d'un TopicRequestor, les clients doivent fermer les

ressources qu'ils n'utilisent pas. N'attendez pas la récupération d'espace mémoire pour restaurer les ressources car elle risque de ne pas avoir lieu suffisamment tôt.

Remarque : Cette méthode permet de fermer l'objet Session transmis au constructeur TopicRequestor.

Génère :

JMSException - si une erreur JMS se produit.

TopicSession

public interface **TopicSession**
extends **Session**

Classe MQSeries : **MQTopicSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

TopicSession fournit des méthodes de création de TopicPublishers, TopicSubscribers et TemporaryTopics.

Voir aussi : **Session**

Constructeur MQSeries

MQTopicSession

```
public MQTopicSession(boolean transacted,
                      int acknowledgeMode) throws JMSException
```

Voir **TopicConnection.createTopicSession**.

Méthodes

createTopic

```
public Topic createTopic(java.lang.String topicName)
                                                  throws JMSException
```

Crée une rubrique avec un nom de rubrique au format URI. (Le format URI est décrit dans la section «Création de rubriques en phase d'exécution» à la page 191.) Cela permet la création d'une rubrique avec le nom spécifique d'un fournisseur.

Remarque : Les clients dépendant de cette capacité ne sont pas transférables.

Arguments :
topicName - le nom de cette rubrique.

Renvoie :
une rubrique portant le nom indiqué.

Génère :
JMSException - si une session ne parvient pas à créer une rubrique en raison d'une erreur JMS.

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic)
                                        throws JMSException
```

Crée un souscripteur temporaire dans la rubrique spécifiée.

Arguments :
topic - la rubrique à laquelle il faut souscrire

Génère :

- JMSException - si une session ne parvient pas à créer un souscripteur en raison d'une erreur JMS.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.

createSubscriber

```
public TopicSubscriber createSubscriber
                            (Topic topic,
                             java.lang.String messageSelector,
                             boolean noLocal) throws JMSException
```

Crée un souscripteur temporaire dans la rubrique spécifiée.

Arguments :

- topic - la rubrique à laquelle il faut souscrire
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis. Cette valeur peut être NULL.
- noLocal - Si cet argument est défini, la livraison des messages publiés par la connexion locale est bloquée.

Génère :

- JMSException - si une session ne parvient pas à créer un souscripteur en raison d'une erreur JMS ou d'un sélecteur incorrect.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name) throws JMSException
```

Crée un souscripteur durable dans la rubrique spécifiée. Un client peut modifier une souscription de longue durée existante en créant un souscripteur durable avec le même nom et une nouvelle rubrique et/ou un nouveau sélecteur de messages.

Arguments :

- topic - la rubrique à laquelle il faut souscrire.
- name - le nom utilisé pour identifier cette souscription.

Génère :

- JMSException - si une session ne parvient pas à créer un souscripteur en raison d'une erreur JMS.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.

Voir **TopicSession.unsubscribe**

TopicSession

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber
                                    (Topic topic,
                                     java.lang.String name,
                                     java.lang.String messageSelector,
                                     boolean noLocal)  throws JMSException
```

Crée un souscripteur durable dans la rubrique spécifiée.

Arguments :

- topic - la rubrique à laquelle il faut souscrire.
- name - le nom utilisé pour identifier cette souscription.
- messageSelector - seuls les messages dont les propriétés correspondent à l'expression du sélecteur de messages sont transmis. Cette valeur peut être NULL.
- noLocal - Si cet argument est défini, la livraison des messages publiés par la connexion locale est bloquée.

Génère :

- JMSException - si une session ne parvient pas à créer un souscripteur en raison d'une erreur JMS ou d'un sélecteur incorrect.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.
- InvalidSelectorException - si le sélecteur de messages est incorrect.

createPublisher

```
public TopicPublisher createPublisher(Topic topic)
                                    throws JMSException
```

Crée un diffuseur de publications pour la rubrique spécifiée.

Arguments :

topic - la rubrique dans laquelle les publications seront diffusées, ou la valeur NULL s'il s'agit d'un producteur non identifié.

Génère :

- JMSException - si une session ne parvient pas à créer un souscripteur en raison d'une erreur JMS.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic()
                                     throws JMSException
```

Crée une rubrique temporaire. Sa durée de vie sera celle de la TopicConnection à moins qu'elle ne soit supprimée avant.

Renvoie :

une rubrique temporaire.

Génère :

JMSException - si une session ne parvient pas à créer une rubrique temporaire en raison d'une erreur JMS.

unsubscribe

```
public void unsubscribe(java.lang.String name)
                                        throws JMSException
```

Annule une souscription de longue durée qui a été créée par un client.

Remarque : N'utilisez pas cette méthode tant qu'il existe une souscription active. Vous devez d'abord fermer votre souscripteur à l'aide de la méthode close().

Arguments :

name - le nom utilisé pour identifier cette souscription.

Génère :

- JMSException - si JMS ne parvient pas à mettre fin à la souscription durable en raison d'une erreur JMS.
- InvalidDestinationException - si une rubrique incorrecte est spécifiée.

TopicSubscriber

public interface **TopicSubscriber**
extends **MessageConsumer**

Classe MQSeries : **MQTopicSubscriber**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

TopicSubscriber permet à un client de recevoir les messages publiés sur une rubrique. Il s'agit de la variante publication/souscription d'un client de messages JMS.

Voir aussi : **MessageConsumer** et **TopicSession.createSubscriber**

MQTopicSubscriber hérite des méthodes de MQMessageConsumer ci-dessous :
close
getMessageListener
receive
receiveNoWait
setMessageListener

Méthodes

getTopic

```
public Topic getTopic() throws JMSException
```

Extrait la rubrique associée à ce souscripteur.

Renvoie :
la rubrique de ce souscripteur.

Génère :
JMSException - si JMS ne parvient pas à extraire la rubrique du souscripteur du fait d'une erreur interne.

getNoLocal

```
public boolean getNoLocal() throws JMSException
```

Extrait l'attribut NoLocal pour TopicSubscriber. La valeur par défaut de cet attribut est false.

Renvoie :
la valeur true si les messages publiés localement sont bloqués.

Génère :
JMSException - si JMS ne parvient pas à extraire l'attribut NoLocal pour le souscripteur de la rubrique du fait d'une erreur interne.

XAConnection

public interface **XAConnection**
Interfaces secondaires : **QueueConnectionFactory** et**TopicConnectionFactory**

Classe MQSeries : **MQXAConnection**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnection
```

XAConnection accroît la capacité de Connection grâce à XASession. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

Voir aussi : **XAQueueConnection** et **XATopicConnection**

XAConnectionFactory

public interface **XAConnectionFactory**
Interfaces secondaires : **XAQueueConnectionFactory** et **XATopicConnectionFactory**

Classe MQSeries : **MQXAConnectionFactory**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXAConnectionFactory
```

Certains serveurs d'applications offrent un support pour l'utilisation des ressources JTS dans les transactions distribuées. Pour inclure les transactions JMS dans une transaction JTS, le serveur d'applications exige un fournisseur JMS compatible avec JTS. Le support JTS des fournisseurs JMS est disponible via XAConnectionFactory, que les serveurs d'applications utilisent pour créer des sessions XA. Les objets XAConnectionFactory sont des objets gérés par JMS tout comme les objets ConnectionFactory. Les serveurs d'applications utilisent JNDI pour les chercher.

Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA/XA avec WebSphere» à la page 371.

