9.4

IBM MQ

Remarque

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section «Remarques», à la page 313.

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

Table des matières

Présentation technique du IBM MQ

IBM MQ permet de connecter vos applications et de gérer la distribution des informations au sein de votre organisation.

IBM MQ permet aux programmes de communiquer entre eux sur un réseau de composants différents (processeurs, systèmes d'exploitation, sous-systèmes et protocoles de communication) à l'aide d'une même interface de programmation cohérente. Les applications conçues via cette interface sont connues sous le nom d'applications de mise en file d'attente de messages.

Utilisez les sous-rubriques suivantes pour en savoir plus sur la mise en file d'attente des messages et les autres fonctions d'IBM MQ.

Concepts associés

Présentation de IBM MQ

Où trouver des informations sur les exigences liées au produit et sur le support ?

Tâches associées

Planification d'une architecture IBM MQ

Référence associée

«Fonctions principales et avantages de la mise en file d'attente de messages», à la page 7

Ces informations mettent en évidence quelques fonctions et avantages de la mise en file d'attente des messages. Elles décrivent les fonctions telles que la sécurité et l'intégrité des données de la mise en file d'attente de messages.

Présentation de la mise en file d'attente de messages

Les produits IBM MQ permettent aux programmes de communiquer entre eux sur un réseau de composants disparates (processeurs, systèmes d'exploitation, sous-systèmes et protocoles de communication) en utilisant une interface de programmation d'application cohérente.

Les applications conçues et écrites à l'aide de cette interface sont appelées applications de *mise en file d'attente de messages*, car elles utilisent le style *messagerie* et *mise en file d'attente* :

- Messagerie signifie que les programmes communiquent en s'envoyant des données dans des messages plutôt qu'en s'appelant directement.
- Mise en file d'attente signifie que les messages sont stockés dans des files d'attente, pour que les programmes puissent être exécutés indépendamment les uns des autres, à des vitesses et des heures différentes et dans des emplacements distincts, sans être connectés de manière logique.

La mise en file d'attente de messages a été utilisée dans le traitement de l'information pendant de nombreuses années. Elle est plus couramment utilisée aujourd'hui dans la messagerie électronique. Sans le système de mise en file d'attente, l'envoi d'un message électronique sur de longues distances nécessite que chaque noeud du chemin soit disponible pour l'acheminement des messages et que les adresses soient connectées et informées du fait que vous tentez de leur envoyer un message. Dans un système de mise en file d'attente, les messages sont stockés sur des noeuds intermédiaires jusqu'à ce que le système soit prêt à les acheminer. A leur destination finale, ils sont stockés dans une boîte aux lettres électronique jusqu'à ce que le destinataire soit prêt à les lire.

Toute fois, la plupart des transactions commerciales complexes sont traitées aujourd'hui sans le système de mise en file d'attente. Dans un réseau de grande taille, il se peut que le système gère plusieurs milliers de connexions prêtes à être utilisées. Si une partie du système rencontre un problème, de nombreuses parties du système deviennent inutilisables.

Vous pouvez considérer la mise en file d'attente de messages comme étant une messagerie électronique pour les programmes. Dans un environnement de mise en file d'attente de messages, chaque programme faisant partie d'une suite d'applications exécute une fonction autonome bien définie en réponse à une demande spécifique. Pour communiquer avec un autre programme, un programme doit placer un message dans une file d'attente prédéfinie. L'autre programme extrait le message de la file d'attente et

traite les demandes et les informations contenues dans le message. La mise en file d'attente de messages est donc un style de communication de programme à programme.

La mise en file d'attente est un mécanisme permettant de conserver les messages tant qu'une application n'est pas prête à les traiter. La mise en file d'attente vous permet :

- de communiquer entre les programmes (chacun pouvant être exécuté dans des environnements différents) sans avoir à écrire le code de communication ;
- de sélectionner l'ordre dans lequel un programme traite les messages ;
- d'équilibrer les charges sur un système en prévoyant plusieurs programmes pour la prise en charge d'une file d'attente lorsque le nombre de messages dépasse un seuil ;
- d'augmenter la disponibilité de vos applications en prévoyant un système secondaire pour la prise en charge des files d'attente si votre système principal est indisponible.

Description d'une file d'attente de messages

Une file d'attente de messages, connue tout simplement sous le nom de file d'attente, est une destination nommée à laquelle les messages peuvent être envoyés. Les messages s'accumulent dans les files d'attente jusqu'à ce qu'ils soient extraits par des programmes prenant en charge ces files d'attente.

Les files d'attente résident dans un gestionnaire de files d'attente et sont gérées par celui-ci, (voir «Terminologie relative à la mise en file d'attente de messages», à la page 9). La nature physique d'une file d'attente dépend du système d'exploitation sur lequel le gestionnaire de files d'attente est en cours d'exécution. Une file d'attente peut être une zone de mémoire tampon volatile dans la mémoire d'un ordinateur ou un jeu de données sur une unité de stockage permanent (telle qu'un disque). La gestion physique des files d'attente incombe au gestionnaire de files d'attente et n'est pas apparente pour les programmes d'application participants.

Les programmes accèdent aux files d'attente uniquement via les services externes du gestionnaire de files d'attente. Ils peuvent ouvrir une file d'attente, y placer des messages, en extraire des messages et la fermer. Ils peuvent également définir et consulter les attributs des files d'attente.

Différents styles de mise en file d'attente de messages

Point-à-point

Un message est placé dans la file d'attente et une application reçoit ce message.

Dans la messagerie point-à-point, l'application d'envoi doit connaître certaines informations sur l'application de réception avant de pouvoir lui envoyer un message. Par exemple, elle doit connaître le nom de la file d'attente à laquelle envoyer les informations et peut également indiquer un nom de gestionnaire de files d'attente.

Publication/Abonnement

Une copie de chaque message publié par une application de publication est livrée à toutes les applications intéressées. Il se peut qu'il y ait plusieurs, une ou aucune application intéressée. Dans la messagerie de publication/abonnement, une application intéressée est appelée un abonné et les messages sont placés dans une file d'attente identifiée par un abonnement.

La messagerie de type publication/abonnement vous permet de dissocier le fournisseur d'informations des consommateurs de ces informations. Il n'est pas nécessaire que l'application émettrice et l'application de réception se connaissent pour que les informations puissent être envoyées et reçues. Pour plus d'informations, voir «Messagerie de type publication/abonnement», à la page 66.

Avantages de la mise en file d'attente de messages pour le concepteur et le développeur d'applications

IBM MQ permet aux programmes d'application d'utiliser la *mise en file d'attente des messages* pour participer à un traitement géré par message. Les programmes d'application peuvent communiquer

sur des plateformes différentes à l'aide des produits logiciels appropriés de mise en file d'attente de messages. Par exemple, les applications z/OS peuvent communiquer via IBM MQ for z/OS. Les applications sont protégées du mécanisme des communications sous-jacentes. Voici quelques autres avantages de la mise en file d'attente de messages :

- Vous pouvez concevoir des applications à l'aide de petits programmes pouvant être partagés entre plusieurs applications.
- Vous pouvez générer rapidement de nouvelles applications en réutilisant ces blocs fonctionnels.
- Les applications écrites pour l'utilisation des techniques de mise en file d'attente de messages ne sont pas affectées par les modifications apportées au mode de fonctionnement des gestionnaires de files d'attente.
- Il n'est pas nécessaire d'utiliser des protocoles de communication. Le gestionnaire de files d'attente traite tous les aspects de la communication pour vous.
- Il n'est pas nécessaire que les programmes recevant des messages soient en cours d'exécution lorsque des messages leur sont envoyés. Les messages sont conservés dans des files d'attente.

Les concepteurs peuvent réduire le coût de leurs applications car le développement est plus rapide, un plus petit nombre de développeurs est requis et les demandes en compétences de programmation sont inférieures à celles des applications qui n'utilisent pas la mise en file d'attente de messages.

IBM MQ implémente une interface de programmation d'application commune appelée *message queue interface* (ou MQI), quel que soit l'emplacement d'exécution des applications. Cela facilite la portabilité des programmes d'application d'une plateforme à l'autre.

Pour plus de détails sur l'interface MQI, voir Présentation de l'interface MQI (Message Queue Interface).

Fonctions principales et avantages de la mise en file d'attente de messages

Ces informations mettent en évidence quelques fonctions et avantages de la mise en file d'attente des messages. Elles décrivent les fonctions telles que la sécurité et l'intégrité des données de la mise en file d'attente de messages.

Les principales fonctions des applications qui utilisent les techniques de mise en file d'attente de messages sont les suivantes :

Remarque : En ce qui concerne les clients et les serveurs IBM MQ, il n'est pas nécessaire de modifier une application serveur pour qu'elle prenne en charge des IBM MQ MQI clients sur les supplémentaires sur les nouvelles plateformes. De même, le IBM MQ MQI client peut, sans modification, fonctionner avec d'autres types de serveurs.

Absence de connexion directe entre les programmes

La mise en file d'attente de messages est une technique de communication indirecte de programme à programme. Elle peut être utilisée au sein de n'importe quelle application dans laquelle les programmes communiquent entre eux. La communication est établie lorsqu'un programme place des messages dans une file d'attente (détenue par un gestionnaire de files d'attente) et qu'un autre programme extrait les messages de la file d'attente.

Les programmes peuvent extraire des messages placés dans une file d'attente par d'autres programmes. Ces derniers peuvent être connectés au même gestionnaire de files d'attente que le programme récepteur ou à un autre gestionnaire de files d'attente. Cet autre gestionnaire de files d'attente peut résider sur un autre système, un système informatique différent, voire dans un secteur d'activité ou une entreprise différente.

Il n'existe aucune connexion physique entre les programmes qui communiquent à l'aide de files d'attente de messages. Un programme envoie des messages à une file d'attente détenue par un gestionnaire de files d'attente et un autre programme extrait les messages de la file d'attente (voir Figure 1, à la page 8).

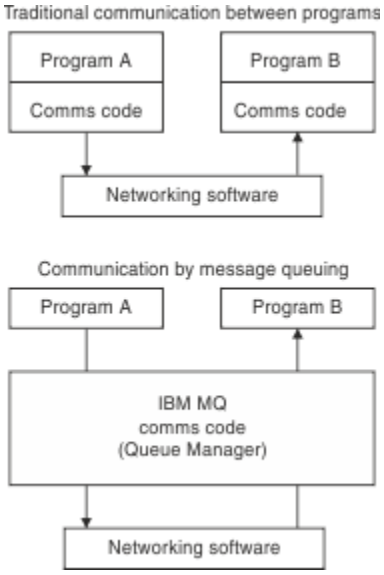


Figure 1. Comparaison de la mise en file d'attente de messages à une communication classique

Comme dans le cas de la messagerie électronique, les messages individuels faisant partie d'une transaction se déplacent via un réseau dans le cadre d'un processus de stockage et retransmission.:NONE. Si un lien entre les noeuds échoue, le message est conservé jusqu'à ce que le lien soit restauré, ou bien l'opérateur ou le programme réachemine le message.

Le mécanisme permettant de déplacer un message d'une file d'attente à une autre n'est pas visible par les programmes. Par conséquent, les programmes sont plus simples.

Communication sans contrainte de temps

Les programmes qui demandent à d'autres programmes d'effectuer un travail n'ont pas besoin d'attendre la réponse à une demande. Ils peuvent effectuer d'autres travaux et traiter la réponse lorsque celle-ci arrive ou ultérieurement. Lors de l'écriture d'une application de messagerie, vous n'avez pas besoin de connaître (ou de vous en préoccuper) à quel moment un programme envoie un message ou la cible est capable de recevoir le message. Le message n'est pas perdu ; il est conservé par le gestionnaire de files d'attente jusqu'à ce que la cible soit prête à le traiter. Le message demeure dans la file d'attente tant qu'il n'est pas supprimé par un programme. En d'autres termes, les programmes d'application émetteur et récepteur sont dissociés ; l'émetteur peut poursuivre le traitement sans attendre que le récepteur accuse réception du message. Il n'est même pas nécessaire que l'application cible soit en cours d'exécution lorsque le message est envoyé. Elle peut extraire le message une fois qu'elle a été démarrée.

Petits programmes

La mise en file d'attente de messages vous permet de tirer parti de l'utilisation de petits programmes autonomes. A la place d'un programme unique de grande taille effectuant toutes les parties d'un travail en séquence, vous pouvez répartir le travail sur plusieurs programmes indépendants de plus petite taille. Le programme demandeur envoie des messages à chacun des programmes distincts, leur demandant

d'accomplir leur fonction ; lorsque chaque programme est terminé, les résultats sont renvoyés sous la forme d'un ou de plusieurs messages.

Traitement géré par message

A l'arrivée des messages dans une file d'attente, ceux-ci peuvent automatiquement démarrer une application à l'aide d'un *déclenchement*. Si nécessaire, les applications peuvent être arrêtées lorsque le message (ou les messages) a été traité.

Traitement géré par événement

Les programmes peuvent être contrôlés en fonction de l'état des files d'attente. Par exemple, vous pouvez faire en sorte qu'un programme démarre dès qu'un message arrive dans une file d'attente ou vous pouvez préciser que le programme ne démarre que lorsque la file d'attente contient, par exemple, 10 messages au-dessus d'une certaine priorité ou 10 messages ayant n'importe quelle priorité.

Priorité de message

Un programme peut attribuer une priorité à un message lorsqu'il place le message dans une file d'attente. Cela détermine l'emplacement de la file d'attente dans lequel le nouveau message est ajouté.

Les programmes peuvent extraire des messages d'une file d'attente soit dans l'ordre des messages dans la file d'attente, soit en extrayant un message spécifique. (Un programme peut être amené à extraire un message spécifique s'il recherche la réponse à une demande qu'il a envoyée plus tôt.)

Sécurité

Des fonctions de sécurité sont fournies et incluent l'authentification des applications lorsqu'elles utilisent un gestionnaire de files d'attente, les vérifications d'autorisation lorsqu'elles utilisent des ressources comme une file d'attente sur le gestionnaire de files d'attente, et le chiffrement des données de message qui sont transmises via le réseau et qui se trouvent dans les files d'attente. Pour plus d'informations sur la sécurité, voir Présentation de la sécurité.

Intégrité des données

L'intégrité des données est assurée par des unités d'oeuvre. La synchronisation du début et de la fin des unités d'oeuvre est entièrement prise en charge sous forme d'option sur chaque MQGET ou MQPUT, ce qui permet la validation ou l'annulation des résultats de l'unité d'oeuvre. Le fonctionnement du support des points de synchronisation est interne ou externe à IBM MQ en fonction de la forme de coordination de point de synchronisation sélectionnée pour l'application.

Prise en charge de la reprise

Pour que la reprise soit possible, toutes les mises à jour persistantes d'IBM MQ sont consignées. Si une reprise est nécessaire, tous les messages persistants sont restaurés, toutes les transactions en cours annulées et toutes les validations et annulations de point de synchronisation traitées normalement par le gestionnaire de point de synchronisation responsable. Pour plus d'informations sur les messages persistants, voir Persistance des messages.

Terminologie relative à la mise en file d'attente de messages

Ces informations donnent un aperçu de certains termes utilisés dans le cadre de la mise en file d'attente de messages.

Il s'agit des termes suivants :

- Canaux
- Grappe
- IBM MQ MQI client

- z/OS Mise en file d'attente intra-groupe
- Message
- Agent MCA (Message Channel Agent)
- Descripteur de message
- Point à point
- Publication/abonnement
- File d'attente
- Gestionnaire de files d'attente
- z/OS Groupe de partage de files d'attente
- z/OS File d'attente partagée
- Abonnement
- Rubrique

Canaux

Les canaux sont utilisés pour déplacer des messages d'un gestionnaire de files d'attente vers un autre et ils protègent les applications des protocoles de communication sous-jacents. Les gestionnaires de files d'attente peuvent résider sur la même plateforme ou sur des plateformes différentes. Les messages envoyés peuvent provenir de nombreux endroits :

- programmes d'application écrits par l'utilisateur qui transfèrent des donnés d'un noeud vers un autre,
- applications d'utilisateur écrites par l'utilisateur qui utilisent des commandes PCF ou WebSphere MQ Administration Interface,
- IBM MQ Explorer,
- gestionnaires de files d'attente qui envoient des messages d'événement d'instrumentation à un autre gestionnaire de files d'attente,
- gestionnaires de files d'attente qui envoient des commandes d'administration distantes à un autre gestionnaire de files d'attente. Par exemple, via l'utilisation de commandes MQSC ou de l'administrative REST API.

Pour plus d'informations sur les canaux, voir «Définitions de canal», à la page 34.

Cluster

Un *cluster* est un réseau de gestionnaires de files d'attente logiquement associés d'une manière ou d'une autre.

Dans un réseau IBM MQ qui utilise la mise en file d'attente répartie sans fonction de mise en cluster, chaque gestionnaire de files d'attente est indépendant. Si un gestionnaire de files d'attente doit envoyer des messages à un autre gestionnaire de files d'attente, il doit avoir défini une file d'attente de transmission et un canal vers le gestionnaire de files d'attente éloignées.

Les clusters sont utilisés pour deux raisons différentes : réduire l'administration du système et améliorer la disponibilité et l'équilibrage de charge.

Même si vous établissez un très petit cluster, vous bénéficiez d'une administration de système simplifiée. Les gestionnaires de files d'attente faisant partie d'un cluster nécessitent moins de définitions, ce qui réduit le risque d'erreur dans vos définitions.

Pour plus d'informations sur la mise en cluster, voir Clusters.

IBM MQ MQI client

Les *clients* IBM MQ MQI sont des composants installables indépendamment d' IBM MQ. MQI Client vous permet d'exécuter des applications IBM MQ à l'aide d'un protocole de communication, afin d'interagir avec un ou plusieurs serveurs MQI (Message Queue Interface) sur d'autres plateformes et de se connecter à leurs gestionnaires de files d'attente.

Pour des détails complets sur l'installation et l'utilisation des composants du IBM MQ MQI client, voir les rubriques suivantes :

- 

Installation d'un client IBM MQ sous AIX
- 

Installation d'un client IBM MQ sous Linux®
- 

Installation d'un client IBM MQ sous Windows
- 

Installation d'un client IBM MQ sous IBM i

et Configuration des connexions entre le serveur et le client.

Mise en file d'attente intragroupe

Les gestionnaires de files d'attente d'un groupe de partage de files d'attente peuvent communiquer à l'aide de canaux normaux ou vous pouvez utiliser une technique appelée *mise en file d'attente intra-groupe*, qui vous permet de transférer rapidement les messages sans définir de canaux. Cette remarque est valable pour IBM MQ for z/OS uniquement.

Pour plus d'informations sur la mise en file d'attente intra-groupe, voir «Intra-group queuing», à la page 221.

Message

Dans le cadre de la mise en file d'attente de messages, un message est un ensemble de données envoyées par un programme et destinées à un autre programme. Voir Messages IBM MQ.

Pour plus d'informations sur les types de messages, voir Types de message.

Agent MCA

Un agent MCA est l'extrémité d'un canal. Une paire d'agents MCA, un émetteur et un récepteur, constituent un canal et déplacent des messages d'un gestionnaire de files d'attente à un autre.

Pour plus d'informations sur l'utilisation des agents MCA, voir Introduction à la gestion de files d'attente réparties.

Descripteur de message

Un message IBM MQ est constitué d'informations de contrôle et de données d'application.

Les informations de contrôle sont définies dans une structure de descripteur de message (MQMD) et contient les éléments suivants :

- Le type du message
- Un identificateur pour le message
- La priorité de livraison du message

La structure et le contenu des données d'application sont déterminés par les programmes participants et non par IBM MQ.

Pour plus d'informations, voir MQMD.

Messagerie point-à-point

Dans la messagerie point-à-point, chaque message est envoyé depuis une application de production vers une application destinataire. Les messages sont transférés lorsque l'application de production place les messages dans une file d'attente et que l'application destinataire les extrait de cette file d'attente.

Messagerie de type publication/abonnement

Dans une messagerie de publication/abonnement, une copie de chaque message publié par une application de publication est livrée à toutes les applications intéressées. Il se peut qu'il y ait plusieurs, une ou aucune application intéressée. Dans la messagerie de publication/abonnement, une application intéressée est appelée un abonné et les messages sont placés dans une file d'attente identifiée par un abonnement.

Pour plus d'informations, voir «Messagerie de type publication/abonnement», à la page 66.

File d'attente

Destination nommée à laquelle les messages peuvent être envoyés. Les messages s'accumulent dans les files d'attente jusqu'à ce qu'ils soient extraits par des programmes prenant en charge ces files d'attente.

Pour plus d'informations, voir «Files d'attente», à la page 20.

Gestionnaire de files d'attente

Un *gestionnaire de files d'attente* est un programme système qui met à la disposition des applications des services de mise en file d'attente.

Il fournit une interface de programme d'application qui permet aux programmes de placer et d'extraire des messages dans les files d'attente. Un gestionnaire de files d'attente fournit des fonctions supplémentaires permettant aux administrateurs de créer des files d'attente, de modifier les propriétés des files d'attente existantes et de contrôler le fonctionnement du gestionnaire de files d'attente.

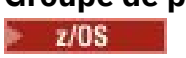
Pour que les services de mise en file d'attente de messages IBM MQ soient disponibles sur un système, un gestionnaire de files d'attente doit être en cours d'exécution. Plusieurs gestionnaires de files d'attente peuvent être en cours d'exécution sur un seul système (par exemple, pour séparer un système de test d'un système *opérationnel*). Pour une application, chaque gestionnaire de files d'attente est identifié par un *descripteur de connexion* (*Hconn*).

Plusieurs applications différentes peuvent utiliser en même temps les services du gestionnaire de files d'attente et peuvent être entièrement non apparentées. Pour qu'un programme utilise les services d'un gestionnaire de files d'attente, il doit établir une connexion avec ce gestionnaire de files d'attente.

Pour que les applications envoient des messages à des applications connectées à d'autres gestionnaires de files d'attente, les gestionnaires de files d'attente doivent être en mesure de communiquer entre eux. IBM MQ implémente un protocole de *stockage et retransmission* garantissant la distribution sécurisée des messages entre ces applications.

Pour plus d'informations, voir «Gestionnaires de files d'attente», à la page 30.

Groupe de partage de files d'attente



Les gestionnaires de files d'attente pouvant accéder au même ensemble de files d'attente partagées forment un groupe appelé *groupe de partage de files d'attente*. Ils communiquent entre eux à l'aide d'une unité de couplage qui stocke les files d'attente partagées. Cette remarque est valable pour IBM MQ for z/OS uniquement.

Pour plus d'informations, voir «Shared queues and queue sharing groups», à la page 177.

File d'attente partagée

z/OS

Une *file d'attente partagée* est un type de file d'attente locale comportant des messages accessibles à un ou plusieurs gestionnaires de files d'attente d'un sysplex. Il ne s'agit pas d'une file d'attente partagée par plusieurs applications, à l'aide du même gestionnaire de files d'attente. Cette remarque est valable pour IBM MQ for z/OS uniquement.

Abonnement

Une application de publication/abonnement peut enregistrer une information présentant un intérêt particulier dans des rubriques spécifiques. Si tel est le cas, l'application est appelée un abonné et le terme abonnement décrit comment les messages concordants sont mis en file d'attente à des fins de traitement.

Un abonnement contient des informations sur l'identité de l'abonné et l'identité de la file d'attente de destination sur laquelle les publications doivent être placées. Il contient également des informations sur la méthode de placement d'une publication dans la file d'attente de destination.

Pour plus d'informations, voir «Abonnés et abonnements», à la page 70.

Topic

Une rubrique est une chaîne de caractères décrivant l'objet des informations publiées dans un message de publication/abonnement.

Les rubriques sont primordiales pour la distribution des messages dans un système de publication/abonnement. Au lieu d'inclure une adresse de destination spécifique dans chaque message, un diffuseur de publications affecte une rubrique à chaque message. Le gestionnaire de files d'attente compare la rubrique à une liste d'abonnés qui s'y sont abonnés et distribue le message à chacun d'eux.

Pour plus d'informations, voir «Rubriques», à la page 73.

Messages et files d'attente

Les messages et les files d'attente constituent les composants de base d'un système de mise en file d'attente de messages.

Description d'un message

Un *message* est une chaîne d'octets significative pour les applications qui l'utilisent. Les messages servent à transférer des applications d'un programme d'application à l'autre (ou entre des parties différentes de la même application). Les applications peuvent être actives sur la même plateforme ou sur des plateformes différentes.

Un message IBM MQ est constitué des éléments suivants :

- *Données d'application.*Le contenu et la structure des données d'application sont définies par les programmes d'application qui les utilisent.
- *Descripteur de message.*Le descripteur de message identifie le message et contient des informations de contrôle supplémentaires, telles que le type de message et la priorité attribuée au message par l'application émettrice.

  Le format du descripteur de message est défini par IBM MQ. Pour obtenir une description complète du descripteur de message, voir MQMD - Descripteur de message.
- *Les propriétés de message.* Métadonnées du message. Le contenu des propriété de message sont définies par les programmes d'application qui les utilisent. Pour plus d'informations, voir Propriétés de message.

Longueurs de message

La longueur maximale par défaut d'un message est de 4 Mo, bien que vous puissiez l'augmenter à une longueur maximale de 100 Mo (où 1 Mo équivaut à 1 048 576 octets). En pratique, la longueur de message peut être limitée par :

- La longueur de message maximale définie pour la file d'attente réceptrice
- La longueur de message maximale définie pour le gestionnaire de files d'attente
- La longueur de message maximale définie pour la file d'attente
- La longueur de message maximale définie pour l'application émettrice ou l'application réceptrice
- L'espace de stockage disponible pour le message

Plusieurs messages peuvent être nécessaires pour l'envoi de toutes les informations requises par une application.

Mode d'envoi et de réception des messages par les applications

Les programmes d'application envoient et reçoivent les messages à l'aide d'**appels MQI**.

Par exemple, pour placer un message dans une file d'attente, une application :

1. ouvre la file d'attente requise, en émettant un appel MQI `MQOPEN` ;
2. émet un appel MQI `MQPUT` pour placer le message dans la file d'attente.

Une autre application peut extraire le message de la même file d'attente en émettant un appel MQI `MQGET`.

Pour plus d'informations sur les appels MQI, voir Appels MQI.

Description d'une file d'attente

Une *file d'attente* est une structure de données destinée au stockage de messages.

Chaque file d'attente est détenue par un *gestionnaire de files d'attente*. Le gestionnaire de files d'attente est responsable de la gestion des files d'attente qui lui appartiennent et du stockage de tous les messages qu'il reçoit dans les files d'attente appropriées. Les messages peuvent être placés dans la file d'attente par des programmes d'application ou par un gestionnaire de files d'attente dans le cadre de ses opérations normales.

Files d'attente prédéfinies et dynamiques

Les files d'attente peuvent être caractérisées par la façon dont elles sont créées :

- Les **files d'attente prédéfinies** sont créées par un administrateur à l'aide des commandes MQSC ou PCF appropriées.Les files d'attente prédéfinies sont permanentes ; elles existent indépendamment des applications qui les utilisent et sont conservées après le redémarrage d'IBM MQ.
- Les **files d'attente dynamiques** sont créées lorsqu'une application émet une demande `MQOPEN` indiquant le nom d'une *file d'attente modèle*. La file d'attente créée est basée sur une *définition de file d'attente modèle* appelée file d'attente modèle.Vous pouvez créer une file d'attente modèle à l'aide de la commande MQSC DEFINE QMODEL. Les attributs d'une file d'attente modèle (par exemple, le nombre maximal de messages qu'elle peut stocker) sont hérités des files d'attente dynamiques créées à partir la file d'attente modèle.

  Les files d'attente modèle comportent un attribut indiquant si la file d'attente dynamique doit être permanente ou temporaire. Les files d'attente permanentes sont conservées après le redémarrage de l'application et du gestionnaire de files d'attente ; les files d'attente temporaires sont perdues au redémarrage.

Extraction de messages des files d'attente

Seules les applications disposant des droits d'accès appropriés peuvent extraire des messages d'une file d'attente conformément aux algorithmes d'extraction suivants :

- Premier entré, premier sorti (FIFO)
- Priorité de message, comme défini dans le descripteur de message. Les messages ayant la même priorité sont extraits suivant la méthode FIFO.
- Demande de programme pour un message spécifique.

La demande MQGET émanant de l'application détermine la méthode utilisée.

Objets IBM MQ

Les gestionnaires de files d'attente définissent les propriétés des objets IBM MQ. Les valeurs de ces propriétés changent la manière dont IBM MQ traite ces objets. Vous créez et gérez des objets à l'aide de commandes et d'interfaces d'IBM MQ. Dans vos applications, vous utilisez MQI (Message Queue Interface) pour contrôler les objets. Les objets sont identifiés par un IBM MQ *descripteur d'objet* (MQOD) lorsqu'ils sont traités à partir d'un programme.

Administration des objets

L'administration des objets impliquent les tâches suivantes :

- Démarrage et arrêt des gestionnaires de files d'attente.
- Création d'objets, en particulier des files d'attente, pour les applications.
- Affichage ou modification des attributs d'objets.
- Suppression d'objets.
- Utilisation de canaux pour créer des chemins de communication vers des gestionnaires de files d'attente sur d'autres systèmes (distants).
- Création de *clusters* de gestionnaires de files d'attente pour simplifier le processus d'administration global et pour équilibrer la charge de travail.

A l'exception des files d'attente dynamiques, les objets doivent être définis sur le gestionnaire de files d'attente avant que vous puissiez les utiliser.

Lorsque vous utilisez une commande IBM MQ pour exécuter une opération d'administration d'objet, le gestionnaire de files d'attente vérifie que vous disposez des droits nécessaires pour effectuer l'opération. De même, lorsqu'une application utilise l'appel MQOPEN pour ouvrir un objet, le gestionnaire de files d'attente vérifie que l'application dispose du niveau de droits d'accès requis avant qu'elle soit autorisée à accéder à cet objet. Les vérifications s'effectuent au niveau du nom de l'objet à ouvrir.

Vous pouvez définir et gérer des objets à l'aide des méthodes suivantes :

- Les commandes PCF décrites dans les rubriques Référence de formats de commande programmable et Automatisation des tâches d'administration
- Les commandes MQSC décrites dans Les commandes MQSC
- z/OS Les panneaux d'opérations et de contrôle IBM MQ for z/OS , décrits dans Administration IBM MQ for z/OS
- Windows Linux IBM MQ Explorer (Windows et Linux pour les systèmes Intel uniquement). Pour plus d'informations, voir Présentation de MQ Explorer.

Vous pouvez également gérer les objets à l'aide des méthodes suivantes :

- Commandes de contrôle, saisies à partir d'un clavier. Voir Administration d' IBM MQ for Multiplatforms à l'aide de commandes de contrôle.
- Appels d'interface d'administration IBM MQ (MQAI) dans un programme. Voir IBM MQ Administration Interface (MQAI).

ALW Pour les séquences de commandes IBM MQ sous AIX, Linux, and Windows, vous pouvez utiliser la fonction MQSC afin d'exécuter une série de commandes stockées dans un fichier. Pour plus d'informations, voir Administration de IBM MQ à l'aide des commandes MQSC.

IBM i Pour les séquences de commandes IBM MQ pour IBM i que vous utilisez régulièrement, vous pouvez écrire des programmes CL. Pour plus d'informations, voir Gestion d'IBM MQ for IBM i à l'aide de commandes CL.

z/OS Pour les séquences de commandes IBM MQ for z/OS que vous utilisez régulièrement, vous pouvez écrire des programmes d'administration qui créent des messages contenant des commandes et qui placent ces messages dans la file d'attente d'entrée des commandes système. Le gestionnaire de files d'attente traite les messages de cette file d'attente de la même façon qu'il traite les commandes entrées à partir de la ligne de commande ou des panneaux d'opérations et de contrôle. Cette technique est décrite dans Ecriture de programmes pour administrer IBM MQ, et montrée dans l'exemple d'application Mail Manager fourni avec IBM MQ for z/OS. Pour la description de cet exemple, voir Exemples de programmes pour IBM MQ for z/OS .

Attributs d'objet

Les propriétés d'un objet sont définies par ses attributs. Certains peuvent être spécifiés et d'autres ne peuvent être que consultés.

Par exemple, la longueur de message maximale pouvant être prise en charge par une file d'attente est définie par son attribut **MaxMsgLength** ; vous pouvez spécifier cet attribut lorsque vous créez une file d'attente. L'attribut **DefinitionType** indique comment la file d'attente a été créée ; vous pouvez uniquement afficher cet attribut.

Dans IBM MQ, deux méthodes permettent de faire référence à un attribut :

- Utilisation de son nom PCF, par exemple, **MaxMsgLength**.
- Utilisation de son nom de commande MQSC, par exemple, MAXMSGL.

Groupes de partage de files d'attente

z/OS

Les gestionnaires de files d'attente pouvant accéder au même ensemble de files d'attente partagées forment un groupe appelé *groupe de partage de files d'attente* et peuvent communiquer entre eux à l'aide d'une unité de couplage qui stocke les files d'attente partagées. Notez qu'un QSG n'est pas strictement un objet.

Une file d'attente partagée est un type de file d'attente locale comportant des messages accessibles à un ou plusieurs gestionnaires de files d'attente d'un groupe de partage de files d'attente. Il ne s'agit pas d'une file d'attente partagée par plusieurs applications, à l'aide du même gestionnaire de files d'attente.

Le nom des groupes de partage de files d'attente comporte jusqu'à quatre caractères. Il doit être unique sur votre réseau et être différent de celui d'un gestionnaire de files d'attente.

Important : les files d'attente partagées et les groupes de partage de files d'attente sont pris en charge dans IBM MQ for z/OS uniquement.

Pour plus d'informations, voir «Shared queues and queue sharing groups», à la page 177.

Objets par défaut du système

Les *objets par défaut du système* sont un ensemble de définitions d'objet créées automatiquement chaque fois qu'un gestionnaire de files d'attente est créé. Vous pouvez copier et modifier l'une quelconque de ces définitions d'objet à des fins d'utilisation dans les applications de votre installation.

Les noms d'objet par défaut ont la racine SYSTEM ; par exemple, la file d'attente locale par défaut est SYSTEM.DEFAULT.LOCAL.QUEUE et le canal récepteur par défaut est SYSTEM.DEF.RECEIVER. Vous ne pouvez pas renommer ces objets ; des objets par défaut de ces noms sont requis.

Lors de la définition d'un objet, tous les attributs non spécifiés explicitement sont copiés à partir de l'objet par défaut approprié. Par exemple, si vous définissez une file d'attente locale, les attributs que vous ne spécifiez pas sont extraits de la file d'attente par défaut SYSTEM.DEFAULT.LOCAL.QUEUE.

Pour plus d'informations, voir Objets système et par défaut .

Types d'objet

La plupart des tâches d'administration impliquent la manipulation de différents types d' IBM MQ objets .

Pour plus d'informations sur l'attribution de nom aux objets IBM MQ, voir «Attribution de nom aux objets IBM MQ», à la page 39.

Pour plus d'informations sur les objets par défaut créés dans un gestionnaire de files d'attente, voir «Objets par défaut du système», à la page 16.

Pour plus d'informations sur les différents types d'objets IBM MQ , voir:

Objets d'informations d'authentification

Un objet d'informations d'authentification fournit les définitions requises pour la vérification de la révocation de certificat.

L'objet d'information d'authentification du gestionnaire de files d'attente fait partie de la prise en charge de Transport Layer Security (TLS) par IBM MQ. Il fournit les informations nécessaires pour la vérification des certificats révoqués. Les autorités de certification révoquent les certificats qui ne sont plus sécurisés.

Vous pouvez utiliser la commande MQSC **DEFINE AUTHINFO** pour définir un objet d'informations d'authentification. Pour plus d'informations sur les attributs des objets d'informations d'authentification, voir **DEFINE AUTHINFO**.

Vous pouvez utiliser les commandes de contrôle IBM MQ suivantes avec un objet d'informations d'authentification :

- **setmqaut** (octroi ou révocation des droits d'accès)
- **dspmqaut** (affichage des autorisations des objets)
- **dmpmqaut** (vidage des autorisations)
- **rcrmqobj** (recréation d'objet)
- **rcdmqimg** (enregistrement d'image de support)
- **dspmqfls** (affichage des noms de fichier)

Pour une présentation de TLS et de l'utilisation des objets d'informations d'authentification, voir les concepts TLS (Transport Layer Security) et les protocoles de sécurité TLS dans IBM MQ.

Canaux

Les canaux sont des objets qui fournissent un chemin de communication d'un gestionnaire de files d'attente vers un autre.

Pour plus d'informations, voir «Canaux», à la page 31.

Objets d'information de communication

IBM MQ Multicast fournit une messagerie multidiffusion fiable, à haute distribution et à faible latence. Un objet d'information de communication (COMMINFO) est nécessaire pour utiliser la transmission Multicast.

Pour plus d'informations, voir «Multidiffusion IBM MQ», à la page 114.

Un objet COMMINFO est un objet IBM MQ contenant les attributs associés à la transmission multidiffusion. Pour plus d'informations sur ces attributs, voir DEFINE COMMINFO. Pour plus d'informations sur la création d'un objet COMMINFO, voir Guide d'initiation à la multidiffusion.

Programmes d'écoute

Les *programmes d'écoute* sont des processus qui acceptent les demandes de réseau d'autres gestionnaires de files d'attente ou d'applications client et qui démarrent des canaux associés.

Les *processus du programme d'écoute* peuvent être démarrés à l'aide de la commande de contrôle **runmqlsr** .

Les *objets programme d'écoute* sont des objets IBM MQ qui permettent de gérer le démarrage et l'arrêt des processus d'écoute dans le cadre d'un gestionnaire de files d'attente. En définissant des attributs d'un objet programme d'écoute :

- Vous configurez le processus d'écoute.
- Vous indiquez si le processus d'écoute démarre et s'arrête automatiquement lors du démarrage et de l'arrêt du gestionnaire de files d'attente.

Important : z/OS Les objets de programme d'écoute ne sont pas pris en charge sur IBM MQ for z/OS. Pour plus d'informations sur l'implémentation de l'écoute par IBM MQ for z/OS en utilisant l'initiateur de canal, voir «The channel initiator on z/OS», à la page 173.

Listes de noms

namelist est un objet IBM MQ contenant une liste de noms de cluster, de noms de file d'attente ou de noms d'objet d'information d'authentification. Dans un cluster, elle permet d'identifier la liste des clusters pour lesquels le gestionnaire de files d'attente détient les référentiels.

Une liste de noms est un objet IBM MQ qui contient une liste d'autres objets IBM MQ. En général, les listes de noms sont utilisées par des applications telles que les moniteurs de déclenchement, où ces listes de noms sont alors utilisées pour identifier un groupe de files d'attente. L'utilisation d'une liste de noms présente un avantage : elle est gérée indépendamment des applications ; elle peut être mise à jour sans arrêter les applications qui l'utilisent. De même, si une application échoue, la liste de noms n'est pas concernée et les autres applications peuvent continuer de l'utiliser.

Les listes de noms sont également utilisées avec les clusters de gestionnaires de files d'attente pour gérer une liste de clusters référencés par plusieurs objets IBM MQ .

Vous pouvez définir et modifier des listes de noms à l'aide des commandes MQSC DEFINE NAMELIST et ALTER NAMELIST .

Remarque : z/OS Sinon, sous z/OS, vous pouvez utiliser les panneaux d'opérations et de contrôle IBM MQ for z/OS .

Les programmes peuvent utiliser l'interface MQI pour identifier les files d'attente qui sont incluses dans ces listes de noms. L'organisation des listes de noms incombe au concepteur d'application et à l'administrateur système.

Pour obtenir la liste des attributs de liste de noms disponibles, voir Attributs des listes de noms.

Définitions de processus

Les objets définition de processus permettent le démarrage des applications sans l'intervention de l'opérateur, en définissant les attributs de l'application à des fins d'utilisation par le gestionnaire de files d'attente.

L'objet définition de processus définit une application qui démarre en réponse à un événement déclencheur sur un gestionnaire de files d'attente IBM MQ. Les attributs de définition de processus incluent l'ID application, le type d'application et des données spécifiques à l'application. Pour plus d'informations, voir *Files d'attente d'initialisation* dans la rubrique «Files d'attente utilisées à des fins spécifiques par IBM MQ», à la page 28.

Pour permettre à une application de démarrer sans l'intervention de l'opérateur, comme décrit dans Démarrage des applications IBM MQ en utilisant des déclencheurs, les attributs de l'application doivent

être connus du gestionnaire de files d'attente. Ces attributs sont définis dans un *objet définition de processus*.

L'attribut **ProcessName** est fixe lorsque l'objet est créé. Toutefois, vous pouvez modifier les autres attributs à l'aide des commandes IBM MQ.

Remarque : z/OS Sous z/OS, vous pouvez également utiliser les panneaux d'opérations et de contrôle IBM MQ for z/OS .

Vous pouvez interroger les valeurs de tous les attributs à l'aide de MQINQ - Consultation des attributs d'un objet.

Pour obtenir la liste des attributs de définition de processus pouvant être utilisés, voir Attributs des définitions de processus.

Files d'attente

Un IBM MQ *file d'attente* est un objet nommé dans lequel les applications peuvent insérer des messages et à partir duquel les applications peuvent extraire des messages.

Pour plus d'informations, voir «Files d'attente», à la page 20.

Gestionnaires de files d'attente

IBM MQ Les gestionnaires de files d'attente fournissent des services de mise en file d'attente aux applications et gèrent les files d'attente qui leur appartiennent.

Pour plus d'informations, voir «Gestionnaires de files d'attente», à la page 30.

Services

Les objets *service* permettent de définir les programmes à exécuter lors du démarrage ou de l'arrêt d'un gestionnaire de files d'attente.

Les programmes doivent être de types :

Serveurs
Un *serveur* est un objet service dont le paramètre SERVTYPE est désigné par SERVER. Un objet service serveur est la définition d'un programme qui sera exécuté lors du démarrage d'un gestionnaire de files d'attente indiqué. Une seule instance d'un processus serveur peut être exécutée à un moment donné. Lors de l'exécution, le statut d'un processus serveur peut être surveillé à l'aide de la commande MQSC, DISPLAY SVSTATUS. En général, les objets de service serveur sont des définitions de programme, telles que les gestionnaires de messages non livrés ou les moniteurs de déclenchement ; cependant, les programmes pouvant être exécutés ne sont pas limités à ceux fournis avec IBM MQ. En outre, un objet service serveur peut être défini pour inclure une commande à exécuter lorsque le gestionnaire de files indiqué est arrêté pour mettre fin au programme.

Commandes
Une *commande* est un objet service dont le paramètre SERVTYPE est désigné par COMMAND. Un objet service de commande est la définition d'un programme qui sera exécuté lors du démarrage ou de l'arrêt d'un gestionnaire de files d'attente indiqué. Plusieurs instances d'un processus de commande peuvent être exécutées en même temps. Les objets service de commande diffèrent des objets service serveur dans la mesure où, une fois le programme exécuté, le gestionnaire de files d'attente ne surveillera pas le programme. En général, les objets service de commande sont des définitions de programmes éphémères et exécutent une tâche spécifique telle que le démarrage d'une ou de plusieurs autres tâches.

Important : z/OS Les objets de service ne sont pas pris en charge sous IBM MQ for z/OS.

Pour plus d'informations, voir Utilisation des services.

Classes d'archivage

z/OS

Une classe d'archivage établit la correspondance (mappe) une ou plusieurs files d'attente avec un ensemble de pages.

Cela signifie que les messages destinés à cette file d'attente sont stockés (sous réserve d'une mise en mémoire tampon) sur cet ensemble de pages.

Les classes d'archivage ne sont prises en charge que sur IBM MQ for z/OS.

Pour plus d'informations sur les classes de stockage, voir Planification de votre environnement IBM MQ sur z/OS.

Objets de rubrique

Un *objet rubrique* est un objet IBM MQ qui permet d'affecter des attributs spécifiques différents des attributs par défaut à des rubriques.

Une *rubrique* est définie par une application qui publie une *chaîne de rubrique ou s'y abonne*. Une chaîne de rubrique peut spécifier une hiérarchie de rubriques en les séparant par une barre oblique (/). Cela peut être visualisé par une *arborescence de rubriques*. Par exemple, si une application publie dans les chaînes de rubrique `/Sport/American Football` et `/Sport/Soccer`, une arborescence de rubriques est créée avec un noeud parent `Sport` avec deux enfants, `American Football`et `Soccer`.

Les rubriques héritent leurs attributs du premier noeud d'administration parent trouvé dans leur arborescence de rubriques. Si une arborescence de rubriques particulière ne contient aucun noeud de rubrique d'administration, toutes les rubriques héritent leurs attributs de l'objet de rubrique de base, SYSTEM.BASE.TOPIC.

Vous pouvez créer un objet de rubrique sur n'importe quel noeud d'une arborescence de rubriques en définissant la chaîne de rubrique du noeud dans l'attribut TOPICSTR de l'objet de rubrique. Vous pouvez également définir d'autres attributs pour le noeud de rubrique d'administration. Pour plus d'informations sur ces attributs, voir Les commandes MQSC ou Administration de l'automatisation à l'aide de commandes PCF. Par défaut, chaque objet de rubrique hérite des attributs de son noeud de rubrique administratif parent le plus proche.

Les objets de rubrique peuvent être également utilisés pour masquer aux développeurs d'applications la totalité de l'arborescence des rubriques. Si un objet de rubrique nommé `FOOTBALL.US` est créé pour la rubrique `/Sport/American Football`, une application peut publier ou s'abonner à l'objet nommé `FOOTBALL.US` au lieu de la chaîne `/Sport/American Football` avec le même résultat.

Si vous entrez un caractère #, +, / ou * dans une chaîne de rubrique dans un objet de rubrique, le caractère est créé comme caractère normal dans la chaîne, car il est considéré faire partie de la chaîne de rubrique associée à l'objet de rubrique.

Pour plus d'informations sur les objets de rubrique, voir «Messagerie de type publication/abonnement», à la page 66.

Concepts associés

«Présentation de la mise en file d'attente de messages», à la page 5
Les produits IBM MQ permettent aux programmes de communiquer entre eux sur un réseau de composants disparates (processeurs, systèmes d'exploitation, sous-systèmes et protocoles de communication) en utilisant une interface de programmation d'application cohérente.

Référence associée

Les commandes MQSC

Files d'attente

Présentation des files d'attente et des attributs de file d'attente IBM MQ.