Voir aussi : **XAQueueConnectionFactory** et **XATopicConnectionFactory**

XAQueueConnection

public interface **XAQueueConnection**
extends **QueueConnection** and **XAConnection**

Classe MQSeries : **MQXAQueueConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQQueueConnection
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnection
```

XATopicConnection offre les mêmes options de création que QueueConnection. La seule différence tient au fait qu'une connexion XA est, par définition, transactionnelle. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

Voir aussi : **XAConnection** et **QueueConnection**

Méthodes

createXAQueueSession

```
public XAQueueSession createXAQueueSession()
```

Crée une session de file d'attente XA (XAQueueSession).

Génère :

JMSException - si JMS ne parvient pas à créer une session de file d'attente XA du fait d'une erreur interne.

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
                                                throws JMSException
```

Crée une session de file d'attente (QueueSession).

Arguments

- transacted - si la valeur est true, la session est transactionnelle.
- acknowledgeMode - indique si le consommateur ou le client accusera réception des messages qu'il recevra. Les valeurs possibles sont les suivantes :
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Cet argument est ignoré si la session est transactionnelle.

Renvoie :

une nouvelle session de file d'attente (il ne s'agit pas d'une session de file d'attente XA).

Génère :

JMSException - si JMS ne parvient pas à créer une session de file d'attente du fait d'une erreur interne.

XAQueueConnectionFactory

public interface **XAQueueConnectionFactory**
extends **QueueConnectionFactory** and **XAConnectionFactory**

Classe MQSeries : **MQXAQueueConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQQueueConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

XAQueueConnectionFactory offre les mêmes options de création que QueueConnectionFactory. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

Voir aussi : **QueueConnectionFactory** et **XAConnectionFactory**

Méthodes

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection()
                                              throws JMSException
```

Crée une connexion de file d'attente XA avec un ID utilisateur par défaut. La connexion est créée en mode arrêt. Les messages ne sont pas distribués avant l'appel explicite de la méthode Connection.start.

Renvoie :
une nouvelle connexion de file d'attente XA.

Génère :

- JMSException - si JMS ne parvient pas à créer une connexion de file d'attente XA du fait d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection
                                    (java.lang.String userName,
                                     java.lang.String password)
                                                  throws JMSException
```

Crée une connexion de file d'attente XA avec un ID utilisateur spécifique. La connexion est créée en mode arrêt. Les messages ne sont pas distribués avant l'appel explicite de la méthode Connection.start.

Arguments

- userName - le nom d'utilisateur de l'appelant
- password - le mot de passe de l'appelant

Renvoie :
une nouvelle connexion de file d'attente XA.

Génère :

- JMSException - si JMS ne parvient pas à créer une connexion de file d'attente XA du fait d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

XAQueueSession

public interface **XAQueueSession**
extends **XASession**

Classe MQSeries : **MQXAQueueSession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
           |
           +----com.ibm.mq.jms.MQXAQueueSession
```

XAQueueSession fournit une session de file d'attente classique permettant de créer des objets QueueReceiver, QueueSender et QueueBrowser. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

La ressource XA qui correspond à la session de file d'attente peut être obtenue en appelant la méthode getXAResource, héritée de XASession.

Voir aussi : **XASession**

Méthodes

getQueueSession

```
public QueueSession  getQueueSession()
                                 throws JMSException
```

Extrait la session de file d'attente associée à la file d'attente XA.

Renvoie :
l'objet de la session de file d'attente.

Génère :
JMSException - si une erreur JMS se produit.

XASession

public interface **XASession**
extends **Session**
Interfaces secondaires : **XAQueueSession** et **XATopicSession**

Classe MQSeries : **MQXASession**

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQXASession
```

XASession accroît la capacité de Session en ajoutant un accès au support JTA du fournisseur JMS. Ce support prend la forme d'un objet javax.transaction.xa.XAResource. La fonctionnalité de cet objet ressemble étroitement à celle définie dans l'interface standard X/Open XA Resource.

Les serveurs d'applications contrôlent l'affectation transactionnelle d'une session XA grâce à la ressource XA correspondante. Ils utilisent la ressource XA pour affecter la session à une transaction, préparer et valider les transactions, etc.

Une ressource XA fournit certaines fonctions relativement évoluées permettant de travailler simultanément sur plusieurs transactions et d'obtenir la liste des transactions en cours.

Un fournisseur JMS compatible avec JTA doit totalement mettre en oeuvre cette fonctionnalité. Pour ce faire, le fournisseur JMS peut utiliser les fonctions d'une base de données prenant en charge XA ou implémenter cette fonctionnalité de toutes pièces.

Une session JMS classique est attribuée au client du serveur d'applications. En arrière-plan, le serveur d'applications contrôle la gestion des transactions de la session XA sous-jacente.

Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA/XA avec WebSphere» à la page 371.

Voir aussi : **XAQueueSession** et **XATopicSession**

Méthodes

getXAResource

```
public javax.transaction.xa.XAResource getXAResource()
```

Renvoie une ressource XA à l'appelant.

Renvoie :
une ressource XA à l'appelant.

getTransacted

```
public boolean getTransacted()
                        throws JMSException
```

Renvoie toujours la valeur true.

Spécifié par :
getTransacted dans l'interface Session.

Renvoie :
true - si la session est transactionnelle.

Génère :
JMSException - si JMS ne parvient pas à renvoyer le mode transactionnel du fait d'une erreur interne du fournisseur JMS.

commit

```
public void commit()
            throws JMSException
```

Cette méthode ne doit pas être appelée pour un objet XASession. Si elle est appelée, une exception TransactionInProgressException est générée.

Spécifié par :
commit dans l'interface Session.

Génère :
TransactionInProgressException - si cette méthode est appelée pour un objet XASession.

rollback

```
public void rollback()
              throws JMSException
```

Cette méthode ne doit pas être appelée pour un objet XASession. Si elle est appelée, une exception TransactionInProgressException est générée.

Spécifié par :
rollback dans l'interface Session.

Génère :
TransactionInProgressException - si cette méthode est appelée pour un objet XASession.

XATopicConnection

public interface **XATopicConnection**
extends **TopicConnection** and **XAConnection**

Classe MQSeries : **MQXATopicConnection**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnection
            |
            +----com.ibm.mq.jms.MQTopicConnection
                    |
                    +----com.ibm.mq.jms.MQXATopicConnection
```

XATopicConnection offre les mêmes options de création que TopicConnection. La seule différence tient au fait qu'une connexion XA est, par définition, transactionnelle. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

Voir aussi : **TopicConnection** et **XAConnection**

Méthodes

createXATopicSession

```
public XATopicSession createXATopicSession()
                                        throws JMSException
```

Crée une session de rubrique XA (XATopicSession).

Génère :

JMSException - si JMS Connection ne parvient pas à créer la session de rubrique XA du fait d'une erreur interne.

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
                                 throws JMSException
```

Crée une session de rubrique (TopicSession).

Spécifié par :

createTopicSession dans l'interface TopicConnection.

Arguments

- transacted - si la valeur est true, la session est transactionnelle.
- acknowledgeMode - une des valeurs suivantes :
  Session.AUTO_ACKNOWLEDGE
  Session.CLIENT_ACKNOWLEDGE
  Session.DUPS_OK_ACKNOWLEDGE

  Indique si le consommateur ou le client accusera réception des messages qu'il recevra. Cet argument est ignoré si la session est transactionnelle.

Renvoie :

une nouvelle session de rubrique (il ne s'agit pas d'une session de rubrique XA).

Génère :

JMSException - si JMS Connection ne parvient pas à créer une session de rubrique en raison d'une erreur interne.