Les messages sont stockés dans une file d'attente de sorte que, si l'application d'insertion attend une réponse à son message, elle est disponible pour effectuer d'autres activités en attendant cette réponse. Les applications accèdent à une file d'attente à l'aide de l'interface MQI (Message Queue Interface), décrite dans la rubrique Présentation de l'interface MQI (Message Queue Interface).

Avant qu'un message puisse être placé dans une file d'attente, celle-ci doit avoir déjà été créée. Une file d'attente est détenue par un gestionnaire de files d'attente qui peut posséder plusieurs files d'attente. Cependant, chaque file d'attente doit porter un nom unique au sein de ce gestionnaire de files d'attente.

Une file d'attente est gérée via un gestionnaire de files d'attente. Dans la plupart des cas, chaque file d'attente est physiquement gérée par son gestionnaire de files d'attente, ce qui n'est toutefois pas apparent pour un programme d'application. Les files d'attente partagées IBM MQ for z/OS peuvent être gérées par n'importe quel gestionnaire de files d'attente dans le groupe de partage de files d'attente.

Pour créer une file d'attente, vous pouvez utiliser des commandes IBM MQ (MQSC), des commandes PCF ou des interfaces spécifiques à la plateforme. Par exemple, les opérations et les panneaux de contrôle IBM MQ for z/OS sont spécifiques à la plateforme.

Vous pouvez créer *de façon dynamique* des files d'attente pour des travaux temporaires, à partir de votre application. Par exemple, vous pouvez créer des files d'attente de *réponse* (qui ne sont plus requises une fois qu'une application prend fin). Pour plus d'informations, voir «Files d'attente dynamiques et modèle», à la page 26.

Avant d'utiliser une file d'attente, vous devez l'ouvrir, afin d'indiquer l'opération souhaitée au niveau de la file d'attente. Par exemple, vous pouvez ouvrir une file d'attente pour :

- parcourir les messages uniquement (et non les extraire) ;
- extraire des messages (et partager l'accès avec d'autres programmes ou avec un accès exclusif) ;
- insérer des messages dans la fiel d'attente ;
- consulter les attributs de la file d'attente ;
- définir les attributs de la file d'attente.

Pour obtenir la liste complète des options pouvant être indiquées lors de l'ouverture d'une file d'attente, voir MQOPEN - Ouverture d'objet.

Attributs des files d'attente

Certains attributs d'une file d'attente sont spécifiés lors de la définition de la file d'attente et ne peuvent pas être modifiés par la suite (par exemple, le type de la file d'attente). D'autres attributs des files d'attente peuvent être regroupés en attributs pouvant être modifiés :

- par le gestionnaire de files d'attente lors du traitement de la file d'attente (par exemple, la longueur en cours d'une file d'attente) ;
- uniquement par des commandes (par exemple, la description de la file d'attente) ;
- par des applications, à l'aide de l'appel MQSET (par exemple, si des opérations d'insertion sont autorisées sur la file d'attente).

Les valeurs de tous les attributs sont disponibles à l'aide de l'appel MQINQ.

Les attributs communs à plusieurs types de file d'attente sont les suivants :

QName
Nom de la file d'attente.

QType
Type de la file d'attente.

QDesc
Description de la file d'attente.

InhibitGet
Indique si les programmes sont autorisés à extraire des messages de la file d'attente. Toutefois, vous ne pouvez jamais extraire de messages de files d'attente éloignées.

InhibitPut
Indique si les programmes sont autorisés à insérer des messages dans la file d'attente.

DefPriority
Priorité par défaut des messages insérés dans la file d'attente.

DefPersistence
Persistance par défaut des messages insérés dans la file d'attente

Portée
Permet de vérifier si une entrée pour cette file d'attente existe également dans un service annuaire.

z/OS L'attribut **Scope** n'est pas pris en charge sous z/OS

Pour une description complète de ces attributs, voir Attributs des files d'attente.

Méthodes de définition des files d'attente

Vous pouvez définir des files d'attente vers IBM MQ à l'aide de la commande MQSC DEFINE ou de la commande PCF de création de file d'attente (Create Queue). Les commandes indiquent le type de file d'attente et ses attributs. Par exemple, un objet file d'attente locale comporte des attributs indiquant ce qui se produit lorsque les applications font référence à cette file d'attente dans les appels MQI. Voici des exemples d'attributs :

- Attribut indiquant si les applications peuvent extraire des messages de la file d'attente (GET activé)
- Attribut indiquant si les applications peuvent placer des messages dans la file d'attente (PUT activé)
- Attribut indiquant si l'accès à la file d'attente est réservé exclusivement à une application ou s'il est partagé entre les applications
- Nombre maximal de messages pouvant être stockés simultanément dans la file d'attente (longueur maximale de la file d'attente)
- Longueur maximale des messages pouvant être placés dans la file d'attente

Il existe également diverses interfaces spécifiques à la plateforme que vous pouvez utiliser pour définir des files d'attente.

Concepts associés

«Files d'attente de cluster», à la page 62
Une file d'attente de cluster est une file d'attente hébergée par un gestionnaire de files d'attente de cluster et accessible aux autres gestionnaires de files d'attente dans le cluster.

«Files d'attente de rebut», à la page 53
La file d'attente de rebut (ou file d'attente des messages non livrés) est la file d'attente à laquelle des messages sont envoyés s'ils ne peuvent pas être acheminés vers la destination appropriée. Chaque gestionnaire de files d'attente a généralement une file d'attente de rebut.

Administration de l'automatisation à l'aide de commandes PCF

IBM MQ Console: Utilisation des files d'attente

Tâches associées

Administration d' IBM MQ à l'aide de commandes MQSC

Création et configuration de gestionnaires de files d'attente et d'objets avec MQ Explorer

IBM i Gestion d'IBM MQ for IBM i à l'aide de commandes CL

z/OS Sources à partir desquelles vous pouvez émettre des commandes MQSC et PCF sous IBM MQ for z/OS

Référence associée

«Comparison between shared queues and cluster queues», à la page 63
This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Information associée

«What is a shared queue?», à la page 177

Files d'attente locales

Les files d'attente de transmission, d'initialisation, de rebut, de commandes, par défaut, de canal et d'événements sont des types de files d'attente.

Une file d'attente est considérée par un programme comme étant *locale* si elle est détenue par le gestionnaire de files d'attente auquel le programme est connecté. Vous pouvez extraire et insérer des messages dans les files d'attente locales.

L'objet définition de file d'attente stocke les informations de définition de la file d'attente, ainsi que les messages physiques insérés dans la file d'attente.

Chaque gestionnaire de files d'attente peut comporter quelques files d'attente locales qu'il utilise à des fins spécifiques :

Files d'attente de transmission

Lorsqu'une application envoie un message à une file d'attente éloignée, le gestionnaire de files d'attente local stocke le message dans une file d'attente locale spéciale, appelée *file d'attente de transmission*. Les applications peuvent placer les messages directement dans une file d'attente de transmission ou indirectement via une définition de file d'attente éloignée.

Lorsqu'un gestionnaire de files d'attente envoie des messages à un gestionnaire de files d'attente éloignées, il identifie la file d'attente de transmission dans l'ordre suivant :

1. La file d'attente de transmission nommée sur l'attribut XMITQ de la définition locale d'une file d'attente éloignée.
2. Une file d'attente de transmission portant le même nom que le gestionnaire de files d'attente éloignées. Cette valeur est la valeur par défaut sur l'attribut XMITQ de la définition locale d'une file d'attente éloignée.
3. La file d'attente de transmission nommée sur l'attribut DEFXMITQ du gestionnaire de files d'attente local.

Un *agent MCA* est un programme de canal qui est associé à la file d'attente de transmission et qui distribue le message à sa destination suivante. La destination suivante est le gestionnaire de files d'attente auquel le canal de messages est connecté. Il ne s'agit pas nécessairement du même gestionnaire de files d'attente que la destination finale du message. Une fois le message distribué à sa destination suivante, il est supprimé de la file d'attente de transmission. Le message a peut-être besoin de traverser plusieurs gestionnaires de files d'attente sur son parcours jusqu'à sa destination finale. Vous devez définir une file d'attente de transmission sur chaque gestionnaire de files d'attente du parcours, chacun stockant des messages en attente de transmission à la destination suivante. Une file d'attente de transmission normale contient les messages pour la destination suivante, même si ces messages ont différentes destinations possibles. Une file d'attente de transmission du cluster contient les messages destinés à plusieurs destinations. Le `correlID` de chaque message identifie le canal dans lequel le message est placé pour son transfert vers la destination suivante.

Vous pouvez définir plusieurs files d'attente de transmission au niveau d'un gestionnaire de files d'attente. Vous pouvez définir plusieurs file d'attente de transmission pour la même destination, chacune étant utilisée pour une classe de service différente. Par exemple, vous pouvez créer différentes files d'attente de transmission pour les messages de petite taille et de grande taille ayant la même destination. Vous pouvez ensuite transférer ces messages au moyen de différents canaux de messages, de sorte que les messages de grande taille ne bloquent pas les messages plus petits. Tous les messages pour les files d'attente de cluster ou les rubriques de cluster sont placés dans la file d'attente de transmission de cluster unique, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, par défaut. Vous pouvez éventuellement modifier la valeur par défaut et séparer le trafic des messages à destination des gestionnaires de files d'attente de cluster dans des files d'attente de transmission du cluster différentes. Si vous définissez l'attribut de gestionnaire de files d'attente `DEFCLXQ` sur `CHANNEL`, chaque canal émetteur de cluster crée une file d'attente de transmission de cluster distincte. Il est également possible de définir manuellement les files d'attente de transmission de cluster pour les canaux émetteurs de cluster à utiliser.

Les files d'attente de transmission peuvent déclencher un agent de canal de message pour envoyer les messages. Voir Démarrage des applications IBM MQ en utilisant des déclencheurs.

z/OS Sur IBM MQ for z/OS, si vous utilisez la mise en file d'attente intra-groupe, la file d'attente de transmission est traitée par un *agent de mise en file d'attente intra-groupe*. Une file d'attente de transmission est utilisée lors de l'utilisation de la mise en file d'attente intra-groupe sur IBM MQ for z/OS.

Files d'attente d'initialisation

Une *file d'attente d'initialisation* est une file d'attente locale dans laquelle le gestionnaire de files d'attente insère un message de déclenchement lorsqu'un événement déclencheur se produit sur une file d'attente d'application.

Un événement déclencheur est un événement qui oblige un programme à démarrer le traitement d'une file d'attente. Par exemple, un événement peut être l'arrivée de plus de 10 messages. Pour plus d'informations sur le fonctionnement du déclenchement, voir Démarrage des applications IBM MQ en utilisant des déclencheurs.

File d'attente de rebut (de messages non livrés)

Une *file d'attente de rebut (de messages non livrés)* est une file d'attente locale dans laquelle le gestionnaire de files d'attente insère des messages qu'il ne peut pas distribuer.

Lorsque le gestionnaire de files d'attente insère un message dans la file d'attente de rebut, il ajoute un en-tête dans le message. Les informations d'en-tête incluent la raison pour laquelle le gestionnaire de files d'attente insère le message dans la file d'attente de rebut. Il contient également la destination du message d'origine, la date et l'heure auxquelles le gestionnaire de files d'attente a placé le message dans la file d'attente de rebut.

Les applications peuvent également utiliser la file d'attente pour les messages qu'elles ne peuvent pas distribuer. Pour plus d'informations, voir Utilisation de la file d'attente de rebut (de messages non livrés).

File d'attente de commandes système

La *file d'attente de commandes système* est une file d'attente à laquelle les applications disposant de droits d'accès appropriés peuvent envoyer des commandes IBM MQ. Ces files d'attente reçoivent les commandes PCF, MQSC et CL, telles qu'elles sont prises en charge sur votre plateforme, de sorte que le gestionnaire de files d'attente les traite.

z/OS Sous IBM MQ for z/OS , la file d'attente est appelée `SYSTEM.COMMAND.INPUT` ; sur les autres plateformes, il est appelé `SYSTEM.ADMIN.COMMAND.QUEUE`. Les commandes acceptées varient selon la plateforme. Voir Informations de référence sur les formats de commande programmables pour plus de détails.

Files d'attente par défaut du système

Les *files d'attente par défaut du système* contiennent les définitions initiales des files d'attente de votre système. Lors de la création d'une définition de file d'attente, le gestionnaire de files d'attente copie la définition à partir de la file d'attente par défaut appropriée du système. L'opération de création d'une définition de file d'attente est différente de l'opération de création d'une file d'attente dynamique. La définition de la file d'attente dynamique est basée sur la file d'attente modèle que vous choisissez comme modèle pour la file d'attente dynamique.

Files d'attente d'événements

Les *files d'attente d'événements* stockent des messages d'événement. Ces messages sont signalés par le gestionnaire de files d'attente ou un canal.

Files d'attente éloignées

Une file d'attente est considérée comme *éloignée* par un programme si elle est détenue par un gestionnaire de files d'attente différent de celui auquel le programme est connecté.

Lorsqu'une liaison de communication a été établie, un programme peut envoyer un message à une file d'attente éloignée. Un programme ne peut jamais extraire de message d'une file d'attente éloignée.

L'objet définition de file d'attente, créé lors de la définition d'une file d'attente éloignée, ne stocke que les informations nécessaires pour que le gestionnaire de files d'attente local recherche la file d'attente dans laquelle vous souhaitez placer votre message. Cet objet est appelé *définition locale d'une file d'attente éloignée*. Tous les attributs de la file d'attente éloignée sont stockés par le gestionnaire de files d'attente qui la détient, car elle est considérée comme file d'attente locale par ce gestionnaire de files d'attente.

Lors de l'ouverture d'une file d'attente éloignée, pour identifier cette dernière, vous devez indiquer l'un des éléments suivants :

- Le nom de la définition locale qui définit la file d'attente éloignée. Du point de vue d'une application, cette étape est la même que l'ouverture d'une file d'attente. Une application n'a pas besoin de savoir si une file d'attente est locale ou éloignée.

  Pour créer une définition locale d'une file d'attente éloignée sur toutes les plateformes sauf IBM i, utilisez la commande DEFINE QREMOTE .

  IBM i Sous IBM i, utilisez la commande CRTMQMQ.
- Le nom du gestionnaire de files d'attente éloignées et le nom de la file d'attente, tel qu'il est connu de ce gestionnaire de files d'attente éloignées.

Les définitions locales des files d'attente éloignées comportent trois attributs en sus des attributs communs décrits dans la section «Attributs des files d'attente», à la page 21. Il s'agit des trois attributs suivants :

RemoteQName
Nom sous lequel le gestionnaire de files d'attente propriétaire de la file d'attente connaît cette dernière.

RemoteQMgrName
Nom du gestionnaire de files d'attente propriétaire.

XmitQName
Nom de la file d'attente de transmission locale utilisée lors de l'acheminement des messages à d'autres gestionnaires de files d'attente.

Pour plus d'informations sur ces attributs, voir Attributs des files d'attente.

Si vous utilisez l'appel MQINQ au niveau de la définition locale d'une file d'attente éloignée, le gestionnaire de files d'attente renvoie les attributs de la définition locale uniquement, à savoir le nom de la file d'attente éloignée, le nom du gestionnaire de files d'attente éloignées et le nom de la file d'attente de transmission, et non les attributs de la file d'attente locale correspondante du système éloigné.

Voir aussi Files d'attente de transmission.

Files d'attente alias

Une *file d'attente alias* est un objet IBM MQ qui permet d'accéder à une autre file d'attente ou rubrique. Cela signifie que plusieurs programmes peuvent utiliser la même file d'attente en y accédant à l'aide de noms différents.

La file d'attente résultant de la résolution d'un nom d'alias, appelée file d'attente de base, peut être l'un des types de files d'attente pris en charge par la plateforme :

- Une file d'attente locale
- La définition locale d'une file d'attente éloignée.
- z/OS Une file d'attente partagée, qui est un type de file d'attente locale disponible uniquement sur IBM MQ for z/OS.

- Une file d'attente prédéfinie
- Une file d'attente dynamique

Un nom d'alias peut également correspondre à une rubrique. Si une application insère actuellement des messages dans une file d'attente, elle peut faire en sorte qu'ils soient publiés dans une rubrique en faisant du nom de la file d'attente un alias de la rubrique. Aucune modification du code d'application n'est nécessaire.

Remarque : Un alias ne peut pas se résoudre directement en un autre alias sur le même gestionnaire de files d'attente.

Exemple d'utilisation de files d'attente alias : un administrateur système accorde des droits d'accès différents au nom de file d'attente de base (c-à-d, la file d'attente à laquelle correspond l'alias) et au nom de file d'attente alias. En d'autres termes, un programme ou un utilisateur peut être autorisé à utiliser la file d'attente alias, et non la file d'attente de base.

Le cas échéant, des droits peuvent être définis de manière interdire les opérations d'insertion pour le nom d'alias mais les autoriser pour la file d'attente de base.

Dans certaines applications, l'utilisation de files d'attente alias signifie que les administrateurs système peuvent aisément modifier la définition d'un objet file d'attente alias sans avoir à modifier l'application.

IBM MQ effectue des contrôles d'autorisation au niveau du nom d'alias lorsque les programmes essaient d'utiliser ce nom. Il ne vérifie pas que le programme est autorisé à accéder au nom auquel correspond l'alias. Un programme peut donc être autorisé à accéder à un nom de file d'attente alias et non au nom de file d'attente résolu.

Outre les attributs de file d'attente généraux décrits dans la rubrique «Files d'attente», à la page 20, les files d'attente alias comportent un attribut **BaseQName**. Il s'agit du nom de la file d'attente de base auquel correspond le nom d'alias. Pour une description complète de cet attribut, voir BaseQName (MQCHAR48).

Les attributs *InhibitGet* et **InhibitPut** (voir «Files d'attente», à la page 20) des files d'attente alias font partie du nom d'alias. Par exemple, si le nom de file d'attente alias ALIAS1 correspond au nom de file d'attente de base BASE, les interdictions appliquées à ALIAS1 ont une incidence sur ALIAS1 uniquement, et BASE n'est pas interdit. Cependant, les interdictions appliquées à BASE ont également une incidence sur ALIAS1.

Les attributs *DefPriority* et **DefPersistence** font également partie du nom d'alias. Par conséquent, vous pouvez par exemple attribuer des priorités par défaut différentes à des alias différents de la même file d'attente de base. De même, vous pouvez modifier ces priorités sans avoir à modifier les applications qui utilisent les alias.

Files d'attente dynamiques et modèle

Cette rubrique fournit un aperçu des files d'attente dynamiques, des propriétés des files d'attente dynamiques temporaires et permanentes, des utilisations de files d'attente dynamiques, quelques remarques sur l'utilisation des files d'attente dynamiques, ainsi que des files d'attente modèle.

Lorsqu'un programme d'application émet un appel MQOPEN pour ouvrir une file d'attente modèle, le gestionnaire de files d'attente crée de façon dynamique une file d'attente locale dont les attributs sont les mêmes que ceux de la file d'attente modèle. En fonction de la valeur de la zone *DefinitionType* de la file d'attente modèle, le gestionnaire de files d'attente crée une file d'attente dynamique temporaire ou permanente (voir Création de files d'attente dynamiques).

Propriétés des files d'attente dynamiques temporaires

Les *files d'attente dynamiques temporaires* possèdent les propriétés suivantes :

- z/OS Elles ne peuvent pas être des files d'attente partagées, accessibles à partir de gestionnaires de files d'attente d'un groupe de partage de files d'attente.

  Notez que les files d'attente partagées et les groupes de partage de files d'attente sont disponibles uniquement dans IBM MQ for z/OS.
- Elles ne stockent que des messages non persistants.

- Elles sont irrécupérables.
- Elles sont supprimées lors du démarrage du gestionnaire de files d'attente.
- Elles sont supprimées lorsque l'application ayant émis l'appel MQOPEN qui a créé la file d'attente ferme la file d'attente ou prend fin.
  - Si la file d'attente contient des messages validés, ceux-ci sont supprimés.
  - S'il existe des appels MQGET, MQPUT ou MQPUT1 non validés en attente au niveau de la file d'attente à ce stade, la file d'attente est signalée comme étant supprimée logiquement et elle ne sera supprimée physiquement (une fois ces appels validés) que dans le cadre du processus de fermeture ou lorsque l'application prend fin.
  - Si la file d'attente est en cours d'utilisation (par l'application qui l'a créée ou par une autre application), la file d'attente est signalée comme étant supprimée logiquement et elle ne sera supprimée physiquement que lorsqu'elle est fermée par la dernière application qui l'utilise.
  - Les tentatives d'accès à une file d'attente supprimée logiquement (autres que la tentative de fermeture de la file d'attente) échouent en raison du code anomalie MQRC_Q_DELETED.
  - Les options MQCO_NONE, MQCO_DELETE et MQCO_DELETE_PURGE sont toutes traités sous forme de MQCO_NONE lorsqu'elles sont spécifiées dans un appel MQCLOSE pour l'appel MQOPEN correspondant qui a créé la file d'attente.

Propriétés des files d'attente dynamiques permanentes

Les *files d'attente dynamiques permanentes* possèdent les propriétés suivantes :

- Elles stockent des messages persistants ou non persistants.
- Elles sont récupérables en cas d'incidents système.
- Elles sont supprimées lorsqu'une application (non nécessairement celle qui a émis l'appel MQOPEN qui a créé la file d'attente) ferme correctement la file d'attente à l'aide de l'option MQCO_DELETE ou MQCO_DELETE_PURGE.
  - Une demande de fermeture à l'aide de l'option MQCO_DELETE échoue si la file d'attente contient toujours des messages (validés ou non validés). Une demande de fermeture à l'aide de l'option MQCO_DELETE_PURGE aboutit même si la file d'attente contient des messages validés (les messages en cours de suppression dans le cadre de la fermeture), mais échoue s'il existe des appels MQGET, MQPUT ou MQPUT1 en attente au niveau de la file d'attente.
  - Si la demande de suppression aboutit et que la file d'attente est en cours d'utilisation (par l'application qui l'a créée ou par une autre application), la file d'attente est signalée comme étant supprimée logiquement et elle ne sera supprimée physiquement que lorsqu'elle est fermée par la dernière application qui l'utilise.
- Elles ne sont pas supprimées si elles sont fermées par une application non autorisée à supprimer la file d'attente, sauf si l'application qui les ferme a émis l'appel MQOPEN qui a créé la file d'attente. Les vérifications d'autorisation s'effectuent au niveau de l'ID utilisateur (ou un ID utilisateur secondaire si l'option MQOO_ALTERNATE_USER_AUTHORITY a été spécifiée) qui a servi à valider l'appel MQOPEN correspondant.
- Elles peuvent être supprimées de la même façon qu'une file d'attente normale.

Utilisations des files d'attente dynamiques

Vous pouvez utiliser des files d'attente dynamiques pour :

- Les applications qui n'exigent pas la conservation des files d'attente après l'arrêt de l'application.
- Les applications demandant des réponses aux messages à traiter par une autre application. Les applications de ce type peuvent créer de façon dynamique une file d'attente de réponses en ouvrant une file d'attente modèle. Par exemple, une application client peut :
  1. Créer une file d'attente dynamique.
  2. Fournir son nom dans la zone **ReplyToQ** de la structure de descripteur du message de demande.
  3. Placer la demande dans une file d'attente traitée par un serveur.

Le serveur peut ensuite placer le message de réponse dans la file d'attente de réponses. Pour terminer, le client peut traiter la réponse et fermer la file d'attente de réponses à l'aide de l'option de suppression.