XATopicConnectionFactory

public interface **XATopicConnectionFactory**
extends **TopicConnectionFactory** and **XAConnectionFactory**

Classe MQSeries : **MQXATopicConnectionFactory**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQConnectionFactory
            |
            +----com.ibm.mq.jms.MQTopicConnectionFactory
                    |
                    +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

XATopicConnectionFactory offre les mêmes options de création que TopicConnectionFactory. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA/XA avec WebSphere» à la page 371.

Voir aussi : **TopicConnectionFactory** et **XAConnectionFactory**

Méthodes

createXATopicConnection

```
public XATopicConnection createXATopicConnection()
                                        throws JMSException
```

Crée une connexion de rubrique XA à l'aide de l'ID utilisateur par défaut. La connexion est créée en mode arrêt. Les messages ne sont pas distribués avant l'appel explicite de la méthode Connection.start.

Renvoie :
une nouvelle connexion de rubrique XA.

Génère :

- JMSException - si JMS Provider ne parvient pas à créer une connexion de rubrique XA du fait d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

createXATopicConnection

```
public XATopicConnection createXATopicConnection(java.lang.String userName,
                                                 java.lang.String password)
                                        throws JMSException
```

Crée une connexion de rubrique XA à l'aide de l'ID utilisateur spécifié. La connexion est créée en mode arrêt. Les messages ne sont pas distribués avant l'appel explicite de la méthode Connection.start.

Arguments

- userName - le nom d'utilisateur de l'appelant
- password - le mot de passe de l'appelant

Renvoie :
une nouvelle connexion de rubrique XA.

Génère :

- JMSException - si JMS Provider ne parvient pas à créer une connexion de rubrique XA du fait d'une erreur interne.
- JMSSecurityException - si l'authentification du client échoue en cas de saisie d'un nom d'utilisateur ou d'un mot de passe non valide.

XATopicSession

public interface **XATopicSession**
extends **XASession**

Classe MQSeries : **MQXATopicSession**

```
java.lang.Object
    |
    +----com.ibm.mq.jms.MQXASession
            |
            +----com.ibm.mq.jms.MQXATopicSession
```

XATopicSession fournit une interface TopicSession permettant de créer des objets TopicSubscriber et TopicPublisher. Pour plus d'informations sur la façon dont MQ JMS utilise les classes XA, reportez-vous à la section «Annexe E. Interface JMS JTA / XA avec WebSphere» à la page 371.

La ressource XA (XAResource) qui correspond à la session de rubrique (TopicSession) peut être obtenue en appelant la méthode getXAResource, héritée de XASession.

Voir aussi : **TopicSession** et **XASession**

Méthodes

getTopicSession

```
public TopicSession getTopicSession()
                                throws JMSException
```

Extrait la session de rubrique associée à XATopicSession.

Renvoie :
l'objet de la session de rubrique.

Génère :

- JMSException - si une erreur JMS se produit.

Partie 4. Annexes

Annexe A. Correspondance entre les propriétés de l'outil d'administration et les propriétés programmables

Les classes MQSeries pour Java Message Service (JMS) permettent de définir et de rechercher les propriétés des objets administrés via l'outil d'administration MQ JMS ou dans un programme d'application. Le tableau 30 présente la correspondance entre chaque nom de propriété utilisé par l'outil d'administration et la variable membre à laquelle il fait référence. Il indique également la correspondance entre les valeurs de propriétés symboliques utilisées par l'outil et leurs équivalents au niveau du programme.

Tableau 30. Comparaison entre les propriétés de l'outil d'administration et leurs équivalents au niveau du programme

Propriété	**Variable membre**	**Correspondance entre propriétés**	
		Outil	**Programme**
DESCRIPTION	description		
TRANSPORT	transportType	• BIND • CLIENT	JMSC.MQJMS_TP_BINDINGS_MQ JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
CLIENTID	clientId		
QMANAGER	queueManager*		
HOSTNAME	hostName		
PORT	port		
CHANNEL	channel		
CCSID	CCSID		
RECEXIT	receiveExit		
RECEXITINIT	receiveExitInit		
SECEXIT	securityExit		
SECEXITINIT	securityExitInit		
SENDEXIT	sendExit		
SENDEXITINIT	sendExitInit		
TEMPMODEL	temporaryModel		
MSGRETENTION	messageRetention	• YES • NO	JMSC.MQJMS_MRET_YES JMSC.MQJMS_MRET_NO
BROKERVER	brokerVersion	• V1	JMSC.MQJMS_BROKER_V1
BROKERPUBQ	brokerPubQueue		
BROKERSUBQ	brokerSubQueue		
BROKERDURSUBQ	brokerDurSubQueue		
BROKERCCSUBQ	brokerCCSubQueue		
BROKERCCDSUBQ	brokerCCDurSubQueue		
BROKERQMGR	brokerQueueManager		
BROKERCONQ	brokerControlQueue		
EXPIRY	expiry	• APP • UNLIM	JMSC.MQJMS_EXP_APP JMSC.MQJMS_EXP_UNLIMITED

Propriétés

Tableau 30. Comparaison entre les propriétés de l'outil d'administration et leurs équivalents au niveau du programme (suite)

Propriété	**Variable membre**	**Correspondance entre propriétés**	
		Outil	**Programme**
PRIORITY	priority	• APP • QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
PERSISTENCE	persistence	• APP • QDEF • PERS • NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
TARGCLIENT	targetClient	• JMS • MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
ENCODING	encoding		
QUEUE	baseQueueName		
TOPIC	baseTopicName		
Remarque : * Pour un objet MQQueue, le nom de la variable membre est baseQueueManagerName.			

Annexe B. Scripts fournis avec les classes MQSeries pour Java Message Service (JMS)

Vous trouverez les fichiers suivants dans le répertoire d'installation `bin` de MQ JMS. Ces scripts vous sont fournis afin de vous aider dans l'exécution des tâches courantes nécessaires lors de l'installation ou de l'utilisation de MQ JMS. Le tableau 31 dresse la liste des scripts et de leur utilisation.

Tableau 31. Utilitaires fournis avec les classes MQSeries pour Java Message Service

Utilitaire	**Utilisation**
IVTRun.bat IVTTidy.bat IVTSetup.bat	Permet d'exécuter le programme de vérification d'installation du modèle point à point (voir la section «Exécution du programme IVT point à point» à la page 23).
PSIVTRun.bat	Permet d'exécuter le programme de vérification d'installation du modèle publication/souscription (voir la section «Test PSIVT» à la page 27).
formatLog.bat	Permet de convertir les fichiers journaux binaires en texte simple (voir la section «Consignation» à la page 31).
JMSAdmin.bat	Permet d'exécuter l'outil d'administration (voir la section «Chapitre 5. Utilisation de l'outil d'administration MQ JMS» à la page 33).
JMSAdmin.config	Fichier de configuration de l'outil d'administration (voir la section «Configuration» à la page 34).
runjms.bat	Script d'aide à l'exécution des applications JMS (voir la section «Exécution de vos propres programmes MQ JMS» à la page 30).
PSReportDump.class	Permet de visualiser les messages d'état du courtier (voir la section «Traitement des rapports envoyés par le courtier» à la page 197).
Remarque : Sous UNIX, l'extension .bat est omise pour les noms de fichiers.	

Scripts

Annexe C. Configuration du serveur LDAP pour les objets Java

Si vous utilisez JNDI pour stocker des objets administrés par MQ JMS et que vous utilisez un serveur LDAP comme fournisseur de services JNDI, le serveur doit être de type LDAP v3 (par exemple, SecureWay eNetwork Directory v3.1) et doit être configuré pour le stockage d'objets Java.