Remarques sur l'utilisation des files d'attente dynamiques

Prenez en considération les points suivants lors de l'utilisation des files d'attente dynamiques :

- Dans un modèle client-serveur, chaque client doit créer et utiliser sa propre file d'attente de réponses dynamique. Si une file d'attente de réponses dynamique est partagée entre plusieurs clients, la suppression de la file d'attente de réponses peut être retardée car la file d'attente contient des activités non validées en attente ou qu'elle est en cours d'utilisation par un autre client. En outre, il se peut que la file d'attente soit signalée comme étant supprimée logiquement et inaccessible pour les demandes d'API ultérieures (autres que MQCLOSE).
- Si votre environnement d'application exige que les files d'attente dynamiques soient partagées entre les applications, assurez-vous que la file d'attente n'est fermée (à l'aide de l'option de suppression) que lorsque toutes les activités au niveau de la file d'attente ont été validées. Cette opération doit être effectuée par le dernier utilisateur. Cela garantit que la suppression de la file d'attente n'est pas retardée et permet de réduire la période pendant laquelle la file d'attente est inaccessible car elle a été signalée comme étant supprimée logiquement.

Files d'attente modèle

Une *file d'attente modèle* est un modèle de définition de file d'attente utilisé lors de la création d'une file d'attente dynamique.

Vous pouvez créer une file d'attente locale de façon dynamique à partir d'un programme IBM MQ en désignant la file d'attente modèle que vous souhaitez utiliser comme modèle pour les attributs de file d'attente. A ce stade, vous pouvez modifier quelques attributs de la nouvelle file d'attente. Cependant, vous ne pouvez pas modifier la zone **DefinitionType**. Si, par exemple, vous avez besoin d'une file d'attente permanente, sélectionnez une file d'attente modèle dont le type de définition est paramétré sur permanente. Certaines applications conversationnelles peuvent utiliser des files d'attente dynamiques pour stocker les réponses à leurs requêtes, car elles n'ont probablement pas besoin de maintenir ces files d'attente une fois qu'elles ont traité les réponses.

Vous indiquez le nom d'une file d'attente modèle dans le *descripteur d'objet* (MQOD) de votre appel MQOPEN. A l'aide des attributs de la file d'attente modèle, le gestionnaire de files d'attente crée de façon dynamique une file d'attente locale.

Vous pouvez spécifier un nom (complet) pour la file d'attente dynamique ou la racine d'un nom (par exemple, ABC) et laisser le gestionnaire de files d'attente y ajouter une partie unique, ou vous pouvez demander au gestionnaire de files d'attente d'attribuer un nom unique complet. Si le gestionnaire de files d'attente attribue le nom, il le place dans la structure MQOD.

Vous ne pouvez pas émettre directement un appel MQPUT1 à une file d'attente modèle, mais vous pouvez émettre un appel MQPUT1 à la file d'attente dynamique qui a été créée via l'ouverture d'une file d'attente modèle.

MQSET et MQINQ ne peuvent pas être exécuté sur une file d'attente modèle. L'ouverture d'une file d'attente modèle avec MQOO_INQUIRE ou MQOO_SET entraîne l'exécution d'appels MQINQ et MQSET ultérieurs sur la file d'attente créée dynamiquement.

Les attributs d'une file d'attente modèle sont un sous-ensemble de ceux d'une file d'attente locale. Pour une description complète, voir Attributs des files d'attente.

Files d'attente utilisées à des fins spécifiques par IBM MQ

IBM MQ utilise certaines files d'attente locales pour certains buts liés à ses opérations.

Vous devez définir ces files d'attente avant qu'IBM MQ puisse les utiliser.

Files d'attente d'initialisation

Les files d'attente d'initialisation sont des files d'attente utilisées pour le déclenchement. Un gestionnaire de files d'attente place un message de déclenchement dans une file d'attente

d'initialisation lorsqu'un événement déclencheur se produit. Un événement déclencheur est une combinaison logique de conditions qui est détectée par un gestionnaire de files d'attente. Par exemple, un événement déclencheur peut être généré lorsque le nombre de messages dans une file d'attente atteint une longueur prédéfinie. Cet événement fait en sorte que le gestionnaire de files d'attente place un message de déclenchement dans une file d'attente d'initialisation indiquée. Ce message de déclenchement est extrait par un *moniteur de déclenchement*, qui correspond à une application spéciale surveillant une file d'attente d'initialisation. Le moniteur de déclenchement démarre ensuite le programme d'application qui a été indiqué dans le message de déclenchement.

Si un gestionnaire de files d'attente doit utiliser le déclenchement, au moins une file d'attente d'initialisation doit être définie pour ce gestionnaire de files d'attente. Voir Gestion des objets pour le déclenchement, runmqtrm et Démarrage des applications IBM MQ en utilisant des déclencheurs.

Files d'attente de transmission

Les files d'attente de transmission sont des files d'attente qui stockent provisoirement des messages destinés à un gestionnaire de files d'attente éloignées. Vous devez définir au moins une file d'attente de transmission pour chaque gestionnaire de files d'attente éloignées auquel le gestionnaire de files d'attente local doit envoyer directement des messages. Ces files d'attente sont également utilisées pour l'administration à distance. Voir Administration à distance à partir d'un gestionnaire de files d'attente local. Pour des informations sur l'utilisation des files d'attente de transmission pour la mise en file d'attente répartie, voir Techniques de distribution répartie d'IBM MQ.

Chaque gestionnaire de files d'attente peut posséder une file d'attente de transmission par défaut. Si un gestionnaire de files d'attente ne faisant pas partie d'un cluster place un message dans une file d'attente éloignée, l'action par défaut consiste à utiliser la file d'attente de transmission par défaut. S'il existe une file d'attente de transmission dotée du même nom que le gestionnaire de files d'attente de destination, le message est placé dans cette file d'attente de transmission. S'il existe une définition d'alias de gestionnaire de files d'attente, dans lequel le paramètre **RQMNAME** correspond au gestionnaire de files d'attente de destination, et si le paramètre **XMITQ** est spécifié, le message est placé dans la file d'attente de transmission appelée par **XMITQ**. S'il n'existe aucun paramètre **XMITQ**, le message est placé dans la file d'attente locale nommée dans le message.

Files d'attente de transmission de cluster

Chaque gestionnaire de files d'attente au sein d'un cluster comporte une file d'attente de transmission appelée SYSTEM.CLUSTER.TRANSMIT.QUEUE, et file d'attente de transmission de cluster modèle, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Les définitions de ces files d'attente sont créées par défaut lorsque vous définissez un gestionnaire de files d'attente. Si l'attribut de gestionnaire de files d'attente, **DEFCLXQ**, est défini sur CHANNEL, une file d'attente de transmission de cluster dynamique permanente est automatiquement créée pour chaque canal émetteur de cluster qui est créé. Les files d'attente sont appelées SYSTEM.CLUSTER.TRANSMIT. *ChannelName*. Vous pouvez également définir les files d'attente de transmission de cluster manuellement.

Un gestionnaire de files d'attente faisant partie du cluster envoie des messages d'une de ces files d'attente aux autres gestionnaires de files d'attente du même cluster.

Lors de la résolution de nom, une file d'attente de transmission de cluster est prioritaire sur la file d'attente de transmission par défaut, et une file d'attente de transmission de cluster spécifique est prioritaire sur SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Files d'attente de rebut

Une file d'attente de rebut (messages non livrés) est une file d'attente qui stocke les messages qui ne peuvent pas être acheminés vers leurs destinations correctes. Un message ne peut pas être acheminé lorsque, par exemple, la file d'attente de destination est saturée. La file d'attente de rebut fournie est appelée SYSTEM.DEAD.LETTER.QUEUE.

Pour la mise en file d'attente répartie, définissez une file d'attente de rebut dans chaque gestionnaire de files d'attente impliqué.

Files d'attente de commandes

La file d'attente de commandes, SYSTEM.ADMIN.COMMAND.QUEUE, est une file d'attente locale à laquelle les applications disposant de droits d'accès appropriés peuvent envoyer des commandes MQSC à des fins de traitement. Ces commandes sont ensuite extraites par un composant IBM MQ

appelé serveur de commandes. Le serveur de commandes valide les commandes, transmet les commandes valides à des fins de traitement par le gestionnaire de files d'attente, puis renvoie les réponses à la file d'attente de réponses appropriée.

Une file d'attente de commandes est créée automatiquement pour chaque gestionnaire de files d'attente lorsque ce dernier est créé.

Files d'attente de réponses
Lorsqu'une application envoie un message de demande, l'application qui reçoit le message peut renvoyer un message de réponse à l'application émettrice. Ce message est placé dans une file d'attente, appelée file d'attente de réponses, qui est généralement une file d'attente locale de l'application émettrice. Le nom de la file d'attente de réponses est indiqué par l'application émettrice dans le cadre du descripteur de message.

Files d'attente d'événements
Les événements outils permettent de surveiller les gestionnaires de files d'attente indépendamment des applications MQI.

Lorsqu'un événement outil se produit, le gestionnaire de files d'attente place un message d'événement dans une file d'attente d'événements. Ce message peut ensuite être lu par une application de surveillance qui peut informer un administrateur ou déclencher une action corrective si l'événement indique un problème.

Remarque : Les événements déclencheurs sont assez différents des événements outils. Les événements déclencheurs ne sont pas occasionnés par les mêmes conditions et ne génèrent pas de messages d'événement.

Pour plus d'informations sur les événements outils, voir Evénements outils.

Gestionnaires de files d'attente

Présentation des *gestionnaires de files d'attente* et des services de mise en file d'attente qu'ils fournissent aux applications.

Un programme doit disposer d'une connexion au gestionnaire de files d'attente avant qu'il puisse utiliser les services de ce gestionnaire de files d'attente. Un programme peut établir cette connexion de façon explicite (à l'aide de l'appel MQCONNX) ou la connexion peut être établie de façon implicite (en fonction de la plateforme et de l'environnement dans lequel le programme est en cours d'exécution).

Un gestionnaire de files d'attente IBM MQ garantit les actions suivantes:

- Les attributs d'objet sont modifiés selon les commandes reçues.
- Des événements spéciaux, tels que les événements déclencheurs ou les événements outils, sont générés lorsque les conditions appropriées sont remplies.
- Les messages sont placés dans la file d'attente correcte, à la demande de l'application qui établit l'appel MQPUT. L'application est informée si cela n'est pas possible et un code anomalie approprié est généré.

Chaque file d'attente fait partie d'un gestionnaire de files d'attente unique et est considérée comme étant une *file d'attente locale* de ce gestionnaire de files d'attente. Le gestionnaire de files d'attente auquel une application est connectée est considéré comme étant le *gestionnaire de files d'attente local* de cette application. Pour l'application, les files d'attente faisant partie de son gestionnaire de files d'attente local sont des files d'attente locales.

Une *file d'attente éloignée* est une file d'attente appartenant à un autre gestionnaire de files d'attente. Un *gestionnaire de files d'attente éloignées* est un gestionnaire de files d'attente autre que le gestionnaire de files d'attente local. Un gestionnaire de files d'attente éloignées peut résider sur une machine distante du réseau ou sur la même machine que le gestionnaire de files d'attente local. IBM MQ prend en charge plusieurs gestionnaires de files d'attente sur une même machine.

Un objet gestionnaire de files d'attente peut être utilisé dans certains appels MQI. Par exemple, vous pouvez visualiser les attributs de l'objet gestionnaire de files d'attente à l'aide de l'appel MQI MQINQ.

Attributs des gestionnaires de files d'attente

Chaque gestionnaire de files d'attente est associé à un ensemble d'attributs (ou de propriétés) définissant ses caractéristiques.Certains attributs d'un gestionnaire de files d'attente sont fixes lorsque celui-ci est créé ; vous pouvez en modifier d'autres à l'aide des commandes IBM MQ. Vous pouvez visualiser les valeurs de tous les attributs, à l'exception de ceux utilisés pour le chiffrement Transport Sockets Layer (TLS), à l'aide de l'appel MQINQ.

Les attributs fixes sont les suivants :

- Nom du gestionnaire de files d'attente
- Plateforme sur laquelle s'exécute le gestionnaire de files d'attente (par exemple, Windows)
- Niveau des commandes de contrôle système prises en charge par le gestionnaire de files d'attente
- Priorité maximale pouvant être affectée aux messages traités par le gestionnaire de files d'attente
- Nom de la file d'attente à laquelle les programmes peuvent envoyer des commandes IBM MQ
- Longueur maximale des messages que peut traiter le gestionnaire de files d'attente z/OS (fixe uniquement dans IBM MQ for z/OS)
- Prise en charge ou non de point de synchronisation par le gestionnaire de files d'attente lorsque des programmes placent et extraient des messages

Les attributs *modifiables* sont les suivants :

- Description du gestionnaire de files d'attente
- Identificateur du jeu de caractères utilisé par le gestionnaire de files d'attente pour les chaînes de caractères lorsqu'il traite les appels MQI
- Intervalle de temps utilisé par le gestionnaire de files d'attente pour restreindre le nombre de messages de déclenchement
- z/OS Intervalle de temps qu'utilise le gestionnaire de files d'attente pour déterminer la fréquence à laquelle il doit rechercher les messages expiré dans les files d'attente (IBM MQ for z/OS uniquement)
- Nom de la file d'attente de rebut (messages non livrés) du gestionnaire de files d'attente
- Nom de la file d'attente de transmission par défaut du gestionnaire de files d'attente
- Nombre maximal de descripteurs ouverts pour une connexion
- Activation et désactivation de diverses catégories de rapport d'événement
- Nombre maximal de messages non validés au sein d'une unité d'oeuvre

Gestionnaires de files d'attente et gestion de la charge de travail

Vous pouvez configurer un cluster de gestionnaires de files d'attente comportant plusieurs définitions pour la même file d'attente (par exemple, les gestionnaires de files d'attente du cluster peuvent être des clones les uns des autres). Les messages d'une file d'attente particulière peuvent être gérés par n'importe quel gestionnaire de files d'attente hébergeant une instance de la file d'attente. Un algorithme de gestion de la charge de travail détermine le gestionnaire de files d'attente qui traite le message et répartit ainsi la charge de travail entre vos gestionnaires de files d'attente ; pour plus d'informations, voir >Algorithme de gestion de la charge de travail des clusters.

Canaux

Un *canal* est une liaison de communication logique, utilisée par les gestionnaires de files d'attente réparties, entre un serveur IBM MQ MQI client et un serveur IBM MQ ou entre deux serveurs IBM MQ.

Les canaux sont utilisés pour déplacer des messages d'un gestionnaire de files d'attente vers un autre et ils protègent les applications des protocoles de communication sous-jacents. Les gestionnaires de files d'attente peuvent résider sur la même plateforme ou sur des plateformes différentes. Les messages envoyés peuvent provenir de nombreux endroits :

- programmes d'application écrits par l'utilisateur qui transfèrent des donnés d'un noeud vers un autre,
- applications d'utilisateur écrites par l'utilisateur qui utilisent des commandes PCF ou WebSphere MQ Administration Interface,
- IBM MQ Explorer,
- gestionnaires de files d'attente qui envoient des messages d'événement d'instrumentation à un autre gestionnaire de files d'attente,
- gestionnaires de files d'attente qui envoient des commandes d'administration distantes à un autre gestionnaire de files d'attente. Par exemple, via l'utilisation de commandes MQSC ou de l'administrative REST API.

Un canal comporte deux définitions : une à chaque extrémité de la connexion. Pour que les gestionnaires de communication puissent communiquer ensemble, vous devez définir un objet canal sur le gestionnaire de files d'attente qui envoie les messages et un autre (complémentaire) sur le gestionnaire de files d'attente qui reçoit les messages. Le même *nom de canal* doit être utilisé à chaque extrémité de la connexion, et le *type de canal* utilisé doit être compatible.

Il existe trois catégories de canaux dans IBM MQ, chacune comportant des types de canaux différents :

- Les canaux de messages, qui sont unidirectionnels et qui transfèrent les messages d'un gestionnaire de files d'attente à un autre.
- Les canaux MQI qui sont bidirectionnels et qui transfèrent les appels MQI d'un IBM MQ MQI client vers un gestionnaire de files d'attente, ainsi que les réponses d'un gestionnaire de files d'attente vers un client IBM MQ.
- Les canaux AMQP, qui sont bidirectionnels et qui connectent un client AMQP à un gestionnaire de files d'attente sur un serveur. IBM MQ utilise des canaux AMQP pour transférer des appels et des réponses AMQP entre des gestionnaires de files d'attente et des applications AMQP.

Canaux de transmission de messages

Un canal de transmission de message a pour but de transférer des messages d'un gestionnaire de files d'attente vers un autre. Les canaux de transmission de message ne sont pas requis par l'environnement client-serveur.

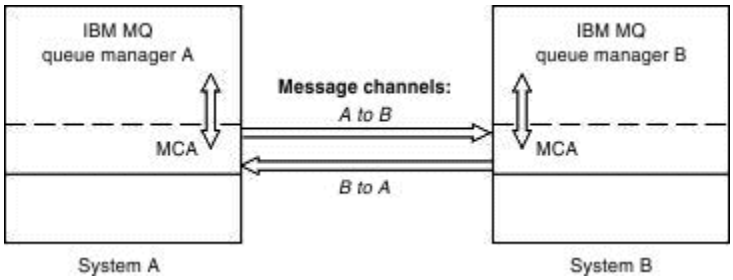


Figure 2. Canaux de transmission de message entre deux gestionnaires de files d'attente

Un canal de transmission de message est une liaison unidirectionnelle. Si vous souhaitez qu'un gestionnaire de files d'attente éloignées répondent aux messages envoyés par un gestionnaire de files d'attente local, vous devez configurer un second canal pour le renvoi des réponses au gestionnaire de files d'attente local.

Un canal de transmission de messages connecte deux gestionnaires de files d'attente grâce à des *Agents MCA*. Un agent MCA se trouve à chaque extrémité d'un canal. Vous pouvez autoriser un agent MCA à transférer des messages via plusieurs unités d'exécution. Ce processus est connu sous le nom de *principe du pipeline*. Le principe du pipeline permet à l'agent MCA de transférer des messages plus efficacement, améliorant ainsi les performances de canal. Pour plus d'informations sur le principe du pipeline, voir Attributs des canaux.

Pour plus d'informations sur les canaux, voir Structures de données et appels d'exit de canal et «Composants de la mise en file d'attente répartie», à la page 50.

Canaux MQI

Un canal MQI permet de connecter un IBM MQ MQI client à un gestionnaire de files d'attente sur une machine serveur et est établi lorsque vous émettez un appel `MQCONN` ou `MQCONNX` à partir d'une application IBM MQ MQI client.

Il s'agit d'une liaison bidirectionnelle qui sert à transférer uniquement des appels MQI et des réponses, y compris des appels `MQPUT` contenant des données de message et des appels `MQGET` donnant lieu au renvoi de données de message. Différentes méthodes permettent de créer et d'utiliser les définitions de canal (voir Définition de canaux MQI).

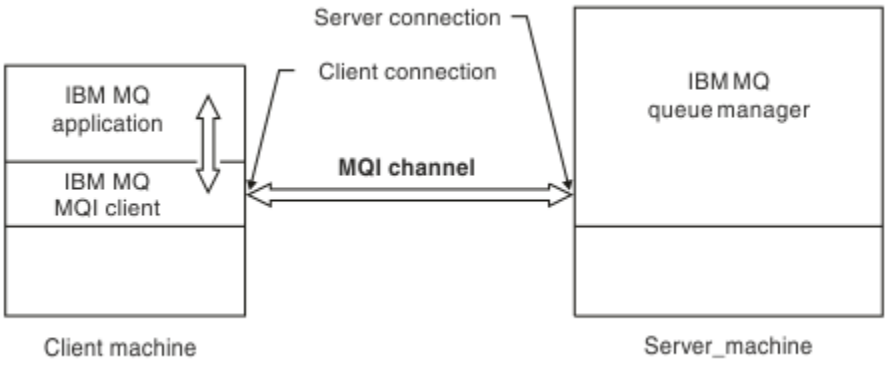


Figure 3. Connexion client et connexion serveur sur un canal MQI

z/OS Un canal MQI permet de connecter un client à un gestionnaire de files d'attente unique ou à un gestionnaire de files d'attente faisant partie d'un groupe de partage de files d'attente (voir Connexion d'un client à un groupe de partage de files d'attente).

Il existe deux types de canal pour les définitions de canal MQI. Ils définissent le canal MQI bidirectionnel.

Canal de connexion client
Ce type est destiné au IBM MQ MQI client.

Canal de connexion serveur
Ce type est destiné au serveur qui exécute le gestionnaire de files d'attente, avec lequel l'application IBM MQ en cours d'exécution dans un environnement IBM MQ MQI client, doit communiquer.

Canaux AMQP

Multi

Il n'existe qu'un type de canal AMQP.

Vous utilisez ce canal pour connecter une application de messagerie AMQP à un gestionnaire de files d'attente, pour permettre à l'application d'échanger des messages avec les applications IBM MQ. Un canal AMQP permet de développer une application à l'aide de MQ Light, puis de la déployer comme application d'entreprise pour profiter des utilitaires de niveau entreprise fournis par IBM MQ.

Canaux de connexion client

Les *canaux de connexion client* sont des objets fournissant un chemin de communication d'un IBM MQ MQI client vers un gestionnaire de files d'attente.

Les canaux de connexion client sont utilisés dans le cadre de la mise en file d'attente répartie pour déplacer des messages entre un gestionnaire de files d'attente et un client. Ils protègent les applications des protocoles de communication sous-jacents. Le client peut résider sur la même plateforme que le gestionnaire de files d'attente ou sur une plateforme différente.

Définitions de canal

Voir «Définitions de canal», à la page 34 pour une description de chaque type de canal.

Concepts associés

«Mise en file d'attente répartie et clusters», à la page 46
La mise en file d'attente répartie permet d'envoyer des messages d'un gestionnaire de files d'attente à un autre. Le gestionnaire de files d'attente récepteur peut résider sur la même machine ou sur une autre, à proximité ou à l'autre bout du monde. Il peut s'exécuter sur la même plateforme que le gestionnaire de files d'attente local ou se trouver sur n'importe laquelle des plateformes prises en charge par IBM MQ. Vous pouvez définir manuellement toutes les connexions dans un environnement de mise en file d'attente répartie, ou créer un cluster et laisser IBM MQ définir la plupart des informations de connexion automatiquement.

Présentation de l'interface MQI (Message Queue Interface)

Tâches associées

Administration des objets IBM MQ éloignés

Arrêt des canaux MQI

Configuration des connexions entre le serveur et le client

Référence associée

Structures de données et appels d'exit de canal

«Communications», à la page 38
Les IBM MQ MQI clients utilisent des canaux MQI pour communiquer avec le serveur.

Définitions de canal

Tableaux décrivant les différente types de canaux de transmission de messages utilisés par IBM MQ.

Pour désigner les canaux de transmission de messages, le terme canal est souvent utilisé comme synonyme de définition de canal. Le contexte permet généralement de savoir si l'on parle d'un canal complet, avec deux extrémités, ou d'une définition de canal avec une seule extrémité.

Canaux de transmission de messages

Les définitions de canaux de transmission de messages peuvent être de différents types :

Type de définition de canaux de transmission de messages	Description
Expéditeur	Un canal émetteur est un canal de transmission de messages utilisé par le gestionnaire de files d'attente pour envoyer des messages à d'autres gestionnaires de files d'attente. Pour envoyer des messages via un canal émetteur, vous devez aussi créer, sur l'autre gestionnaire de files d'attente, un canal récepteur portant le même nom que le canal émetteur. Vous pouvez également utiliser des canaux émetteurs avec des canaux demandeurs si vous mettez en place un mécanisme de procédure appelée (callback).