Vérification de la configuration du serveur LDAP

Pour savoir si le serveur LDAP est déjà configuré pour prendre en charge des objets Java, exécutez l'outil d'administration MQ JMS en mode LDAP (voir la section «Appel de l'outil d'administration» à la page 33).

Créez et affichez un objet de test en tapant les commandes suivantes :

```
DEFINE QCF(ldapTest)
DISPLAY QCF(ldapTest)
```

Si aucune exception n'est renvoyée, cela signifie que le serveur est correctement configuré et que vous pouvez procéder au stockage des objets JMS.

Si l'exception 'SchemaViolationException' est renvoyée ou si le message ″Impossible de lier l'objet″ s'affiche, le serveur n'est pas correctement configuré. Soit le serveur n'est pas configuré pour stocker des objets Java, soit les autorisations d'accès aux objets ou les suffixes ne sont pas corrects. Suivez les procédures indiquées ci-après pour configurer le serveur.

Procédures de configuration

Nombreux sont les serveurs LDAP qui fournissent des outils d'administration du serveur. Pour savoir comment les utiliser, reportez-vous à la documentation fournie avec le serveur. Ces outils doivent vous permettre de visualiser et de modifier le schéma qui contient les définitions 'attribute' et 'objectclass'.

Assurez-vous que le schéma contient les définitions de classes d'objet suivantes, et ajoutez-les si nécessaire :

```
( 1.3.6.1.4.1.42.2.27.4.2.1
  NAME 'javaContainer'
  DESC 'Conteneur objet Java'
  SUP top
  STRUCTURAL
  MUST ( cn )
)
( 1.3.6.1.4.1.42.2.27.4.2.4
  NAME 'javaObject'
  DESC 'Représentation objet Java'
  SUP top
  ABSTRACT
  MUST ( javaClassName )
  MAY ( javaClassNames $
        javaCodebase $
        javaDoc $
        description )
)
```

```
( 1.3.6.1.4.1.42.2.27.4.2.5
  NAME 'javaSerializedObject'
  DESC 'Objet sérialisé Java'
  SUP javaObject
  AUXILIARY
  MUST ( javaSerializedData )
)
( 1.3.6.1.4.1.42.2.27.4.2.7
  NAME 'javaNamingReference'
  DESC 'Référence JNDI'
  SUP javaObject
  AUXILIARY
  MAY ( javaReferenceAddress $
        javaFactory )
)
```

Assurez-vous aussi que le schéma contient les définitions d'attributs suivantes, et ajoutez-les si nécessaire :

```
( 1.3.6.1.4.1.42.2.27.4.1.11
        NAME 'javaReferenceAddress'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.10
        NAME 'javaFactory'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

( 1.3.6.1.4.1.42.2.27.4.1.7
        NAME 'javaCodebase'
        SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )
```

Une fois que vous avez terminé les modifications, arrêtez puis redémarrez le serveur LDAP, et relancez la procédure de vérification de la configuration du serveur décrite à la section «Vérification de la configuration du serveur LDAP» à la page 367.

Annexe D. Connexion à MQSeries Integrator V2

Vous pouvez utiliser MQSeries Integrator V2 :

- comme courtier de publication/souscription pour MQ JMS, ou
- pour acheminer ou transformer des messages créés par une application client JMS, et pour envoyer ou publier des messages à destination d'un client JMS

Publication/souscription

Vous pouvez utiliser MQSeries Integrator V2 comme courtier de publication/souscription pour MQ JMS. Vous devez pour cela exécuter les opérations de configuration suivantes :

- Base MQSeries

  Vous devez tout d'abord créer une file d'attente de publication de courtier. Il s'agit d'une file d'attente MQSeries dépendant du gestionnaire de files d'attente du courtier. Elle permet de soumettre des publications au courtier. Vous pouvez lui affecter le nom de votre choix, mais ce nom doit correspondre au nom de file d'attente indiqué par la propriété BROKERPUBQ de TopicConnectionFactory. Par défaut, la valeur SYSTEM.BROKER.DEFAULT.STREAM est affectée à la propriété BROKERPUBQ de TopicConnectionFactory. Aussi, si vous ne définissez pas un autre nom dans TopicConnectionFactory, vous devez nommer la file d'attente SYSTEM.BROKER.DEFAULT.STREAM.

- MQSeries Integrator V2

  Vous devez ensuite configurer un *flux de messages* dans un groupe d'exécution associé au courtier. Le rôle de ce flux de messages consiste à lire les messages provenant de la file d'attente de publication du courtier. (Vous pouvez, si vous le souhaitez, configurer plusieurs files d'attente de publication, chacune devant posséder son propre TopicConnectionFactory et son propre flux de messages.)

  Un flux de messages de base se compose d'un noeud MQInput (configuré pour lire les messages provenant de la file d'attente SYSTEM.BROKER.DEFAULT.STREAM) dont le flux de sortie est connecté au flux d'entrée d'un noeud de publication (ou MQOutput).

  Le diagramme du flux de messages se présente de la manière suivante :

Figure 7. Flux de messages MQSeries Integrator

Une fois le flux de messages déployé et le courtier démarré, du point de vue de l'application JMS, le courtier MQSeries Integrator V2 se comporte comme un courtier de publication/souscription MQSeries. L'état de souscription en cours peut être visualisé à partir du Centre de contrôle MQSeries Integrator.

Remarques :

1. Aucune modification des classes MQSeries pour Java Message Service n'est requise.

2. Le courtier de publication/souscription MQSeries et le courtier MQSeries Integrator V2 ne peuvent coexister sur un même gestionnaire de files d'attente.
3. Pour plus d'informations sur l'installation et la configuration de MQSeries Integrator V2, reportez-vous au manuel *MQSeries Integrator pour Windows NT Version 2.0 - Guide d'installation*.

Transformation et acheminement

Vous pouvez utiliser MQSeries Integrator V2 pour acheminer ou transformer les messages créés par une application client JMS, et pour envoyer ou publier des messages à destination d'un client JMS.

MQSeries JMS utilise le dossier mcd de MQRFH2 pour transporter les informations relatives aux messages (voir la section «En-tête MQRFH2» à la page 204). Par défaut, la propriété Message Domain (Msd) est utilisée pour identifier la nature du message (texte, octets, flot de données, mappe ou objet).

Lorsqu'une application JMS crée un message de type texte ou octets, elle peut ne pas tenir compte de cette propriété Msd et définir d'autres zones de dossier Mcd à la place. Elle effectue cette opération en définissant une propriété JMS Type possédant un format URI spécial, par exemple :

`mcd://domain/set/type[?format=fmt]`

Les valeurs des zones *domain*, *set*, *type* et *fmt* (*fmt* étant facultatif) sont copiées dans le MQRFH2 en sortie. Cela signifie que l'application peut affecter à ces zones des valeurs reconnues par un flux de messages MQSeries Integrator V2.

Annexe E. Interface JMS JTA/XA avec WebSphere

Les classes MQSeries pour Java Message Service incluent les interfaces JMS XA. Ces dernières permettent à MQ JMS de participer à une validation en deux phases coordonnée par un gestionnaire de transactions conforme à Java Transaction API (JTA).

La présente section décrit comment utiliser ces fonctions avec WebSphere Application Server, Advanced Edition, afin que WebSphere puisse coordonner les opérations JMS d'envoi et de réception, ainsi que les mises à jour de base de données, dans une transaction globale.

Avant d'utiliser MQ JMS et les classes XA avec WebSphere, vous devrez peut-être procéder à des opérations d'installation et de configuration supplémentaires. Reportez-vous au fichier `Readme.txt` de la page Web MQSeries - Utilisation de Java SupportPac pour obtenir les dernières informations (`www.ibm.com/software/ts/mqseries/txppacs/ma88.html`).

Utilisation de l'interface JMS avec WebSphere

La présente section indique comment utiliser l'interface JMS avec WebSphere Application Server, Advanced Edition.

Vous devez déjà avoir une bonne connaissance des programmes JMS, MQSeries, ainsi que des beans EJB. Les caractéristiques de base figurent dans les spécifications JMS, EJB V2 (disponibles auprès de Sun), dans le présent manuel, les exemples fournis avec MQ JMS, et les autres manuels se rapportant à MQSeries et WebSphere.

Objets gérés

JMS utilise des objets administrés pour encapsuler des informations propres au fournisseur. Cela réduit l'impact des caractéristiques propres au fournisseur sur les applications utilisateur final. Les objets administrés sont stockés dans un espace annuaire JNDI et peuvent être être extraits et utilisés de manière transférable sans avoir connaissance du contenu propre au fournisseur.

Pour une utilisation en autonome, MQ JMS fournit les classes suivantes :

- MQQueueConnectionFactory
- MQQueue
- MQTopicConnectionFactory
- MQTopic

WebSphere fournit deux objets administrés supplémentaires, afin que MQ JMS puisse s'intégrer à WebSphere :

- JMSWrapXAQueueConnectionFactory
- JMSWrapXATopicConnectionFactory

Vous utilisez ces objets exactement de la même manière que MQQueueConnectionFactory et MQTopicConnectionFactory. Néanmoins, ils utilisent les versions XA des classes JMS et engagent MQ XAResource dans la transaction WebSphere.

Gestion par conteneur contre gestion par bean

Les transactions gérées par conteneur sont des transactions dans des beans EJB délimitées automatiquement par le conteneur EJB. Les transactions gérées par bean sont des transactions dans des beans EJB délimitées par le programme (via l'interface UserTransaction).

Validation en deux phases contre optimisation en une phase

Le coordinateur WebSphere n'appelle une véritable validation en deux phases que si plusieurs ressources XAResource sont utilisées dans une transaction donnée. Les transactions qui impliquent une seule ressource sont validées à l'aide de l'optimisation en une phase. Cela supprime dans une large mesure le besoin d'utiliser divers objets ConnectionFactories pour des transactions réparties et non réparties.

Définition d'objets administrés

Vous pouvez utiliser l'outil d'administration MQ JMS pour définir les fabriques de connexion propres à WebSphere et les stocker dans un espace annuaire JNDI. Le fichier admin.config situé dans *MQ_install_dir*/bin doit comporter les lignes suivantes :

```
INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
PROVIDER_URL=iiop://hostname/
```

MQ_install_dir est le répertoire d'installation de MQ JMS et *hostname* le nom ou l'adresse IP de la machine qui exécute WebSphere.

Pour accéder à com.ibm.ejs.ns.jndi.CNInitialContextFactory, vous devez ajouter le fichier ejs.jar du répertoire lib WebSphere à CLASSPATH.

Pour créer les nouvelles fabriques, utilisez l'instruction define avec les deux types nouveaux suivants :

```
def WSQCF(name) [properties]
def WSTCF(name) [properties]
```

Ces nouveaux types utilisent les mêmes propriétés que les types QCF ou TCF équivalents, à la différence que seul le type de transport BIND est admis (les propriétés client ne peuvent donc pas être configurées). Pour plus d'informations, reportez-vous à la section «Administration des objets JMS» à la page 37.

Extraction d'objets d'administration

Dans un bean EJB, vous extrayez les objets administrés par JMS à l'aide de la méthode InitialContext.lookup(), par exemple :

```
InitialContext ic = new InitialContext();
TopicConnectionFactory tcf = (TopicConnectionFactory) ic.lookup("jms/Samples/TCF1");
```

Les objets peuvent être rattachés aux interfaces JMS génériques et utilisés en tant que tels. Normalement, il n'est pas nécessaire de programmer vers les classes spécifiques MQSeries du code de l'application.

Exemples :

Trois exemples illustrent les rudiments de l'utilisation de MQ JMS avec WebSphere Application Server Advanced Edition. Ils se trouvent dans les sous-répertoires de *MQ_install_dir*/samples/ws, où *MQ_install_dir* est le répertoire d'installation de MQ JMS.

- Sample1 présente une simple opération de transmission et d'extraction de message dans une file d'attente à l'aide de transactions gérées par conteneur.
- Sample2 présente une simple opération de transmission et d'extraction de message dans une file d'attente à l'aide de transactions gérées par bean.
- Sample3 présente l'utilisation de l'API de publication/souscription.

Pour plus d'informations sur la création et le déploiement de beans EJB, reportez-vous à la documentation WebSphere Application Server.

Les fichiers readme.txt situés dans chaque répertoire sample comprennent un exemple de résultat pour chaque EJB. Les scripts fournis supposent qu'un gestionnaire de files d'attente par défaut soit disponible sur la machine locale. Si votre installation est différente de l'installation par défaut, vous pouvez éditer ces scripts selon vos besoins.

Sample1

Sample1EJB.java, dans le répertoire sample1, définit deux méthodes qui utilisent JMS :

- putMessage() envoie un message de type texte dans une file d'attente, et renvoie l'ID du message envoyé.
- getMessage() lit le message correspondant à l'ID du message à partir de la file d'attente.

Avant d'exécuter l'exemple, vous devez stocker deux objets administrés dans l'espace annuaire WebSphere JNDI :

QCF1 une fabrique de connexion de la file d'attente propre à WebSphere

Q1 une file d'attente

Les deux objets doivent être joints dans le sous-contexte jms/Samples.

Pour configurer les objets administrés, vous pouvez soit utiliser l'outil d'administration MQ JMS et les configurer manuellement, soit utiliser le script fourni.

L'outil d'administration MQ JMS doit être configuré pour accéder à l'espace annuaire WebSphere. Pour obtenir des informations sur la configuration de l'outil d'administration, reportez-vous à la section «Configuration pour WebSphere» à la page 35.

Pour configurer les objets administrés avec les paramètres par défaut, vous pouvez entrer la commande suivante pour exécuter le script `admin.scp` :

```
JMSAdmin < admin.scp
```

Le bean doit être déployé à l'aide des méthodes getMessage et putMessage indiquées en tant que TX_REQUIRED. Cela garantit que le conteneur démarre une transaction avant d'aborder une méthode et valide la transaction à la fin de la méthode. Dans une méthode, vous n'avez pas besoin de code d'application qui se rapporte à l'état transactionnel. Néanmoins, n'oubliez pas que le message envoyé depuis putMessage survient au point de synchronisation, et n'est pas disponible avant la validation de la transaction.

Dans le répertoire sample1, un programme client simple, Sample1Client.java, appelle le bean EJB. Un script, runClient, est également utilisé pour simplifier l'exécution de ce programme.

Le programme client (ou script), accepte un seul paramètre, utilisé comme corps d'un message de type texte qui sera envoyé par la méthode putMessage du bean EJB. Ensuite, getMessage est appelé pour lire le message dans la file d'attente et renvoyer le corps du message au client pour affichage. Le bean EJB envoie des messages d'avancement à la sortie standard (stdout) du serveur d'applications de sorte que vous puissiez contrôler cette sortie pendant l'exécution.

Si le serveur d'applications se trouve sur une machine éloignée du client, vous pouvez éditer Sample1Client.java. Si vous n'utilisez pas les valeurs par défaut, vous pouvez éditer le script runClient pour harmoniser le chemin d'installation local et le nom du fichier jar déployé.