Type de définition de canaux de transmission de messages	Description
serveur	Un canal serveur est un canal de transmission de messages utilisé par le gestionnaire de files d'attente pour envoyer des messages à d'autres gestionnaires de files d'attente. Pour envoyer des messages via un canal serveur, vous devez aussi créer, sur l'autre gestionnaire de files d'attente, un canal récepteur portant le même nom que le canal serveur. Vous pouvez également utiliser des canaux serveur avec des canaux demandeur. Dans ce cas, la définition du canal demandeur à l'autre extrémité du canal demande à la définition du canal serveur de démarrer. Le serveur envoie des messages au demandeur. Le serveur peut également initialiser la communication s'il connaît le nom de connexion du canal partenaire.
Récepteur	Un canal récepteur est un canal de transmission de messages utilisé par le gestionnaire de files d'attente pour recevoir des messages en provenance d'autres gestionnaires de files d'attente. Pour recevoir des messages via un canal récepteur, vous devez aussi créer, sur l'autre gestionnaire de files d'attente, un canal émetteur ou serveur portant le même nom que le canal récepteur.
Demandeur	Un canal demandeur est un canal de transmission de messages utilisé par le gestionnaire de files d'attente pour recevoir des messages d'autres gestionnaires de files d'attente. Un canal demandeur peut demander le canal partenaire défini sur l'extrémité distante pour démarrer. Si le canal partenaire est un canal serveur, le canal serveur accepte la demande de démarrage et commencer à envoyer des messages depuis la file d'attente de transmission identifiée dans la définition de canal serveur au canal demandeur. Si le canal partenaire est un canal émetteur, le canal émetteur accepte la demande de démarrage, puis ferme la connexion au demandeur. Le canal émetteur démarre, négocie une session avec le canal demandeur partenaire, puis commence à envoyer des messages depuis la file d'attente de transmission identifiée dans la définition du canal émetteur. Ce dernier cas fournit essentiellement un mécanisme de rappel car le canal demandeur demande au canal émetteur de rappeler.

Type de définition de canaux de transmission de messages	Description
Emetteur de cluster	Une définition de canal émetteur de cluster (CLUSSDR) définit l'extrémité émettrice d'un canal, sur laquelle un gestionnaire de files d'attente du cluster peut envoyer des informations relatives au cluster à l'un des référentiels complets. Le canal émetteur de cluster permet d'informer le référentiel en cas de modification du statut du gestionnaire de files d'attente, par exemple en cas d'ajout ou de suppression d'une file d'attente. Il permet également de transmettre des messages. Les gestionnaires de files d'attente de référentiel complet eux-mêmes disposent de canaux émetteurs de cluster qui pointent l'un vers l'autre. Ils leur permettent de se communiquer mutuellement les modifications du statut du cluster. Il n'est pas très important de savoir vers quel référentiel complet pointe une définition de canal CLUSSDR du gestionnaire de files d'attente. Une fois le premier contact établi, un plus grand nombre d'objets gestionnaire de files d'attente du cluster est défini automatiquement selon les besoins, de sorte que le gestionnaire de files d'attente puisse envoyer des informations relatives au cluster à tous les référentiels complets, et des messages à tous les gestionnaires de files d'attente.
Récepteur de cluster	Une définition de canal récepteur de cluster (CLUSRCVR) définit l'extrémité réceptrice d'un canal, sur laquelle un gestionnaire de files d'attente du cluster peut recevoir des messages d'autres gestionnaires de files d'attente du cluster. Un canal récepteur de cluster peut également transmettre des informations relatives au cluster destinées au référentiel. En définissant le canal récepteur de cluster, le gestionnaire de files d'attente indique aux autres gestionnaires qu'il est disponible pour recevoir des messages. Vous devez disposer d'au moins un canal récepteur de cluster pour chaque gestionnaire de files d'attente du cluster.

Pour chaque canal, vous devez définir les deux extrémités afin de disposer d'une définition de canal pour chaque extrémité. Les deux extrémités du canal doivent avoir des types compatibles.

Vous pouvez combiner les définitions de canal de la façon suivante :

- Emetteur-Récepteur
- Serveur-Récepteur
- Demandeur-Serveur
- Demandeur-Emetteur (rappel)
- Emetteur de cluster-Récepteur de cluster

Agents MCA

Chaque définition de canal créée appartient à un gestionnaire de files d'attente spécifique. Un gestionnaire peut comporter plusieurs canaux d'un type identique ou différent. A chaque extrémité du canal se trouve un programme, l'agent MCA. A l'une des extrémités du canal, l'agent MCA demandeur

extrait les messages de la file de transmission et les envoie sur le canal. A l'autre extrémité du canal, l'agent MCA répondeur reçoit les messages et les délivre au gestionnaire de files d'attente éloignées.

Un agent MCA demandeur peut être associé à un canal émetteur, serveur ou demandeur. Un agent MCA répondeur peut être associé à n'importe quel type de canal de transmission de messages.

IBM MQ prend en charge les combinaisons suivantes de types de canaux aux deux extrémités d'une connexion :

Canal appelant		**Direction du flux de messages**	**Canal répondeur**	
Type de canal	**Programme d'écoute requis ?**		**Programme d'écoute requis ?**	**Type de canal**
Expéditeur	Non	Appelant à répondeur	Oui	Récepteur
serveur	Non	Appelant à répondeur	Oui	Récepteur
serveur	Non	Appelant à répondeur	Oui	Demandeur
Demandeur	Non	Répondeur à appelant	Oui	serveur
Demandeur	Oui	Répondeur à appelant	Oui	Expéditeur

Canaux MQI

Les canaux MQI peuvent être de l'un des types suivants :

Type de canal MQI	**Description**
Connexion serveur	Un canal de connexion serveur est un canal MQI bidirectionnel qui est utilisé pour connecter un client IBM MQ à un serveur IBM MQ. Le canal de connexion serveur constitue l'extrémité serveur du canal.
Connexion client	Un canal de connexion client est un canal MQI bidirectionnel qui est utilisé pour connecter un client IBM MQ à un serveur IBM MQ. IBM MQ Explorer utilise également des connexions client pour se connecter aux gestionnaires de files d'attente éloignées. Le canal de connexion client constitue l'extrémité client du canal. Lorsque vous créez un canal de connexion client, un fichier est créé sur l'ordinateur hébergeant le gestionnaire de files d'attente. Vous devez ensuite copier ce fichier de connexion client sur l'ordinateur client IBM MQ.

Multi *Prise en charge de plusieurs unités d'exécution-Pipetage*

Vous pouvez éventuellement autoriser un agent MCA à transférer des messages à l'aide de plusieurs unités d'exécution. Ce processus, appelé *pipeline*, permet à l'agent MCA de transférer les messages plus efficacement, avec moins d'états d'attente, ce qui améliore les performances du canal. Chaque agent MCA est limité à un maximum de deux unités d'exécution.

Vous pouvez contrôler le pipeline avec le paramètre *PipeLineLength* dans le fichier qm.ini . Ce paramètre est ajouté à la strophe Channels.

Remarque : Le principe du pipeline n'est efficace que pour les canaux TCP/IP.

Lorsque vous utilisez la fonction de pipeline, les gestionnaires de files d'attente aux deux extrémités du canal doivent être configurés pour avoir une valeur *PipeLineLongueur* supérieure à 1.

Remarques sur les exits de canal

Le pipeline peut entraîner l'échec de certains programmes d'exit pour les raisons suivantes:

- Il se peut que les exits ne soient pas appelés en série.
- Les exits peuvent être appelés alternativement à partir de différentes unités d'exécution.

Vérifiez la conception de vos programmes d'exit avant d'utiliser le pipeline:

- Les exits doivent être réentrants à toutes les étapes de leur exécution.
- Lorsque vous utilisez des appels MQI, n'oubliez pas que vous ne pouvez pas utiliser le même descripteur MQI lorsque l'exit est appelé à partir d'unités d'exécution différentes.

Prenons l'exemple d'un exit de message qui ouvre une file d'attente et utilise son descripteur pour les appels MQPUT sur tous les appels ultérieurs de l'exit. Cette opération échoue en mode de pipeline car l'exit est appelé à partir de différentes unités d'exécution. Pour éviter cet échec, conservez un descripteur de file d'attente pour chaque unité d'exécution et vérifiez l'identificateur de l'unité d'exécution chaque fois que l'exit est appelé.

Communications

Les IBM MQ MQI clients utilisent des canaux MQI pour communiquer avec le serveur.

Une définition de canal doit être créée sur les deux extrémités IBM MQ MQI client et serveur de la connexion. La procédure de création de définitions de canal est expliquée dans la rubrique Définition de canaux MQI.

Les protocoles de transmission possibles sont présentés dans le tableau suivant :

Tableau 1. Protocoles de transmission pour les canaux MQI

Plateforme client	LU 6.2	TCP/IP	NetBIOS	SPX
IBM i IBM i		Oui		
Linux AIX Systèmes AIX and Linux	Oui[1]	Oui		
Windows Windows	Oui	Oui	Oui	Oui

Remarque :

1. Linux LU6.2 n'est pas pris en charge sur les plateformes suivantes :
   - Linux (plateforme POWER)
   - Linux (plateforme x86-64)
   - Linux (plateforme zSeries s390x)

Protocoles de transmission - Combinaison de plateformes IBM MQ MQI client et serveur indique les combinaisons possibles de plateformes IBM MQ MQI client et serveur en utilisant ces protocoles de transmission.

Une application IBM MQ sur un IBM MQ MQI client peut utiliser tous les appels MQI de la même façon que lorsque le gestionnaire de files d'attente est local. **MQCONN** ou **MQCONNX** associe l'application IBM MQ au gestionnaire de files d'attente sélectionné, afin de créer un *descripteur de connexion*. Les autres appels utilisant ce descripteur de connexion sont ensuite traités par le gestionnaire de files d'attente connecté. La communication du IBM MQ MQI client nécessite une connexion active entre le client et le serveur,

par opposition à la communication entre les gestionnaires de files d'attente, qui est indépendante de la connexion et sans contrainte de temps.

Le protocole de communication est indiqué à l'aide de la définition de canal et n'a pas d'incidence sur l'application. Par exemple, une application Windows peut se connecter à un gestionnaire de files d'attente sur TCP/IP et à un autre gestionnaire de files d'attente sur NetBIOS.

Remarques sur les performances

Le protocole de communication que vous utilisez peut avoir une incidence sur les performances du système client et serveur IBM MQ. Dans certaines situations où la transmission est lente, vous pouvez utiliser la compression de canal IBM MQ .

Attribution de nom aux objets IBM MQ

La convention d'attribution de nom adoptée pour les objets IBM MQ dépend de l'objet. Le nom des machines et les ID utilisateur employés avec IBM MQ sont également soumis à certaines restrictions de dénomination.

Chaque instance d'un gestionnaire de files d'attente est désignée par son nom. Ce nom doit être unique au sein du réseau des gestionnaires de files d'attente interconnectés, de sorte qu'un gestionnaire de files d'attente puisse identifier clairement le gestionnaire de files d'attente cible auquel un message donné est envoyé.

Pour les autres types d'objet, un nom est associé à chaque objet qui peut être désigné par ce nom. Ces noms doivent être uniques au sein d'un gestionnaire de files d'attente et d'un type d'objet. Par exemple, vous pouvez avoir une file d'attente et un processus du même nom, mais vous ne pouvez pas avoir deux files d'attente du même nom.

Dans IBM MQ, les noms peuvent comporter au maximum 48 caractères, à l'exception des *canaux* qui peuvent comporter au maximum 20 caractères. Pour plus d'informations sur l'attribution de nom aux objets IBM MQ, voir «Règles d'appellation des objets IBM MQ», à la page 40.

Le nom des machines et les ID utilisateur employés avec IBM MQ sont également soumis à certaines restrictions de dénomination :

- Vérifiez que le nom du poste ne contient aucun espace. En effet, IBM MQ ne prend pas en charge les noms de postes comportant des espaces. Si toutefois vous installez IBM MQ sur un tel poste, vous ne pourrez pas créer de gestionnaire de files d'attente.
- En ce qui concerne les droits IBM MQ, les ID utilisateur et les noms de groupes ne doivent pas comporter plus de vingt caractères (sans espaces).
- Windows Un serveur IBM MQ for Windows ne prend pas en charge la connexion d'un IBM MQ MQI client si le client est exécuté sous un ID utilisateur qui contient le caractère @, par exemple, abc@d.

Concepts associés

«Noms de fichier IBM MQ», à la page 43
Chaque objet gestionnaire de files d'attente, file d'attente, définition de processus, liste de noms, canal, canal de connexion client, programme d'écoute, service et informations d'authentification IBM MQ est représenté par un fichier. Dans la mesure où les noms d'objet ne sont pas nécessairement des noms de fichier valides, le gestionnaire de files d'attente convertit, si nécessaire, le nom d'objet en nom de fichier valide.

Référence associée

«Règles d'appellation des objets IBM MQ», à la page 40
Les noms d'objet IBM MQ ont des longueurs maximales et tiennent compte de la casse. Les caractères ne sont pas tous pris en charge pour chaque type d'objet et la plupart des objets ont des règles concernant l'unicité des noms.

Règles d'appellation des objets IBM MQ

Les noms d'objet IBM MQ ont des longueurs maximales et tiennent compte de la casse. Les caractères ne sont pas tous pris en charge pour chaque type d'objet et la plupart des objets ont des règles concernant l'unicité des noms.

Il existe un grand nombre de types d'objets IBM MQ différents et les objets d'un certain type peuvent avoir tous le même nom, car ils existent dans des espaces nom d'objet différents. Par exemple, une file d'attente locale et un canal émetteur peuvent porter le même nom. Toutefois, un objet ne peut pas porter le même nom qu'un autre objet dans un même espace nom. Par exemple, une file d'attente locale ne peut pas porter le nom d'une file d'attente modèle et un canal émetteur ne peut pas porter le nom d'un canal récepteur.

Les objets IBM MQ suivants existent dans des espaces nom d'objet distincts :

- Informations d'authentification
- Canal
- Canal client
- Programme d'écoute
- Liste de noms
- Processus
- File d'attente
- Service
- Classe d'archivage
- Abonnement
- Topic

Nombre de caractères des noms d'objet

En règle générale, les noms d'objet IBM MQ peuvent contenir jusqu'à 48 caractères. Cette règle s'applique aux objets suivants :

- Informations d'authentification
- Cluster
- Programme d'écoute
- Liste de noms
- Définition de processus
- File d'attente
- Gestionnaire de files d'attente
- Service
- Abonnement
- Topic

Il existe des restrictions :

1. z/OS Sur les systèmes z/OS, les gestionnaires de files d'attente doivent comporter 4 caractères maximum et doivent être en majuscules et numériques.
2. Les noms d'objet de canal et de canal de connexion client peuvent comporter jusqu'à 20 caractères. Pour plus d'informations sur les canaux, voir Définition des canaux.
3. Les chaînes de rubrique peuvent avoir une longueur de 10 240 caractères maximum. Tous les noms d'objet IBM MQ tiennent compte de la casse.
4. Les noms d'abonnement peuvent avoir une longueur de 10 240 caractères maximum et contenir des espaces.

5. La longueur maximale des noms de classe de stockage est de 8 caractères.
6. La longueur maximale des noms de structure d'unité de couplage est de 12 caractères.

Caractères dans les noms d'objet

Caractères valides pour les noms d'objet IBM MQ :

Caractères	Restrictions
Majuscules A - Z	• Aucun
Minuscules a - z	• Dans les scripts MQSC, les noms en minuscules doivent être placés entre des guillemets simples. Cela empêche la conversion des caractères minuscules en majuscules. • Les systèmes qui utilisent EBCDIC Katakana ne peuvent pas utiliser les minuscules a-z dans les noms d'objet. • z/OS Il peut exister des restrictions lorsque vous utilisez des minuscules sur les systèmes z/OS. Par exemple, les noms de gestionnaire de files d'attente ne peuvent pas contenir des minuscules. • IBM i Sur les systèmes IBM i, lorsque des commandes CL sont utilisées, les noms en minuscules doivent être placés entre des guillemets simples. Cela empêche la conversion des caractères minuscules en majuscules.
Caractères numériques 0 - 9	• Aucun
le point (.)	• Aucun
le caractère de soulignement (_)	• Multi Aucun • z/OS Evitez d'utiliser des noms avec des caractères de soulignement de début et de fin, car les opérations et les panneaux de configuration IBM MQ for z/OS ne peuvent pas les traiter.
la barre oblique (/)	• Windows Sur les systèmes Windows, le premier caractère d'un nom de gestionnaire de files d'attente ne peut pas être une barre oblique. • IBM i Sur les systèmes IBM i, lorsque des commandes CL sont utilisées, les noms contenant une barre oblique doivent être placés entre des guillemets simples. • z/OS Aucun

Caractères	Restrictions
le symbole de pourcentage (%)	• ALW Aucun • z/OS Si vous utilisez RACF comme gestionnaire de sécurité pour IBM MQ for z/OS, n'utilisez pas % dans les noms d'objet, car les noms ne sont pas inclus dans les vérifications de sécurité lorsque des profils génériques RACF sont utilisés. • IBM i Sur les systèmes IBM i, lorsque des commandes CL sont utilisées, les noms contenant le symbole de pourcentage doivent être placés entre des guillemets simples.

Il existe des règles générales concernant les caractères dans les noms d'objet :

1. Les espaces de début ou imbriqués ne sont pas autorisés.
2. Les caractères en langue nationale ne sont pas autorisés.
3. Les noms dont la longueur est inférieure à la longueur totale de la zone peuvent être complétés sur la droite avec des espaces. Tous les noms courts retournés par le gestionnaire de files d'attente sont toujours complétés à droite avec des espaces.

Noms des files d'attente

Un nom de file d'attente a deux parties :

- Nom d'un gestionnaire de files d'attente
- Nom local de la file d'attente que connaît le gestionnaire de files d'attente

Chaque partie du nom de file d'attente comporte 48 caractères.

Pour faire référence à une file d'attente locale, vous pouvez omettre le nom du gestionnaire de files d'attente (en le remplaçant par des espaces ou en utilisant un caractère null de début). Toutefois, tous les noms de file d'attente retournés à un programme par IBM MQ contiennent le nom du gestionnaire de files d'attente.

z/OS Une file d'attente partagée, accessible à n'importe quel gestionnaire de files d'attente dans son groupe de partage de files d'attente, ne peut pas porter le nom d'une file d'attente local non partagée dans le même groupe de partage de files d'attente. Cette restriction empêche une application d'ouvrir par erreur une file d'attente partagée en pensant ouvrir une file d'attente locale ou inversement. Les files d'attente partagées et les groupes de partage de files d'attente sont disponibles uniquement dans IBM MQ for z/OS.

Pour faire référence à une file d'attente éloignée, un programme doit inclure le nom du gestionnaire de files d'attente dans le nom de file d'attente complet ou il doit exister une définition locale de la file d'attente éloignée.

Lorsqu'une application utilise un nom de file d'attente, ce nom peut être un nom de file d'attente locale (ou un alias d'une telle file d'attente) ou le nom d'une définition locale d'une file d'attente éloignée, mais l'application n'a pas besoin de les distinguer, sauf si elle doit obtenir un message de la file d'attente (lorsque la file d'attente doit être locale). Lorsque l'application ouvre l'objet file d'attente, l'appel MQOPEN exécute une résolution de nom pour déterminer la file d'attente dans laquelle elle doit exécuter les opérations suivantes. Cela implique que l'application n'a pas de dépendance intégrée dans les files d'attente définies dans des emplacements donnés dans un réseau de gestionnaires de files d'attente. Par conséquent, si un administrateur système déplace les files d'attente dans un réseau et modifie leur définition, les applications qui utilisent ces files d'attente n'ont pas besoin d'être modifiées.

Noms d'objet réservés

Les noms d'objet commençant par SYSTEM. sont réservés aux objets définis par le gestionnaire de files d'attente. Vous pouvez utiliser les commandes **Alter**, **Define** et **Replace** pour changer ces définitions d'objet en fonction de l'installation. Les noms qui sont définis pour IBM MQ sont affichés en entier dans Noms de file d'attente.

z/OS Sur IBM MQ for z/OS, le nom de la structure d'application de l'unité de couplage CSQSYSAPPL est réservé.

Concepts associés

Nom d'installation sous AIX, Linux, and Windows

Noms de fichier IBM MQ

Chaque objet gestionnaire de files d'attente, file d'attente, définition de processus, liste de noms, canal, canal de connexion client, programme d'écoute, service et informations d'authentification IBM MQ est représenté par un fichier. Dans la mesure où les noms d'objet ne sont pas nécessairement des noms de fichier valides, le gestionnaire de files d'attente convertit, si nécessaire, le nom d'objet en nom de fichier valide.

Le chemin d'accès par défaut à un répertoire de gestionnaires de files d'attente est le suivant :

- Un préfixe, défini dans les informations de configuration IBM MQ :
  - Linux AIX Sous AIX and Linux, le préfixe par défaut est `/var/mqm`. Celui-ci est configuré dans la section `DefaultPrefix` du fichier de configuration mqs.ini.
  - Windows Sur les systèmes Windows 32 bits, le préfixe par défaut est `C:\Program Files (x86)\IBM\WebSphere MQ`. Sur les systèmes Windows 64 bits, le préfixe par défaut est `C:\Program Files\IBM\MQ`. Pour les installations 32 bits et 64 bits, les répertoires de données sont installés dans `C:\ProgramData\IBM\MQ`. Celui-ci est configuré dans la section `DefaultPrefix` du fichier de configuration mqs.ini.

  Le cas échéant, le préfixe peut être modifié depuis la page de propriétés IBM MQ de l'explorateur IBM MQ, sinon, éditez le fichier de configuration `mqs.ini` manuellement.
- Le nom du gestionnaire de files d'attente est converti en nom de répertoire valide. Par exemple, le gestionnaire de files d'attente :

  ```
  queue.manager
  ```

  est représenté sous forme de :

  ```
  queue!manager
  ```

  Ce processus est appelé *transformation de noms*.

Dans IBM MQ, vous pouvez attribuer aux gestionnaires de files d'attente un nom contenant jusqu'à 48 caractères.

Par exemple, vous pouvez attribuer à un gestionnaire de files d'attente le nom suivant :

```
QUEUE.MANAGER.ACCOUNTING.SERVICES
```

Cependant, chaque gestionnaire de files d'attente est représenté par un fichier et la longueur maximale d'un nom de fichier est limitée, ainsi que le nombre de caractères pouvant être utilisés dans le nom. En conséquence, les noms de fichiers représentant des objets sont automatiquement convertis en fonction des exigences du système de fichiers.