Sample2

Sample2EJB.java, dans le répertoire sample2, effectue la même tâche que sample1, et suppose les mêmes objets administrés. Contrairement à sample1, sample2 utilise des transactions gérées par bean pour contrôler les limites transactionnelles.

Si vous n'avez pas encore lancé sample1, pensez à configurer les objets administrés QCF1 et Q1, comme indiqué à la section «Sample1» à la page 373.

Les méthodes putMessage et getMessage commencent par obtenir une instance de UserTransaction. Elles utilisent cette instance pour créer une transaction via la méthode UserTransaction.begin(). Ensuite, le corps du code est identique à celui de sample1 jusqu'à la fin de chaque méthode. A la fin de chaque méthode, la transaction se termine par l'appel UserTransaction.commit().

Dans le répertoire sample2, un programme client simple, Sample2Client.java, appelle le bean EJB. Un script, runClient, est également utilisé pour simplifier l'exécution de ce programme. Vous pouvez utiliser ces programmes comme décrit à la section «Sample1» à la page 373.

Sample3

Sample3EJB.java, dans le répertoire sample3, illustre l'utilisation de l'API de publication/souscription avec WebSphere. La publication d'un message est très semblable au cas point à point. Il existe néanmoins des différences pour la réception de messages via TopicSubscriber.

Les programmes de publication/souscription mettent généralement en oeuvre des souscriptions non durables. Ces dernières n'existent que le temps des sessions qui les détiennent (ou moins de temps si la souscription est fermée de manière explicite). En outre, les souscripteurs ne peuvent recevoir des messages du courtier que pendant leur durée de vie.

Pour convertir sample1 en publication/souscription, vous pouvez remplacer QueueSender dans putMessage par un objet TopicPublisher, et QueueReceiver dans getMessage par un objet TopicSubscriber non durable. Néanmoins, l'opération n'aboutirait pas, car lorsque le message serait envoyé, le courtier ne connaîtrait pas les souscripteurs à la rubrique. Le message serait donc supprimé.

La solution consiste à créer une souscription durable avant la publication du message. Les souscriptions durables demeurent des points d'extrémité de

distribution au delà de la durée de vie de la session. Ainsi, le message est disponible pour une extraction pendant l'appel de getMessage().

Le bean EJB comprend deux autres méthodes :

- createSubscription qui crée une souscription durable ;
- destroySubscription qui supprime une souscription durable.

Ces méthodes (avec putMessage et getMessage) doivent être déployées avec l'attribut TX_REQUIRED.

Avant d'exécuter sample3, vous devez stocker deux objets administrés dans l'espace annuaire JNDI WebSphere :

TCF1

T1

Les deux objets doivent être joints dans le sous-contexte jms/Samples.

Pour configurer les objets administrés, vous pouvez soit utiliser l'outil d'administration MQ JMS et les configurer manuellement, soit utiliser un script. Le script `admin.scp` se trouve dans le répertoire sample3.

L'outil d'administration MQ JMS doit être configuré pour accéder à l'espace annuaire WebSphere. Pour obtenir des informations sur la configuration de l'outil d'administration, reportez-vous à la section «Configuration pour WebSphere» à la page 35.

Pour configurer les objets administrés avec les paramètres par défaut, vous pouvez entrer la commande suivante pour exécuter le script `admin.scp` :

```
JMSAdmin < admin.scp
```

Si vous avez déjà lancé admin.scp pour configurer des objets pour sample1 ou sample2, vous obtiendrez des messages d'erreur lors de l'exécution d'admin.scp pour sample3. (Ces messages surviennent lorsque vous tentez de créer les sous-contextes jms et Samples.) Vous pouvez ignorer ces messages.

En outre, avant d'exécuter sample3, vérifiez que le courtier MQSeries de publication/souscription (SupportPac MA0C) est installé et en cours d'exécution.

Dans le répertoire sample3, un programme client exemple, Sample3Client.java, appelle le bean EJB. Un script, runClient, est également utilisé pour simplifier l'exécution de ce programme. Vous pouvez utiliser ces programmes comme décrit à la section «Sample1» à la page 373.

Annexe F. Remarques

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 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 information. La remise de ce information ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE EN L'ETAT. IBM DECLINE TOUTE RESPONSABILITE, EXPLICITE OU IMPLICITE, RELATIVE AUX INFORMATIONS QUI Y SONT CONTENUES, Y COMPRIS EN CE QUI CONCERNE LES GARANTIES DE VALEUR MARCHANDE OU D'ADAPTATION A VOS BESOINS. 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. Il est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut modifier sans préavis 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.

Remarques

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 United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans ce information et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux dispositions de l'ICA, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

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.

Marques

Les termes qui suivent sont des marques d'International Business Machines Corporation dans certains pays :

AIX	AS/400	BookManager
CICS	IBM	IBMLink
Language Environment	MQSeries	MVS/ESA
OS/2	OS/390	OS/400
SecureWay	SupportPac	System/390
S/390	VisualAge	VSE/ESA
WebSphere		

Java, HotJava, JDK et toutes les marques et logos incluant Java sont des marques de Sun Microsystems, Inc. dans certains pays.

Microsoft, Windows et Windows NT sont des marques de Microsoft Corporation dans certains pays.

UNIX est une marque de The Open Group dans certains pays.

D'autres sociétés sont propriétaires des autres marques, noms de produits ou logos qui pourraient apparaître dans ce document.

Glossaire de termes et d'abréviations

Ce glossaire explicite certains termes propres à ce manuel et certains mots plus généraux employés dans une acception inhabituelle. La définition proposée dans chaque cas n'est pas obligatoirement la seule définition du terme concerné, mais c'est celle qui correspond à l'utilisation du terme dans ce manuel.

Si vous ne trouvez pas le terme que vous recherchez, reportez-vous à l'index ou à la *Terminologie du traitement de l'information*, GCF2-0076, publiée par IBM France, ou encore à un dictionnaire d'informatique générale.

Abstract Window Toolkit pour Java (AWT) : Groupe de composants d'interface graphique (GUI) mis en oeuvre à l'aide des versions natives des composants propres à une plateforme.

applet : Programme Java conçu pour être exécuté uniquement sur une page Web.

API : Application Programming Interface

API (Application Programming Interface) : L'API d'un système tel que MQSeries est constituée des fonctions et des variables que les programmeurs peuvent utiliser dans leurs applications.

AWT : Abstract Window Toolkit pour Java.

conversion de type : Terme utilisé dans Java pour décrire la conversion explicite de la valeur d'un objet ou d'un type primitif en une autre valeur ou un autre type.

canal : Voir canal MQI.

canal MQI : Un canal MQI relie un client MQSeries à un gestionnaire de files d'attente situé sur un serveur et transmet les appels et les réponses MQI en duplex.

classe : Une classe est un ensemble cohérent de données et de méthodes permettant de manipuler ces données. On appelle ″instance d'une classe″ un objet créé à partir du moule de cette classe.

client : Dans MQSeries, un client est un composant d'exécution permettant aux applications utilisateur locales d'accéder aux fonctions de mise en file d'attente sur un serveur.

Commandes MQSeries (MQSC) : Commandes compréhensibles et communes à toutes les plateformes, permettant de manipuler les objets MQSeries.

EJB : Enterprise JavaBeans.

encapsulage : Technique de programmation propre aux environnements orientés objet. Elle permet de protéger les données d'un objet ou de les rendre privées : les programmeurs ne peuvent manipuler ces données qu'en appelant les méthodes correspondantes.