Les règles régissant la transformation d'un nom de gestionnaire de files d'attente sont les suivantes :

1. Transformation de caractères individuels :
   - Depuis . à !
   - Depuis / vers &
2. Si le nom n'est toujours pas valide :
   a. Tronquez-le à huit caractères
   b. Ajoutez un suffixe numérique de trois caractères

Par exemple, en utilisant le préfixe par défaut et un gestionnaire de files d'attente intitulé `queue.manager` :

- Windows Sous Windows doté du système NTFS ou FAT32, le nom du gestionnaire de files d'attente devient :

```
C:\Program Files\IBM\MQ\qmgrs\queue!manager
```

- Windows Sous Windows doté du système FAT, le nom du gestionnaire de files d'attente devient :

```
C:\Program Files\IBM\MQ\qmgrs\queue!ma
```

- Linux AIX Sous AIX and Linux, le nom du gestionnaire de files d'attente devient :

```
/var/mqm/qmgrs/queue!manager
```

L'algorithme de transformation fait également la distinction entre les noms dont seule la casse diffère sur les systèmes de fichiers qui ne sont pas sensibles à la casse.

Transformation de nom d'objet

Les noms d'objet ne sont pas nécessairement des noms de système de fichiers valides. Vous devrez peut-être convertir les noms d'objet. La méthode utilisée est différente de celle des noms de gestionnaire de files d'attente car, bien qu'il n'existe que quelques noms de gestionnaire de files d'attente sur chaque machine, il peut y avoir un grand nombre d'autres objets pour chaque gestionnaire de files d'attente. Les objets files d'attente, définitions de processus, listes de noms, canaux, canaux de connexion client, programmes d'écoute et informations d'authentification sont représentés dans le système de fichiers.

Lorsqu'un nouveau nom est généré par le processus de transformation, il n'y a aucune relation simple avec le nom d'objet d'origine. Vous pouvez utiliser la commande **dspmqfls** pour la transformation entre les noms d'objet réels et convertis.

Référence associée
dspmqfls (affichage des noms de fichier)
Information associée
Strophe AllQueueManagers du fichier mqs.ini

IBM i Noms d'objet sur IBM i

Un gestionnaire de files d'attente est associé à une bibliothèque de gestionnaires de files d'attente qui possède un nom unique. Il se peut que les noms de gestionnaire de files d'attente et les noms d'objet doivent être transformés pour répondre aux exigences du système de fichiers intégré (IFS) d'IBM i.

Lorsqu'un gestionnaire de files d'attente est créé, IBM MQ lui associe une bibliothèque de gestionnaire de files d'attente. Un nom unique est attribué à cette bibliothèque de gestionnaire de files d'attente ; il comporte au maximum 10 caractères et est basé en grande partie sur le nom de gestionnaire de files d'attente défini par l'utilisateur. Le gestionnaire de files d'attente et la bibliothèque de gestionnaire de files d'attente sont placés dans un répertoire qui est également basé sur le nom de gestionnaire de files

d'attente à l'aide du préfixe /QIBM/UserData/mqm. Voici un exemple de gestionnaire de files d'attente, de bibliothèque de gestionnaire de files d'attente et de répertoire :

Nom gest. de files	ORANGE
Nom de bibliothèque du gestionnaire de files d'attente	QMORANGE
Répertoire	/QIBM/UserData/mqm/ORANGE

Tous les noms de gestionnaire de files d'attente et bibliothèques de gestionnaire de files d'attente sont écrits dans les strophes du fichier /QIBM/UserData/mqm/mqs.ini.

Répertoires et fichiers IFS d'IBM MQ

IFS (IBM i Integrated File System) est largement utilisé par IBM MQ pour stocker des données. Pour plus d'informations sur le système IFS, consultez le document *Integrated File System Introduction*.

Chaque objet IBM MQ, tel qu'un canal ou une file d'attente, est représenté par un fichier. Dans la mesure où les noms d'objet ne sont pas nécessairement des noms de fichier valides, le gestionnaire de files d'attente convertit, si nécessaire, le nom d'objet en nom de fichier valide.

Le chemin d'accès à un répertoire de gestionnaires de files d'attente est formé à partir des éléments suivants :

- Un préfixe, qui est défini dans le fichier de configuration du gestionnaire de files d'attente, qm.ini. Le préfixe par défaut est /QIBM/UserData/mqm.
- Un littéral, qmgrs.
- Un nom du gestionnaire de files d'attente codé, qui correspond au nom de gestionnaire de files d'attente converti en nom de répertoire valide. Par exemple, le gestionnaire de files d'attente queue/manager est représenté par queue&manager.

Ce processus est désigné par transformation de noms.

Transformation de noms de gestionnaire de files d'attente IFS

Dans IBM MQ, vous pouvez attribuer aux gestionnaires de files d'attente un nom contenant jusqu'à 48 caractères.

Par exemple, vous pouvez attribuer à un gestionnaire de files d'attente le nom QUEUE/MANAGER/ACCOUNTING/SERVICES. A l'instar d'une bibliothèque créée pour chaque gestionnaire de files d'attente, chaque gestionnaire de files d'attente est également représenté par un fichier. En raison de points de code différents dans EBCDIC, le nombre de caractères pouvant être utilisés dans le nom est limité. En conséquence, les noms de fichiers IFS représentant des objets sont automatiquement convertis en fonction des exigences du système de fichiers.

À partir de l'exemple d'un gestionnaire de files d'attente portant le nom queue/manager, transformant le caractère / en & et en supposant le préfixe par défaut, le nom du gestionnaire de files d'attente dans IBM MQ for IBM i devient /QIBM/UserData/mqm/qmgrs/queue&manager.

Transformation de nom d'objet

Les noms d'objet ne sont pas nécessairement des noms de système de fichiers valides ; par conséquent, il est peut-être nécessaire de convertir les noms d'objet. La méthode utilisée est différente de celle des noms de gestionnaire de files d'attente car, bien qu'il n'existe que quelques noms de gestionnaire de files d'attente pour chaque machine, il peut y avoir un grand nombre d'autres objets pour chaque gestionnaire de files d'attente. Seules les définitions de processus, les files d'attente et les listes de noms sont représentées dans le système de fichiers ; ces critères n'ont pas d'incidence sur les canaux.

Lorsqu'un nouveau nom est généré par le processus de transformation, il n'y a aucune relation simple avec le nom d'objet d'origine. Vous pouvez utiliser la commande DSPMQMOBJN pour afficher les noms convertis des objets IBM MQ.

Mise en file d'attente répartie et clusters

La mise en file d'attente répartie permet d'envoyer des messages d'un gestionnaire de files d'attente à un autre. Le gestionnaire de files d'attente récepteur peut résider sur la même machine ou sur une autre, à proximité ou à l'autre bout du monde. Il peut s'exécuter sur la même plateforme que le gestionnaire de files d'attente local ou se trouver sur n'importe laquelle des plateformes prises en charge par IBM MQ. Vous pouvez définir manuellement toutes les connexions dans un environnement de mise en file d'attente répartie, ou créer un cluster et laisser IBM MQ définir la plupart des informations de connexion automatiquement.

Files d'attente réparties

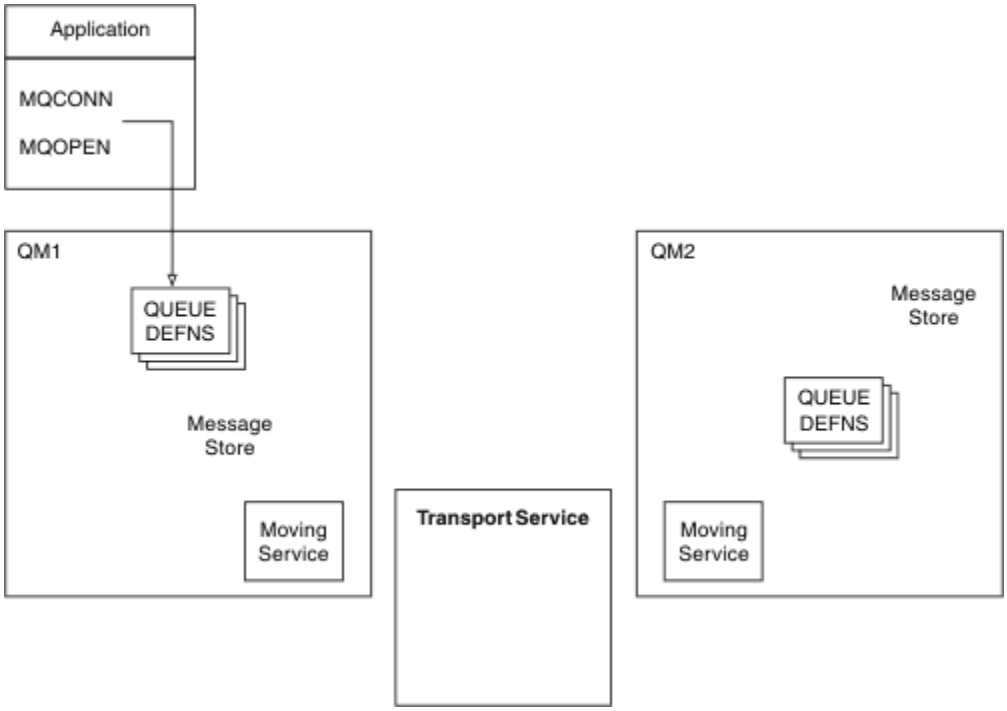


Figure 4. Présentation des composants de la mise en file d'attente répartie

Dans la figure précédente :

- Une application utilise l'appel MQCONN pour se connecter à un gestionnaire de files d'attente. L'application utilise ensuite l'appel MQOPEN pour ouvrir une file d'attente de sorte qu'elle puisse y insérer des messages.
- Chaque gestionnaire de files d'attente comporte une définition pour chacune de ses files d'attente. Il peut avoir des définitions de *files d'attente locales* (c'est-à-dire hébergées par ce gestionnaire de files d'attente) et des définitions de *files d'attente éloignées* (c'est-à-dire, hébergées par d'autres gestionnaires de files d'attente).
- Si les messages sont destinés à une file d'attente éloignée, le gestionnaire de files d'attente local les conserve dans une *file d'attente de transmission*, qui les maintient dans un emplacement de stockage des messages jusqu'à ce qu'ils puissent être transmis au gestionnaire de files d'attente éloignées.
- Chaque gestionnaire de files d'attente contient un logiciel de communication appelé *service de déplacement*, utilisé par le gestionnaire de files d'attente pour communiquer avec d'autres gestionnaires de files d'attente.
- Le *service de transfert* est indépendant du gestionnaire de files d'attente et peut correspondre à l'un des suivants (en fonction de la plateforme) :

- Communication avancée de programme à programme SNA (Systems Network Architecture)
- protocole TCP/IP
- Système NetBIOS (Network Basic Input/Output System)
- SPX (Sequenced Packet Exchange)

Composants requis pour l'envoi d'un message

Si un message doit être envoyé à un gestionnaire de files d'attente éloignées, le gestionnaire de files d'attente local a besoin des définitions pour une *file d'attente de transmission* et un *canal*. Un canal est une liaison de communication unidirectionnelle entre deux gestionnaires de files d'attente. Il peut transporter des messages destinés à n'importe quel nombre de files d'attente sur le gestionnaire de files d'attente éloignées.

Chaque extrémité d'un canal comporte une définition distincte qui la définit, par exemple, comme extrémité émettrice ou extrémité réceptrice. Un canal simple est constitué d'une définition de canal *émetteur* sur le gestionnaire de files d'attente local et d'une définition de canal *récepteur* sur le gestionnaire de files d'attente éloignées. Ces deux définitions doivent porter le même nom et former conjointement un canal.

Le logiciel traitant l'envoi et la réception des messages est appelé *agent MCA* (Message Channel Agent). Un *Agent MCA* se trouve à chaque extrémité d'un canal.

Chaque gestionnaire de files d'attente doit posséder une *file d'attente de rebut* (également appelée *file d'attente des messages non livrés*). Les messages sont insérés dans cette file d'attente s'ils ne peuvent pas être distribués à leur destination.

La figure suivante affiche la relation entre les gestionnaires de files d'attente, les files d'attente de transmission, les canaux et les agents MCA :

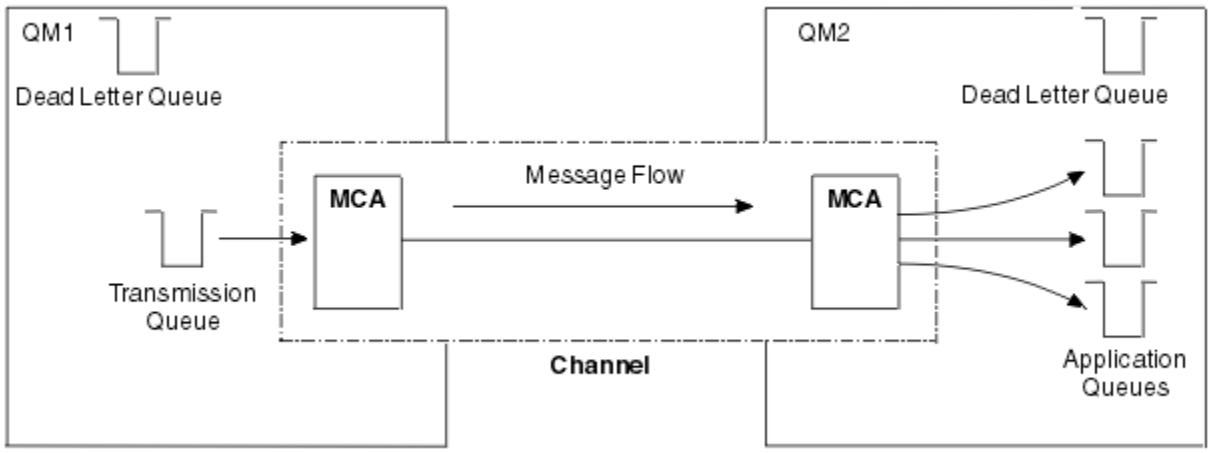


Figure 5. Envoi de messages

Composants requis pour le renvoi d'un message

Si votre application requiert le renvoi des messages à partir du gestionnaire de files d'attente éloignées, vous devez définir un autre canal à exécuter dans le sens opposé entre les gestionnaires de files d'attente, comme présenté dans la figure suivante :

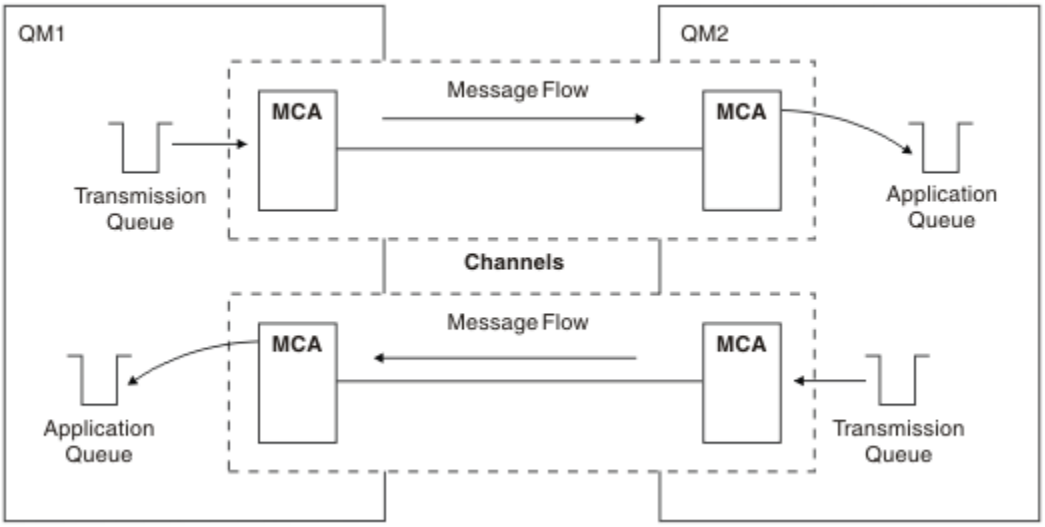


Figure 6. Envoi de messages dans les deux sens

Clusters

Au lieu de définir manuellement toutes les connexions dans un environnement de mise en file d'attente répartie, vous pouvez regrouper un ensemble de gestionnaires de files d'attente dans un cluster. Lors de cette opération, les gestionnaires de files d'attente peuvent rendre les files d'attente qu'ils hébergent disponibles pour tous les autres gestionnaires de files d'attente du cluster sans avoir besoin de définitions de canal explicites, des définitions de file d'attente éloignée ou des files d'attente de transmission de chaque destination. Chaque gestionnaire de files d'attente d'un cluster dispose d'une file d'attente de transmission unique qui transmet des messages à tout autre gestionnaire de files d'attente du cluster. Pour chaque gestionnaire de files d'attente, il suffit de définir un seul canal récepteur de cluster et un seul canal émetteur de cluster ; tous les autres canaux sont automatiquement gérés par le cluster.

Un client IBM MQ peut se connecter à un gestionnaire de files d'attente membre d'un cluster, comme à n'importe quel autre gestionnaire de files d'attente. Comme dans le cas de la mise en file d'attente répartie configurée manuellement, vous utilisez l'appel MQPUT pour insérer un message dans une file d'attente de n'importe quel gestionnaire de files d'attente. Vous utilisez l'appel MQGET pour extraire des messages d'une file d'attente locale.

Les gestionnaires de files d'attente sur les plateformes qui prennent en charge les clusters n'ont pas besoin d'être membres d'un cluster. Vous pouvez également continuer de configurer manuellement la mise en file d'attente répartie et l'utiliser en plus ou à la place des clusters.

Avantages de l'utilisation de clusters

La mise en cluster offre deux avantages essentiels :

- Les clusters simplifient l'administration de réseaux IBM MQ qui nécessitent généralement la configuration de nombreuses définitions d'objet pour les canaux, les files d'attente de transmission et les files d'attente éloignées. C'est en particulier le cas pour des réseaux de grande taille susceptibles d'être modifiés dans lesquels de nombreux gestionnaires de files d'attente doivent être interconnectés. Cette architecture est particulièrement difficile à configurer et gérer activement.
- Des clusters peuvent être utilisés pour répartir la charge de travail du trafic des messages entre les files d'attente et les gestionnaires de files d'attente d'un cluster. Vous pouvez ainsi répartir la charge de travail liée aux messages d'une seule file d'attente sur des instances équivalentes de celle-ci située sur plusieurs gestionnaires de files d'attente. Cette répartition de la charge de travail permet d'obtenir une résilience accrue lors de pannes système et d'améliorer les performances d'évolutivité des flux de messages particulièrement actifs dans un système. Dans un tel environnement, chacune des instances des files d'attente réparties dispose d'applications consommatrices qui traitent les messages. Pour plus d'informations, voir Utilisation des clusters pour la gestion de la charge de travail.

Procédure de routage des messages dans un cluster

Un cluster s'apparente à un réseau de gestionnaires de files d'attente géré par un administrateur système consciencieux. Lorsque vous définissez une file d'attente de cluster, l'administrateur système crée automatiquement des définitions de files d'attente éloignées de manière appropriée dans les autres gestionnaires de files d'attente.

Vous n'avez pas à créer de définitions de files d'attente de transmission, car IBM MQ fournit une file d'attente de transmission dans chaque gestionnaire de files d'attente du cluster. Cette file d'attente de transmission unique peut être utilisée pour transporter les messages vers n'importe quelle autre file d'attente du cluster. Vous n'êtes pas limité à l'utilisation d'une file d'attente de transmission unique. Un gestionnaire de files d'attente peut utiliser plusieurs files d'attente de transmission pour séparer les messages insérés dans chaque gestionnaire de files d'attente d'un cluster. En règle générale, un gestionnaire de files d'attente utilise une file d'attente de transmission de cluster unique. Vous pouvez modifier l'attribut de gestionnaire de files d'attente DEFCLXQ afin qu'un gestionnaire de files d'attente utilise une file d'attente de transmission de cluster différente pour chaque gestionnaire de files d'attente d'un cluster. Vous pouvez également définir les files d'attente de transmission de cluster manuellement.

Tous les gestionnaires de files d'attente qui rejoignent un cluster acceptent ce mode de fonctionnement. Ils envoient des informations sur eux-mêmes et sur les files d'attente qu'ils hébergent et ils reçoivent des informations sur les autres membres du cluster.

Pour vous assurer qu'aucune information n'est perdue lorsqu'un gestionnaire de files d'attente devient indisponible, vous désignez deux gestionnaires de files d'attente du cluster afin qu'ils agissent en tant que *référentiels complets*. Ces gestionnaires de files d'attente stockent un ensemble complet d'informations sur tous les gestionnaires de files d'attente et files d'attente du cluster. Les autres gestionnaires de files d'attente du cluster stockent uniquement les informations sur les gestionnaires de files d'attente et les files d'attente avec lesquels ils échangent des messages. Ces gestionnaires de files d'attente sont appelés *référentiels partiels*. Pour plus d'informations, voir «Référentiel de cluster», à la page 61.

Pour qu'un gestionnaire de files d'attente puisse appartenir à un cluster, le gestionnaire de files d'attente doit disposer de deux canaux, un canal émetteur de cluster et un canal récepteur de cluster :

- Un canal émetteur de cluster est un canal de communication, à l'instar d'un canal émetteur. Vous devez créer manuellement un canal émetteur de cluster dans un gestionnaire de files d'attente pour le connecter à un référentiel complet déjà membre du cluster.
- Un canal récepteur de cluster est un canal de communication, à l'instar d'un canal récepteur. Vous devez créer manuellement un canal récepteur de cluster. Le canal fait office de mécanisme pour que le gestionnaire de files d'attente puisse recevoir les communications du cluster.

Tous les autres canaux nécessaires pour les communications entre ce gestionnaire de files d'attente et d'autres membres du cluster sont créés automatiquement.

La figure suivante représente les composants d'un cluster nommé CLUSTER :

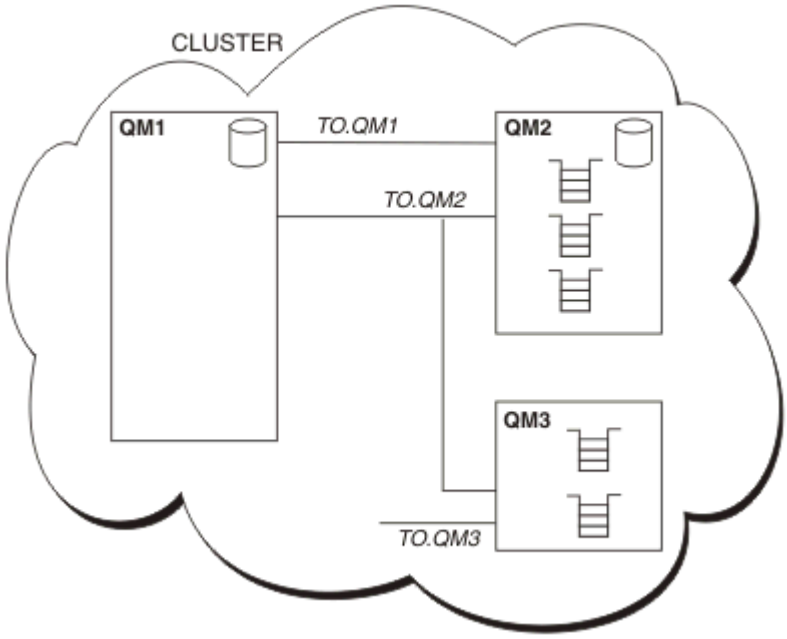


Figure 7. Cluster de gestionnaires de files d'attente

- CLUSTER contient trois gestionnaires de files d'attente : QM1, QM2 et QM3.
- QM1 et QM2 héberge des référentiels complets d'informations relatives aux gestionnaires de files d'attente et aux files d'attente du cluster.
- QM2 et QM3 héberge certaines files d'attente de cluster, c-à-d, des files d'attente accessibles à n'importe quel autre gestionnaire de files d'attente du cluster.
- Chaque gestionnaire de files d'attente comporte un canal récepteur de cluster appelé TO.qmgr sur lequel il peut recevoir des messages.
- Chaque gestionnaire de files d'attente comporte également un canal émetteur de cluster sur lequel il peut envoyer des informations à l'un des gestionnaires de files d'attente de référentiel.
- QM1 et QM3 envoient des informations au référentiel de QM2 et ce dernier envoie des informations au référentiel de QM1.

Composants de la mise en file d'attente répartie

Les composants de la mise en file d'attente répartie sont les canaux de message, les agents MCA, les files d'attente de transmission, les initiateurs et programmes d'écoute de canal, et les programmes d'exit de canal. La définition de chaque extrémité d'un canal de transmission de messages peut être de différents types.

Les canaux de transmission de messages sont les canaux qui transfèrent des messages d'un gestionnaire de files d'attente vers un autre. Ne confondez pas les canaux de transmission de messages avec les canaux MQI. Il existe deux types de canal MQI : connexion serveur (SVRCONN) et connexion client (CLNTCONN). Pour plus d'informations, voir Canaux.

La définition de chaque extrémité d'un canal de transmission de messages peut être de l'un des types suivants :

- Emetteur (SDR)
- Récepteur (RCVR)
- Serveur (SVR)
- Demandeur (RQSTR)

- Emetteur de cluster (CLUSSDR)
- Récepteur de cluster (CLUSRCVR)

Un canal de transmission de messages est défini à l'aide d'un de ces types définis sur une extrémité et d'un type compatible à l'autre extrémité. Les combinaisons possibles sont les suivantes :

- Emetteur-récepteur
- Demandeur-serveur
- Demandeur-émetteur (rappel)
- Serveur-récepteur
- Emetteur de cluster-récepteur de cluster

Les instructions détaillées de création d'un canal émetteur-récepteur figurent dans Définition des canaux. Pour consulter des exemples de paramètres requis pour la configuration des canaux émetteurs-récepteurs, consultez la rubrique Exemples d'informations de configuration applicable à votre plateforme. Pour les paramètres requis pour la définition d'un canal de n'importe quel type, voir DEFINE CHANNEL.

Canaux émetteur-récepteur

Un émetteur sur un système démarre le canal de sorte qu'il puisse envoyer des messages à l'autre système. L'émetteur demande au récepteur, situé à l'autre extrémité du canal, de démarrer. L'émetteur envoie des messages de sa file d'attente de transmission au récepteur. Ce dernier insère les messages dans la file d'attente de destination. Ce processus est illustré dans la Figure 8, à la page 51.

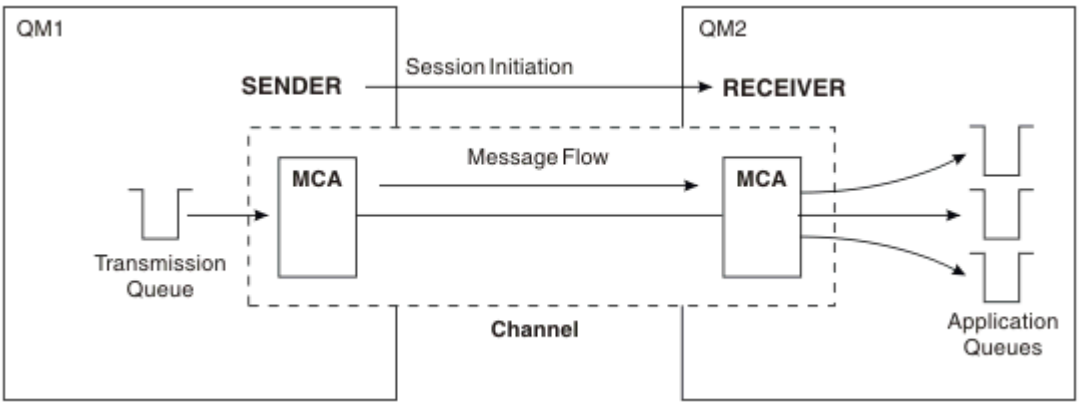


Figure 8. Un canal émetteur-récepteur

Canaux demandeur-serveur

Un demandeur sur un système démarre le canal de sorte qu'il puisse recevoir des messages de l'autre système. Le demandeur demande au serveur, situé à l'autre extrémité du canal, de démarrer. Le serveur envoie des messages au demandeur à partir de la file d'attente de transmission définie dans sa définition de canal.

Un canal serveur peut également lancer la communication et envoyer des messages à un demandeur. Cela ne s'applique qu'aux serveurs *entièrement qualifiés*, à savoir les canaux serveur qui portent le nom de connexion du partenaire spécifié dans la définition de canal. Un serveur entièrement qualifié peut être démarré par un demandeur ou peut lancer une communication avec un demandeur.

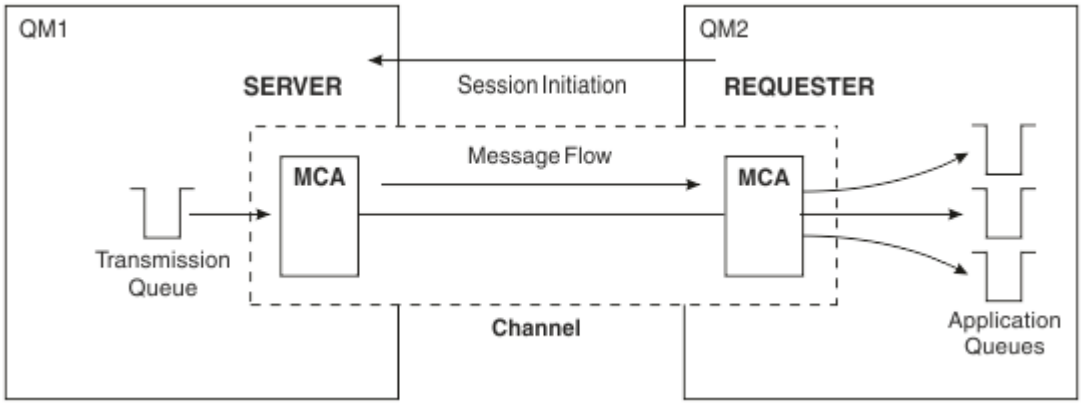


Figure 9. Un canal demandeur-serveur

Canaux demandeur-émetteur

Le demandeur démarre le canal et l'émetteur met fin à l'appel. L'émetteur redémarre ensuite la communication en fonction des informations figurant dans sa définition de canal (connue sous le nom de *rappel*). Il envoie des messages de la file d'attente de transmission au demandeur.

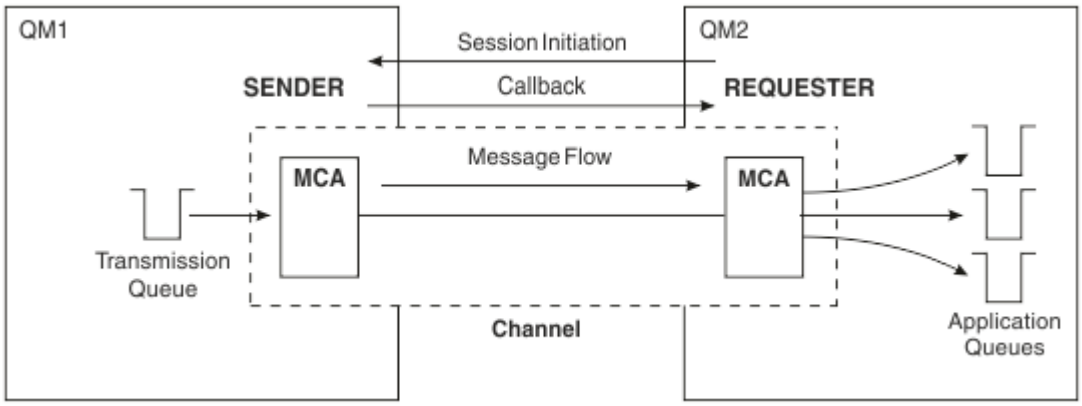


Figure 10. Un canal demandeur-émetteur

Canaux serveur-récepteur

Ce type de canal est similaire au canal émetteur-récepteur mais s'applique uniquement aux serveurs *entièrement qualifiés*, à savoir les canaux serveur qui portent le nom de connexion du partenaire spécifié dans la définition de canal. Le démarrage du canal doit être déclenché à l'extrémité serveur de la liaison. L'illustration de ce processus est similaire à celle de la Figure 8, à la page 51.

Canaux émetteurs de cluster

Dans un cluster, chaque gestionnaire de files d'attente comporte un canal émetteur de cluster sur lequel il peut envoyer des informations de cluster à l'un des gestionnaires de files d'attente de référentiel complet. Les gestionnaires de files d'attente peuvent également envoyer des messages à d'autres gestionnaires de files d'attente sur des canaux émetteurs de cluster.

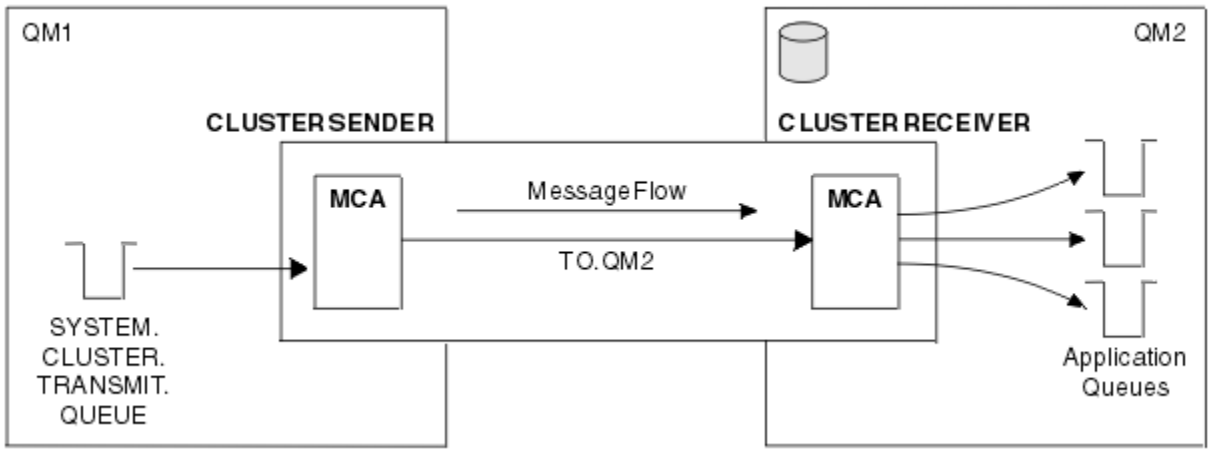


Figure 11. Un canal émetteur de cluster

Canaux récepteurs de cluster

Dans un cluster, chaque gestionnaire de files d'attente comporte un canal récepteur de cluster sur lequel il peut recevoir des messages et des informations concernant le cluster. L'illustration de ce processus est similaire à celle de la Figure 11, à la page 53.

Files d'attente de rebut

La file d'attente de rebut (ou file d'attente des messages non livrés) est la file d'attente à laquelle des messages sont envoyés s'ils ne peuvent pas être acheminés vers la destination appropriée. Chaque gestionnaire de files d'attente a généralement une file d'attente de rebut.

Une *file d'attente de rebut* (DLQ), parfois appelée *file d'attente de messages non livrés*, est une file d'attente stockant des messages qui ne peuvent pas être livrés à leurs files d'attente de destination, par exemple parce que la file d'attente n'existe pas ou qu'elle est saturée. Les files d'attente de rebut sont également utilisées à l'extrémité d'émission d'un canal pour les erreurs de conversion de données. Chaque gestionnaire de files d'attente d'un réseau dispose généralement d'une file d'attente locale à utiliser comme file d'attente de rebut. Ainsi, les messages non distribuables à la destination appropriée pourront être stockés puis récupérés ultérieurement.

Les messages peuvent être placés dans la file d'attente de rebut par des gestionnaires de files d'attente, de agents MCA et des applications. Tous les messages de la file d'attente de rebut doivent être précédés d'une structure d'*en-tête de non distribution*, MQDLH. La zone *Reason* de la structure MQDLH contient un code anomalie indiquant la raison pour laquelle le message se trouve dans la file d'attente de rebut.

Il est conseillé de définir une file d'attente de rebut pour chaque gestionnaire de files d'attente. Si vous ne le faites pas et que l'agent MCA ne peut pas insérer un message, celui-ci reste dans la file d'attente de transmission et le canal s'arrête. De même, si des messages rapides non persistants (voir Messages rapides non persistants) ne peuvent pas être distribués et qu'aucune file d'attente de rebut n'existe sur le système cible, ces messages sont supprimés.

Cependant, les files d'attente de rebut peuvent avoir une incidence sur la séquence de distribution des messages et, par conséquent, vous pouvez choisir de ne pas les utiliser.

Tâches associées

Utilisation des files d'attente de rebut

Traitement des incidents liés aux messages non distribués

Référence associée

runmqdlq (exécution du gestionnaire de files d'attente de rebut)

Définition de files d'attente éloignées

Les définitions de file d'attente éloignée sont des définitions destinées aux files d'attente détenues par un autre gestionnaire de files d'attente.

Les applications peuvent extraire des messages uniquement des files d'attente locales, mais elles peuvent insérer des messages dans des files d'attente locales ou des files d'attente éloignées. Par conséquent, en sus d'une définition pour chacune de ses files d'attente locales, un gestionnaire de files d'attente peut comporter des *définitions de file d'attente éloignée*. L'avantage des définitions de file d'attente éloignée est qu'elles permettent à une application d'insérer un message dans une file d'attente éloignée sans avoir à indiquer le nom de la file d'attente éloignée ou du gestionnaire de files d'attente éloignées ou le nom de la file d'attente de transmission. Grâce aux définitions de file d'attente éloignée, vous n'êtes pas tributaire des emplacements.

Les définitions de file d'attente éloignée peuvent être utilisées à d'autres fins, comme décrit ultérieurement.

Mode d'accès au gestionnaire de files d'attente éloignées

Il se peut que vous n'ayez pas toujours un canal entre chaque gestionnaire de files d'attente source et cible. Il existe un certain nombre d'autres méthodes de liaison entre les deux, y compris le mode multi-tronçons, le partage de canaux, l'utilisation de canaux différents et la mise en cluster.

Multi-tronçons

S'il n'y pas de liaison de communication directe entre le gestionnaire de files d'attente source et le gestionnaire de files d'attente cible, il est possible de traverser un ou plusieurs *gestionnaires de files d'attente intermédiaires* sur le chemin vers le gestionnaire de files d'attente cible. C'est ce qu'on appelle *multi-tronçons*.

Vous définissez des canaux entre tous les gestionnaires de files d'attente, ainsi que des files d'attente de transmission sur les gestionnaires de files d'attente intermédiaires. Cela est illustré dans la Figure 12, à la page 54.

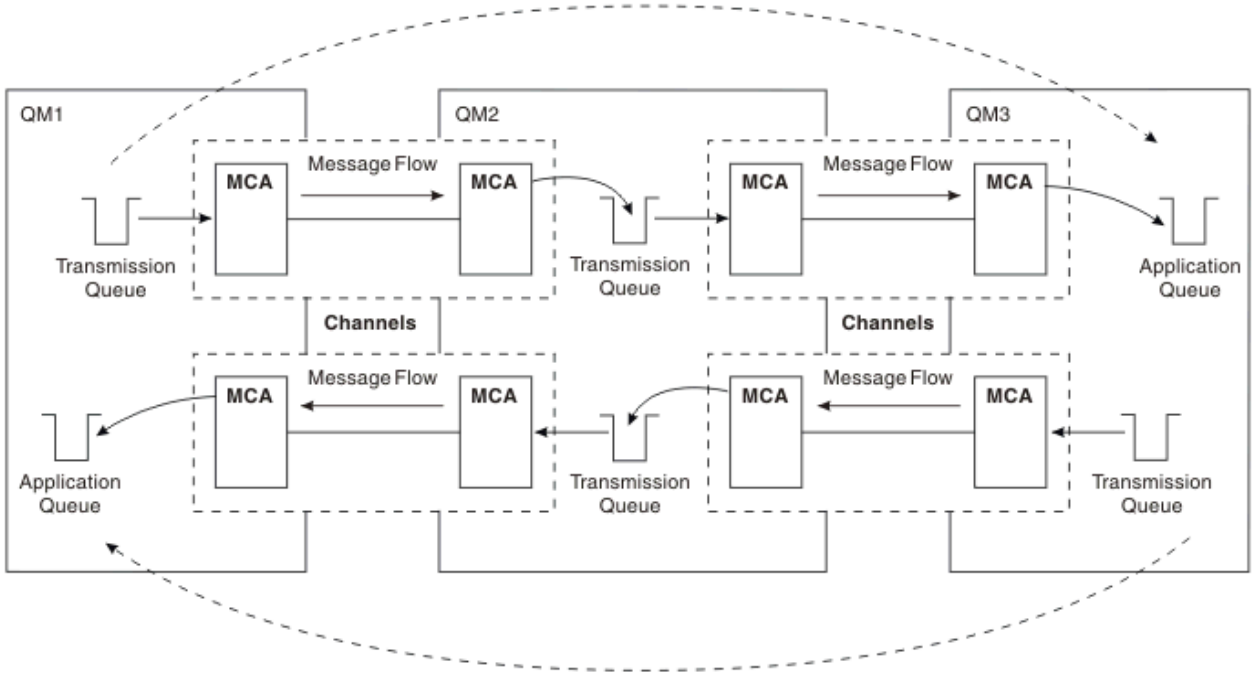


Figure 12. Traversée de gestionnaires de files d'attente intermédiaires

Partage de canaux

En tant que concepteur d'application, vous avez le choix de forcer vos applications à spécifier le nom du gestionnaire de files d'attente éloignées ainsi que le nom de la file d'attente, ou de créer une *définition de file d'attente éloignée* pour chaque file d'attente éloignée. Cette définition contient le nom du gestionnaire de files d'attente éloignées, le nom de la file d'attente et le nom de la file d'attente de transmission. Quelle que soit l'opération que vous choisissez, tous les messages provenant de toutes les applications adressant des files d'attente dans le même emplacement éloigné sont envoyés via la même file d'attente de transmission. Cela est illustré dans la Figure 13, à la page 55.

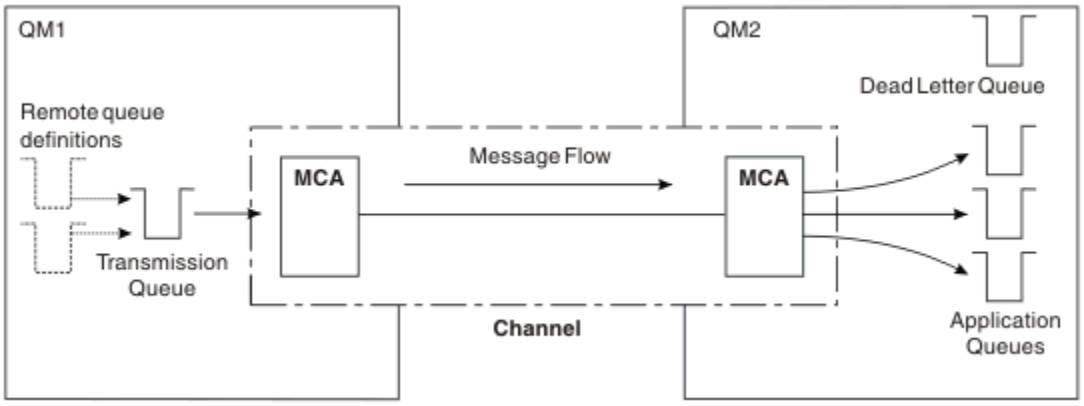


Figure 13. Partage d'une file d'attente de transmission

La Figure 13, à la page 55 indique que les messages envoyés de plusieurs applications à plusieurs files d'attente éloignées peuvent utiliser le même canal.

Utilisation de canaux différents

Si vous avez des messages de types différents à envoyer entre deux gestionnaires de files d'attente, vous pouvez définir plusieurs canaux entre les deux. Vous aurez parfois besoin d'autres canaux, peut-être pour des raisons de sécurité ou pour sacrifier la vitesse de distribution au profit d'un trafic de messages extrêmement élevé.

Pour configurer un deuxième canal, vous devez définir un autre canal et une autre file d'attente de transmission, puis créer une définition de file d'attente éloignée indiquant l'emplacement et le nom de la file d'attente de transmission. Vos applications peuvent ensuite utiliser l'un ou l'autre des canaux mais les messages sont toujours distribués aux mêmes files d'attente cible. Cela est illustré dans la Figure 14, à la page 55.

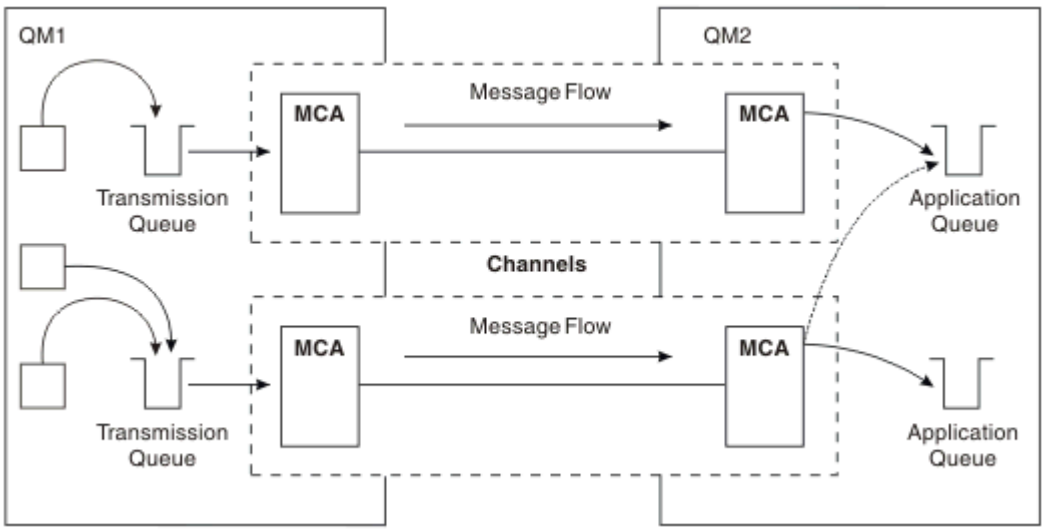


Figure 14. Utilisation de plusieurs canaux

Lorsque vous utilisez des définitions de file d'attente éloignée pour spécifier une file d'attente de transmission, vos applications ne doivent **pas** elles-mêmes spécifier l'emplacement (c-à-d, le gestionnaire de files d'attente de destination). Si elles le spécifient, le gestionnaire de files d'attente n'utilise pas les définitions de file d'attente éloignée. Grâce aux définitions de file d'attente éloignée, vous n'êtes pas tributaire des emplacements. Les applications peuvent insérer des messages dans une file d'attente *logique* sans connaître l'emplacement de la file d'attente et vous pouvez modifier la file d'attente *physique* sans avoir à modifier vos applications.

Utilisation de la mise en cluster

Chaque gestionnaire de files d'attente dans un cluster définit un canal récepteur de cluster. Lorsqu'un autre gestionnaire de files d'attente souhaite envoyer un message à ce gestionnaire de files d'attente, il définit automatiquement le canal émetteur de cluster correspondant. Par exemple, si un cluster contient plusieurs instances d'une file d'attente, le canal émetteur de cluster peut être définir sur n'importe quel gestionnaire de files d'attente hébergeant la file d'attente. IBM MQ utilise un algorithme de gestion de charge de travail qui utilise une routine circulaire permettant de sélectionner un gestionnaire de files d'attente disponible de destination d'un message. Pour plus d'informations, voir Clusters.