Enterprise JavaBeans (EJB) : Architecture de composants orientés serveur, distribuée par Sun Microsystems, permettant de créer une logique applicative réutilisable et des applications professionnelles transférables. Les composants Enterprise JavaBean sont écrits en langage Java et fonctionnent sur les serveurs compatibles avec EJB.

file d'attente : La file d'attente est un objet MQSeries que certaines applications utilisent pour communiquer par messages.

gestionnaire de files d'attente : Un gestionnaire de files d'attente est un programme système qui fournit des services de mise en file d'attente de messages aux applications.

HTML : Langage de marquage hypertexte (HTML)

IEEE : Institute of Electrical and Electronics Engineers.

IIOP : Protocole Internet Inter-ORB.

instance : Une instance est un objet. Lorsqu'une classe est utilisée pour produire un objet, on dit que l'objet est une instance de cette classe.

interface : Une interface est une classe ne contenant que des méthodes abstraites et pas de variables d'instance. Une interface offre un ensemble commun de méthodes qui peuvent être mises en œuvre par les sous-classes d'un certain nombre de classes distinctes.

Internet : Internet est un réseau public coopératif de partage d'informations. Sur le plan matériel, Internet exploite un sous-ensemble de toutes les ressources des réseaux publics de télécommunication actuels. Sur le plan technique, ce qui caractérise Internet, c'est l'utilisation d'un ensemble de protocoles nommé TCP/IP (Transmission Control Protocol/Internet Protocol).

JAAS : Java Authentication and Authorization Service.

Java Authentication and Authorization Service (JAAS) : Service Java d'authentification des entités et de contrôle des accès.

Java Developers Kit (JDK) : Ensemble de logiciels distribué par Sun Microsystem à l'intention des développeurs en Java. Le kit se compose d'un interpréteur Java, de classes Java et d'outils de

développement Java : compilateur, débogueur, désassembleur, afficheur d'applets, générateur de fichiers raccords et générateur de documentation.

Java Naming and Directory Interface (JNDI) : API écrite en langage de programmation Java. Elle fournit des fonctions d'attribution de nom et de répertoires à des applications écrites en langage Java.

Java Message Service (JMS) : API de Sun Microsystem permettant d'accéder aux systèmes de messagerie des entreprises à partir de programmes Java.

Java Runtime Environment (JRE) : Sous-ensemble de JDK contenant les fichiers et les exécutables qui constituent la plateforme Java standard. JRE comprend Java Virtual Machine, les classes Core, et les fichiers d'assistance.

Java Transaction API (JTA) : API permettant aux applications et aux serveurs J2EE d'accéder aux transactions.

Java Transaction Service (JTS) : Gestionnaire de transactions prenant en charge JTA et mettant en œuvre le mappage Java de OMG Object Transaction Service 1.1 à un niveau inférieur à celui de l'API.

Java Virtual Machine (JVM) : Implémentation logicielle d'un CPU exécutant du code Java compilé (applets et applications).

Java 2 Platform, Enterprise Edition (J2EE) : Ensemble de services, d'API et de protocoles permettant de créer des applications Web à niveaux multiples.

JDK : Java Developers Kit.

JNDI : Java Naming and Directory Service.

JMS : Java Message Service.

JRE : Java Runtime Environment.

JTA : Java Transaction API.

JTS : Java Transaction Service.

JVM : Java Virtual Machine.

J2EE : Java 2 Platform, Enterprise Edition.

Langage de marquage hypertexte (HTML) : Langage utilisé pour décrire les informations à afficher sur le Web.

LDAP : Lightweight Directory Access Protocol.

Lightweight Directory Access Protocol (LDAP) : Protocole client-serveur permettant d'accéder à une fonction de répertoire.

message : Dans le cadre des applications de gestion de files d'attente de messages, un message est une communication entre programmes.

messages, file d'attente : Voir File d'attente.

mise en file d'attente de messages : Technique de programmation dans le cadre de laquelle chaque programme au sein d'une application communique avec les autres programmes par le biais de messages placés en file d'attente.

méthode : En programmation orientée objet, le terme méthode désigne l'équivalent d'une fonction ou d'une procédure.

module (ou package) : Un module Java autorise l'accès d'une portion de code Java à un ensemble de classes donné. Tout code Java faisant partie d'un module donné a accès à toutes les classes du module et à toutes les méthodes et zones non privées de ces classes.

MQDLH : (MQSeries dead letter header) En-tête. Voir le manuel *MQSeries Application Programming Reference*.

MQMD : MQSeries Message Descriptor.

MQSC : Commandes MQSeries.

MQSeries : Famille de programmes IBM permettant de gérer des files d'attente de messages.

MQSeries Message Descriptor (MQMD) : Données de contrôle décrivant le format et les propriétés du message présenté comme partie intégrante du message MQSeries.

navigateur Web : Programme de formatage et d'affichage d'informations diffusées sur le Web.

objet : (1) Dans Java, un objet est une instance de classe. Une classe modélise un ensemble d'éléments ; un objet modélise un membre particulier de cet ensemble. (2) Dans MQSeries, les gestionnaires de files d'attente, les files d'attente et les canaux sont les différents types d'objets gérés.

ORB (Object Request Broker) : Architecture standard d'applications permettant l'interaction entre différents objets créés en différents langages et hébergés sur différentes machines dans un environnement informatique réparti.

Object Management Group (OMG) : Consortium chargé de définir les normes de programmation orientée objet.

OMG : Object Management Group.

ORB : Object Request Broker.

privé : Une zone privée est invisible de l'extérieur de la classe à laquelle elle appartient.

protégé : Une zone protégée n'est visible que de l'intérieur de sa classe, de l'intérieur d'une sous-classe ou de l'intérieur des modules auxquels la classe appartient.

Protocole Internet Inter-ORB(IIOP) : Norme de communication TCP/IP entre les ORB (voir ce terme) de différents fournisseurs.

public : Une classe ou une interface publique est visible de n'importe où. Une méthode ou une variable publique est visible de partout où sa classe est visible.

Red Hat Package Manager (RPM) : Ensemble de logiciels utilisés sur les plateformes Red Hat Linux et autres plateformes Linux et UNIX.

RPM : Red Hat Package Manager.

serveur : (1) Un serveur MQSeries est un gestionnaire de files d'attente offrant des services de mise en file d'attente de messages à des applications client exécutées sur des postes de travail éloignés. (2) Plus généralement, un serveur est un programme qui répond aux demandes d'informations dans le modèle informatique client-serveur. (3) Ordinateur sur lequel est exécuté un programme serveur.

servlet : Programme Java conçu pour être exécuté uniquement sur un serveur Web.

sous-classe : Une sous-classe est une classe qui étend les possibilités d'une autre classe. Toute sous-classe hérite des méthodes et variables publiques et protégées de sa superclasse.

superclasse : Une superclasse est une classe dont les fonctionnalités sont étendues par une ou plusieurs autres classes. Les méthodes et variables publiques et protégées de la superclasse sont mises à disposition de ses sous-classes.

surcharge : Survient lorsqu'un identificateur fait référence à plusieurs éléments dans la même portée. Sous Java, vous pouvez surcharger les méthodes, mais pas les variables ni les opérateurs.

TCP/IP : Transmission Control Protocol/Internet Protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP) : Ensemble de protocoles de communication permettant la connectivité d'égal à égal sur réseaux locaux et réseaux ouverts.

Uniform Resource Locator (URL) : Ensemble de caractères désignant les ressources d'information sur un ordinateur ou un réseau tel qu'Internet.

URL : Uniform Resource Locator.

VisiBroker pour Java : ORB (voir ce terme) écrit en Java.

Web : Voir World Wide Web.

World Wide Web (ou Web) : Le Web est un service Internet fondé sur un ensemble de protocoles et permettant à un ordinateur serveur configuré de façon appropriée de diffuser des documents sur Internet de façon standard.

Glossaire

Bibliographie

Cette section répertorie la documentation disponible pour toute la gamme de produits MQSeries.

Publications MQSeries communes aux différentes plateformes

La plupart de ces publications, parfois appelées manuels de la «gamme» MQSeries, concernent l'ensemble des produits MQSeries Level 2. Les derniers éléments MQSeries Level 2 de la gamme sont les suivants :

- MQSeries for AIX, V5.2
- MQSeries pour AS/400, V5.2
- MQSeries pour AT&T GIS UNIX, V2.2
- MQSeries pour Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA, V2.1.1
- MQSeries pour Windows, version 2.0
- MQSeries pour Windows, version 2.1
- MQSeries for Windows NT et Windows 2000, version 5.2

Les publications MQSeries communes aux différentes plateformes sont les suivantes :

- *MQSeries Brochure*, G511-1908
- *Introduction à MQSeries (Messaging and Queuing)*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries - Clients*, GC11-1099
- *MQSeries - Administration du système*, SC11-1219
- *MQSeries - Guide de référence des commandes MQSC*, SC11-1526
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries - Guide des messages*, GC11-1218
- *MQSeries - Guide de programmation d'applications*, SC11-1087
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries - Utilisation de Java*, SC11-1511
- *MQSeries Application Messaging Interface*, SC34-5604

Publications MQSeries spécifiques d'une plateforme

Outre les manuels de la gamme MQSeries, il existe au moins une publication par plateforme pour chaque produit MQSeries.

MQSeries for AIX, V5.2

MQSeries for AIX - Mise en route, GC33-1867

MQSeries pour AS/400, V5.2

MQSeries pour AS/400 - Mise en route, GC11-1588

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries pour AT&T GIS UNIX, V2.2

MQSeries for AT&T GIS UNIX System Management Guide (SC33-1642)

MQSeries pour Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries pour Compaq (DIGITAL) OpenVMS - Guide de gestion du système (GC11-1197).

MQSeries for Compaq Tru64 UNIX, V5.1

MQSeries for Compaq Tru64 UNIX - Mise en route, GC34-5684

MQSeries for HP-UX, V5.2

MQSeries for HP-UX Quick Beginnings, GC33-1869

MQSeries for Linux, V5.2

MQSeries for Linux Quick Beginnings, GC34-5691

MQSeries for OS/2 Warp, V5.1

MQSeries for OS/2 Warp - Mise en route, GC11-1214

MQSeries for OS/390, V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3, Version 1.2

MQSeries link pour R/3 - Guide de l''utilisateur (GC11-1445).

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide (GC33-1768)

MQSeries for Sun Solaris, V5.2

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide (GC33-1893).

MQSeries for VSE/ESA, V2.1.1

MQSeries for VSE/ESA Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA System Management Guide (GC34-5364).

MQSeries pour Windows, version 2.0

MQSeries pour Windows - Guide de l'utilisateur (GC11-1170).

MQSeries pour Windows, version 2.1

MQSeries pour Windows - Guide de l'utilisateur (GC11-1289).

MQSeries for Windows NT et Windows 2000, version 5.2

MQSeries for Windows NT - Mise en route (GC11-1509).

MQSeries pour Windows NT - Utilisation de l'interface COM (Component Object Model) (SC11-1510).

MQSeries LotusScript Extension (SC34-5404).

Documentation en ligne

La plupart des manuels MQSeries sont disponibles à la fois en version papier et en ligne.

Format HTML

La documentation MQSeries est livrée au format HTML avec les produits MQSeries suivants :

- MQSeries for AIX, V5.2
- MQSeries pour AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT et Windows 2000, version 5.2 (HTML compilé)
- MQSeries Link pour R/3 version 1.2

Elle est également disponible sur le site Web consacré à la gamme des produits MQSeries :

```
http://www.ibm.com/software/mqseries/
```

Format PDF (Portable Document Format)

Les fichiers de type PDF peuvent être visualisés et imprimés à l'aide de Adobe Acrobat Reader.

Pour vous procurer Adobe Acrobat Reader ou obtenir les informations les plus récentes sur les plateformes prenant en charge Acrobat Reader, accédez à l'adresse suivante sur le Web :

```
http://www.adobe.com/
```

Les manuels MQSeries sont livrés au format PDF pour les produits suivants :

- MQSeries for AIX, V5.2

- MQSeries pour AS/400, V5.2
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT et Windows 2000, version 5.2
- MQSeries Link pour R/3 version 1.2

La totalité de la documentation MQSeries est également disponible au format PDF sur le site Web consacré à la gamme des produits MQSeries :

`http://www.ibm.com/software/mqseries/`

Format BookManager

La bibliothèque MQSeries est fournie au format IBM BookManager dans une série de bibliothèques en ligne, parmi lesquelles *Transaction Processing and Data*, SK2T-0730. Pour visualiser ces manuels au format BookManager, utilisez les logiciels IBM suivants :

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ pour Windows

Format PostScript

La bibliothèque MQSeries est livrée au format PostScript (.PS) avec de nombreux produits MQSeries version 2. Ces manuels peuvent être imprimés sur n'importe quelle imprimante PostScript, ou consultés à l'aide du logiciel de visualisation adéquat.

Format Aide Windows

Le manuel *MQSeries pour Windows - Guide de l'utilisateur* est livré au format Aide Windows avec les produits MQSeries pour Windows version 2.0 et version 2.1.

Informations MQSeries disponibles sur Internet

Le site Web consacré à la gamme des produits MQSeries se trouve à l'adresse suivante :

`http://www.ibm.com/software/mqseries/`

En suivant les liens à partir de ce site, vous pouvez :

- obtenir les informations les plus récentes sur la gamme des produits MQSeries ;
- accéder aux manuels MQSeries au format HTML ou PDF ;
- télécharger MQSeries SupportPacs.

Index

D

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

Z

Envoi de vos commentaires à IBM

Si l'un des aspects de ce manuel vous plaît ou vous déplaît particulièrement, veuillez utiliser l'une des méthodes listées ci-dessous pour envoyer vos commentaires à IBM.

Indiquez en toute franchise ce que vous considérez comme des erreurs ou des omissions spécifiques et faites part de vos commentaires sur l'exactitude, l'organisation, le contenu et l'aspect complet ou non de ce manuel.

Veuillez limiter vos commentaires aux informations contenues dans ce manuel et à la manière dont elles sont présentées.

Pour tout commentaire relatif aux fonctions des produits ou systèmes IBM, adressez-vous à votre représentant IBM ou à votre revendeur IBM agréé.

En envoyant vos commentaires à IBM, vous accordez à IBM le droit non exclusif d'utiliser ou diffuser vos commentaires, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part.

Vous pouvez envoyer vos commentaires à IBM de l'une des manières suivantes :

- Par courrier, à cette adresse :

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  Royaume-Uni

- Par fax :
  - Si vous ne résidez pas au Royaume-Uni, utilisez 44–1962–870229 après votre code d'accès international.
  - Depuis le Royaume-Uni, utilisez 01962–870229.
- De manière électronique, en utilisant l'ID réseau approprié :
  - IBM Mail Exchange : GBIBM2Q9 at IBMMAIL
  - IBMLink : HURSLEY(IDRCF)
  - Internet : idrcf@hursley.ibm.com

Quelle que soit la méthode utilisée, assurez-vous que vous joignez :

- Le titre de la publication et le numéro de commande
- Le sujet auquel se rapportent vos commentaires
- Vos nom et adresse / numéro de téléphone / numéro de fax / ID réseau.

SC11-1511-02