Informations d'adressage

Lorsqu'une application insère des messages destinés à un gestionnaire de files d'attente éloignées, le gestionnaire de files d'attente local ajoute dans les messages un en-tête de transmission avant de les placer dans la file d'attente de transmission. Cet en-tête contient le nom de la file d'attente de destination et du gestionnaire de files d'attente, à savoir les *informations d'adressage*.

Dans un environnement comportant un gestionnaire de files d'attente unique, l'adresse d'une file d'attente de destination est établie lorsqu'une application ouvre une file d'attente dans laquelle insérer des messages. Dans la mesure où la file d'attente de destination se trouve sur le même gestionnaire de files d'attente, aucune information d'adressage n'est nécessaire.

Dans un environnement de mise en file d'attente répartie, le gestionnaire de files d'attente doit connaître non seulement le nom de la file d'attente de destination, mais également l'emplacement de cette file d'attente (c'est-à-dire, le nom du gestionnaire de files d'attente) et le chemin d'accès à cet emplacement éloigné (c'est-à-dire, la file d'attente de transmission). Ces informations d'adressage sont contenues dans l'en-tête de transmission. Le canal récepteur supprime l'en-tête de transmission et utilise les informations qu'il contient pour localiser la file d'attente de destination.

Vous pouvez éviter que vos applications aient à indiquer le nom du gestionnaire de files d'attente de destination si vous utilisez une définition de file d'attente éloignée. Cette définition indique le nom de la file d'attente éloignée, le nom du gestionnaire de files d'attente éloignées auquel des messages sont destinés, ainsi que le nom de la file d'attente de transmission servant à transférer les messages.

Description des alias

Les alias servent à fournir une qualité de service pour les messages. L'alias de gestionnaire de files d'attente permet à un administrateur système de modifier le nom d'un gestionnaire de files d'attente cible sans que vous ayez à modifier vos applications. Il permet également à l'administrateur système de modifier le chemin d'accès à un gestionnaire de files d'attente de destination ou de configurer un chemin traversant un certain nombre d'autres gestionnaires de files d'attente (multi-tronçons). L'alias de file d'attente de réponses fournit une qualité de service pour les réponses.

Les alias de gestionnaire de files d'attente et les alias de file d'attente de réponses sont créés à l'aide d'une définition de file d'attente éloignée comportant une zone RNAME à blanc. Ces définitions ne définissent pas de files d'attente réelles ; elles sont utilisées par le gestionnaire de files d'attente pour résoudre des noms de file d'attente physique, des noms de gestionnaire de files d'attente et des files d'attente de transmission.

Les définitions d'alias se caractérisent par une zone RNAME à blanc.

Résolution de nom de file d'attente

La résolution de nom de file d'attente a lieu sur tous les gestionnaires de files d'attente chaque fois qu'une file d'attente est ouverte. Elle a pour but d'identifier la file d'attente cible, le gestionnaire de files d'attente cible (qui peut être local) et le chemin d'accès à ce gestionnaire de files d'attente (qui peut être NULL). Le nom résolu comporte trois parties : le nom du gestionnaire de files d'attente, le nom de file d'attente et, si le gestionnaire de files d'attente est éloigné, la file d'attente de transmission.

Lorsqu'il existe une définition de file d'attente éloignée, aucune définition d'alias n'est référencée. Le nom de file d'attente fourni par l'application est résolu en nom de la file d'attente de destination, du gestionnaire de files d'attente éloignées et de la file d'attente de transmission indiquée dans la définition de file d'attente éloignée. Pour plus d'informations sur la résolution de nom de file d'attente, voir Résolution de nom de file d'attente.

S'il n'existe aucune définition de file d'attente éloignée et qu'un nom de gestionnaire de files d'attente est spécifié ou résolu par le service annuaire, le gestionnaire de files d'attente vérifie si une définition d'alias de gestionnaire de files d'attente correspond au nom de gestionnaire de files d'attente fourni. Si c'est le cas, les informations qu'elle contient sont utilisées pour résoudre le nom de gestionnaire de files d'attente en nom du gestionnaire de files d'attente de destination. La définition d'alias de gestionnaire de files d'attente peut également être utilisée pour déterminer la file d'attente de transmission sur le gestionnaire de files d'attente de destination.

Si le nom de file d'attente résolu ne correspond pas à une file d'attente locale, le nom de gestionnaire de files d'attente et le nom de file d'attente sont tous deux inclus dans l'en-tête de transmission de chaque message inséré par l'application dans la file d'attente de transmission.

La file d'attente de transmission généralement utilisée porte le même nom que le gestionnaire de files d'attente résolu, sauf si elle est modifiée par une définition de file d'attente éloignée ou une définition d'alias de gestionnaire de files d'attente. Si vous n'avez pas défini une file d'attente de transmission de ce type et que vous avez défini une file d'attente de transmission par défaut, cette dernière est alors utilisée.

Les noms des gestionnaires de files d'attente exécutés sur z/OS sont limités à quatre caractères.

Définition des alias de gestionnaire de files d'attente

Les définitions d'alias de gestionnaire de files d'attente s'appliquent lorsqu'une application ouvrant une file d'attente pour insérer un message indique le nom de file d'attente **et** le nom de gestionnaire de files d'attente.

Les définitions d'alias de gestionnaire de files d'attente sont utilisées pour les trois opérations suivantes :

- Lors de l'envoi de messages, pour remapper le nom de gestionnaire de files d'attente
- Lors de l'envoi de messages, pour modifier ou spécifier la file d'attente de transmission
- Lors de la réception de messages, pour déterminer si le gestionnaire de files d'attente local est la destination prévue de ces messages

Messages sortants - remappage du nom de gestionnaire de files d'attente

Les définitions d'alias de gestionnaire de files d'attente permettent de remapper le nom de gestionnaire de files d'attente indiqué dans un appel MQOPEN. Par exemple, un appel MQOPEN spécifie le nom de file d'attente THISQ et le nom de gestionnaire de files d'attente YOURQM. Sur le gestionnaire de files d'attente local, il existe une définition d'alias de gestionnaire de files d'attente similaire à l'exemple suivant :

```
DEFINE QREMOTE (YOURQM) RQMNAME(REALQM)
```

Cet exemple montre que le gestionnaire de files d'attente réel à utiliser, lorsqu'une application insère des messages dans le gestionnaire de files d'attente YOURQM, est REALQM. Si le gestionnaire de files d'attente local est REALQM, il insère les messages dans la file d'attente THISQ, qui est une file d'attente locale. Si le gestionnaire de files d'attente local n'est pas intitulé REALQM, il achemine le message vers

une file d'attente de transmission intitulée REALQM. Le gestionnaire de files d'attente modifie l'en-tête de transmission de sorte qu'il indique REALQM à la place de YOURQM.

Messages sortants - modification ou spécification de la file d'attente de transmission

La Figure 15, à la page 58 présente un scénario où les messages arrivent sur le gestionnaire de files d'attente QM1 à l'aide d'en-têtes de transmission affichant des noms de file d'attente sur le gestionnaire de files d'attente QM3. Dans ce scénario, QM3 est accessible à l'aide de plusieurs tronçons via QM2.

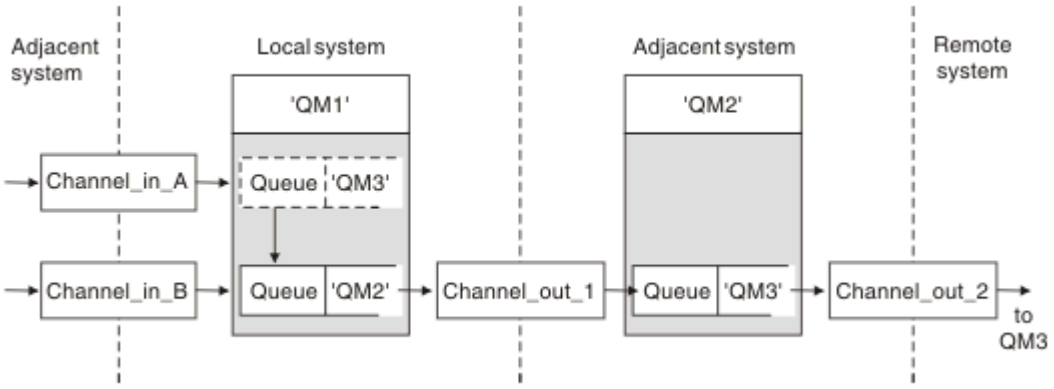


Figure 15. Alias de gestionnaire de files d'attente

Tous les messages de QM3 sont capturés sur QM1 à l'aide d'un alias de gestionnaire de files d'attente. Ce dernier est intitulé QM3 et contient la définition QM3 via la file d'attente de transmission QM2. La définition est similaire à l'exemple suivant :

```
DEFINE QREMOTE (QM3) RNAME(' ') RQMNAME(QM3) XMITQ(QM2)
```

Le gestionnaire de files d'attente insère les messages dans la file d'attente de transmission QM2 mais ne modifie pas l'en-tête de la file d'attente de transmission car le nom du gestionnaire de files d'attente de destination, QM3, ne change pas.

Tous les messages arrivant sur QM1 et présentant un en-tête de transmission contenant un nom de file d'attente sur QM2 sont également insérés dans la file d'attente de transmission QM2. Les messages comportant des destinations différentes sont ainsi collectés sur une file d'attente de transmission commune d'un système adjacent approprié pour être retransmis ensuite à leurs destinations.

Messages entrants - détermination de la destination

Un agent MCA récepteur ouvre la file d'attente référencée dans l'en-tête de transmission. S'il existe une définition d'alias de gestionnaire de files d'attente du même nom que le gestionnaire de files d'attente référencé, le nom de gestionnaire de files d'attente reçu dans l'en-tête de transmission est remplacé par le nom RQMNAME issu de cette définition.

Ce processus est utilisé pour les deux opérations suivantes :

- Acheminement des messages vers un autre gestionnaire de files d'attente
- Modification du nom de gestionnaire de files d'attente pour qu'il soit identique à celui du gestionnaire de files d'attente local

Définition des alias de files d'attente de réponse

Une définition d'alias de file d'attente de réponses indique d'autres noms pour les informations de réponse dans le descripteur de message. Cela a pour avantage que vous pouvez modifier le nom d'une file d'attente ou d'un gestionnaire de files d'attente sans avoir à modifier vos applications.

Résolution de nom de file d'attente

Lorsqu'une application répond à un message, elle utilise les données du *descripteur* du message qu'elle a reçu pour trouver le nom de la file d'attente à laquelle répondre. L'application émettrice indique l'emplacement dans lequel les réponses sont envoyées et joint ces informations à ses messages. Ce concept doit être coordonné dans le cadre de votre conception d'application.

La résolution de nom de file d'attente a lieu à l'extrémité émettrice de votre application, avant que le message soit inséré dans une file d'attente. La résolution de nom de file d'attente se produit par conséquent avant une interaction avec l'application distante vers laquelle le message est envoyé. C'est la seule situation où la résolution de nom a lieu lorsqu'une file d'attente n'est pas ouverte.

Résolution de nom de file d'attente à l'aide d'un alias de gestionnaire de files d'attente

En général, une application spécifie une file d'attente de réponses et laisse le nom du gestionnaire de files d'attente de réponses à blanc. Le gestionnaire de files d'attente complète son propre nom au moment de l'insertion. Cette méthode fonctionne correctement sauf lorsque vous souhaitez qu'un autre canal soit utilisé pour les réponses, par exemple un canal utilisant la file d'attente de transmission `QM1_relief` à la place du canal de retour par défaut qui utilise la file d'attente de transmission `QM1`. Dans cette situation, les noms de gestionnaire de files d'attente indiqués dans les en-têtes de file d'attente de transmission ne correspondent pas à des noms de gestionnaire de files d'attente "réels", mais sont spécifiés à l'aide des définitions d'alias de gestionnaire de files d'attente. Pour renvoyer des réponses via des chemins secondaires, il est également nécessaire de mapper les données de file d'attente de réponses à l'aide des définitions d'alias de file d'attente de réponses.

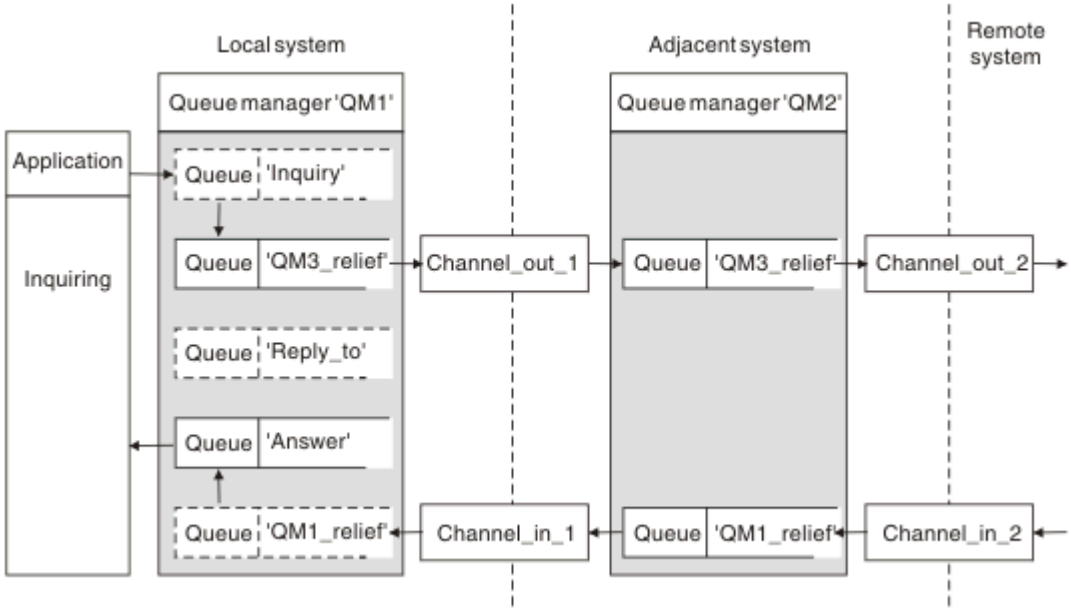


Figure 16. Alias de file d'attente de réponses utilisé pour la modification de l'emplacement des réponses

Dans l'exemple de la figure Figure 16, à la page 59 :

1. L'application insère un message à l'aide de l'appel MQPUT et en spécifiant les informations suivantes dans le descripteur de message :

   ```
   ReplyToQ='Reply_to'
   ReplyToQMgr=' '
   ```

   ReplyToQMgr doit être à blanc pour que l'alias de file d'attente de réponses soit utilisé.

2. Vous créez une définition d'alias de file d'attente de réponses intitulée `Reply_to`, qui contient le nom `Answer`, ainsi que le nom de gestionnaire de files d'attente `QM1_relief`.

```
DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
RQMNAME ('QM1_relief')
```

3. Les messages sont envoyés à l'aide d'un descripteur de message affichant `ReplyToQ='Answer'` et `ReplyToQMgr='QM1_relief'`.
4. La spécification d'application doit inclure l'information selon laquelle les réponses doivent être trouvées dans la file d'attente `Answer` et non dans `Reply_to`.

Pour préparer les réponses, vous devez créer le canal de retour parallèle, en définissant :

- Sur QM2, la file d'attente de transmission intitulée `QM1_relief`

```
DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
```

- Sur QM1, l'alias de gestionnaire de files d'attente `QM1_relief`

```
DEFINE QREMOTE ('QM1_relief') RNAME() RQMNAME(QM1)
```

  Cet alias de gestionnaire de files d'attente met fin à la chaîne des canaux de retour parallèles et capture les messages pour QM1.

Si vous pensez que vous serez amené à effectuer cette opération ultérieurement, assurez-vous que les applications utilisent le nom d'alias depuis le début. Pour l'instant, il s'agit d'un alias de file d'attente normal pour la file d'attente de réponses mais, par la suite, il peut être remplacé par un alias de gestionnaire de files d'attente.

Nom de file d'attente de réponses

Des précautions particulières sont nécessaires lors de l'attribution de nom aux files d'attente de réponses. La raison pour laquelle une application insère un nom de file d'attente de réponses dans le message est qu'elle peut spécifier la file d'attente à laquelle ses réponses sont envoyées. Lors de la création d'une définition d'alias de file d'attente de réponses à l'aide de ce nom, la file d'attente de réponses réelle (c-à-d, une définition de file d'attente locale) ne peut pas porter le même nom. Par conséquent, la définition d'alias de file d'attente de réponses doit contenir un nouveau nom de file d'attente ainsi que le nom de gestionnaire de files d'attente, et la spécification d'application doit inclure l'information selon laquelle ses réponses se trouvent dans cette autre file d'attente.

Les applications doivent désormais extraire les messages d'une file d'attente différente de celle qu'elles ont désignée comme file d'attente de réponses, lorsqu'elles insèrent le message d'origine.

Composants de cluster

Les clusters sont composés de gestionnaires de files d'attente, de référentiels de cluster, de canaux de cluster et de files d'attente de cluster.

Voir les sous-rubriques suivantes pour plus d'informations sur chaque composant du cluster :

Concepts associés

Comparaison de la mise en cluster et de la mise en file d'attente répartie

Tâches associées

Configuration d'un cluster de gestionnaires de files d'attente

Configuration d'un nouveau cluster

Référentiel de cluster

Un référentiel est un ensemble d'informations concernant les gestionnaires de files d'attente membres d'un cluster.

Les informations du référentiel comprennent les noms des gestionnaires de files d'attente, leurs emplacements, leurs canaux, les files d'attente qu'ils hébergent et d'autres détails. Les informations sont stockées sous forme de messages dans une file d'attente appelée SYSTEM.CLUSTER.REPOSITORY.QUEUE. Cette file d'attente est l'un des objets par défaut. Multi Sous Multiplateformes, il est défini lors de la création d'un gestionnaire de files d'attente IBM MQ. z/OS Dans IBM MQ for z/OS, elle est définie dans le cadre de la personnalisation du gestionnaire de files d'attente.

Référentiel complet et référentiel partiel

En règle générale, deux gestionnaires de files d'attente dans un cluster contiennent un référentiel complet. Les autres gestionnaires de files d'attente contiennent tous un référentiel partiel.

Un gestionnaire de files d'attente qui héberge un ensemble complet d'informations concernant tous les gestionnaires de files d'attente d'un cluster dispose d'un référentiel complet. Les autres gestionnaires de files d'attente du cluster comportent des référentiels partiels contenant un sous-ensemble des informations des référentiels complets.

Un référentiel partiel contient uniquement des informations relatives aux gestionnaires avec lesquels le gestionnaire de files d'attente a besoin d'échanger des messages. Les gestionnaires de files d'attente demandent les mises à jour apportées aux informations dont ils ont besoin pour que, si ces informations sont modifiées, le gestionnaire de files d'attente de référentiel complet leur envoie les nouvelles informations. La plupart du temps, le référentiel partiel contient toutes les informations nécessaires pour qu'un gestionnaire de files d'attente soit opérationnel dans le cluster. Lorsqu'un gestionnaire de files d'attente a besoin d'informations complémentaires, il interroge le référentiel complet et met à jour son référentiel partiel. Les gestionnaires de files d'attente utilisent la file d'attente SYSTEM.CLUSTER.COMMAND.QUEUE pour demander des mises à jour aux référentiels et les recevoir.

Lorsque vous migrez des gestionnaires de files d'attente qui sont membres d'un cluster, migrez les référentiels complets avant les référentiels partiels. En effet, un référentiel plus ancien ne peut pas stocker des attributs récents introduits dans une nouvelle édition. Il les tolère mais ne les stocke pas.

Gestionnaire de files d'attente de cluster

Un gestionnaire de files d'attente de cluster est un gestionnaire de files d'attente membre d'un cluster.

Un gestionnaire de files d'attente peut être membre de plusieurs clusters. Chaque gestionnaire de files d'attente de cluster doit porter un nom unique dans tous les clusters dont il est membre.

Un gestionnaire de files d'attente de cluster peut héberger des files d'attente qu'il annonce aux autres gestionnaires de files d'attente du cluster. Ce n'est cependant pas obligatoire. Il peut plutôt envoyer des messages dans les files d'attente hébergées ailleurs dans le cluster et recevoir uniquement les réponses qui sont explicitement dirigées vers ce gestionnaire.

z/OS Dans IBM MQ for z/OS, un gestionnaire de files d'attente de cluster peut être membre d'un groupe de partage de files d'attente. Dans ce cas, il partage ses définitions de file d'attente avec les autres gestionnaires de files d'attente du même groupe de partage.

Les gestionnaires de files d'attente de cluster sont autonomes. Ils disposent d'un contrôle complet sur les files d'attente et les canaux qu'ils définissent. Leurs définitions ne peuvent pas être modifiées par d'autres gestionnaires de files d'attente (autres que les gestionnaires du même groupe de partage de files d'attente). Les gestionnaires de files d'attente de référentiel ne contrôlent pas les définitions des autres gestionnaires de files d'attente du cluster. Ils contiennent un ensemble complet de toutes les définitions pour les utiliser si nécessaire. Un cluster est une fédération de gestionnaires de files d'attente.

Après vous avez créé ou modifié une définition sur un gestionnaire de files d'attente de cluster, les informations sont envoyées au gestionnaire de files d'attente de référentiel complet. Les autres référentiels du cluster sont mis à jour ultérieurement.

Gestionnaire de files d'attente de référentiel complet

Un gestionnaire de files d'attente de référentiel complet est un gestionnaire de files d'attente de cluster qui contient une représentation complète des ressources du cluster. Pour garantir la disponibilité, configurez plusieurs gestionnaires de files d'attente de référentiel complet dans chaque cluster. Les gestionnaires de files d'attente de référentiel complet reçoivent des informations envoyées par les autres gestionnaires de files d'attente du cluster et mettent à jour leurs référentiels. Ils s'envoient entre eux des messages pour s'assurer de disposer des nouvelles informations à jour concernant le cluster.

Gestionnaires de files d'attente et référentiels

Chaque cluster comprend au moins un (de préférence, deux) gestionnaire de files d'attente contenant des référentiels complet des informations sur les gestionnaires de files d'attente, les files d'attente et les canaux d'un cluster. Ces référentiels contiennent également les demandes mises à jour des informations émanant des autres gestionnaires de files d'attente du cluster.

Les autres gestionnaires de files d'attente comportent un référentiel partiel contenant des informations sur le sous-ensemble de files d'attente et de gestionnaire de files d'attente avec lesquels ils doivent communiquer. Les gestionnaires de files d'attente génèrent leurs référentiels partiels en effectuant des interrogations lorsqu'ils ont besoin pour la première fois d'accéder à une autre file d'attente ou un autre gestionnaire de files d'attente. Ils demandent à être avertis des nouvelles informations concernant cette file d'attente ou ce gestionnaire de files d'attente.

Chaque gestionnaire de files d'attente stocke ses informations de référentiel dans des messages dans une file d'attente appelée `SYSTEM.CLUSTER.REPOSITORY.QUEUE`. Les gestionnaires de files d'attente échangent des informations de référentiel dans des messages dans une file d'attente appelée `SYSTEM.CLUSTER.COMMAND.QUEUE`.

Chaque gestionnaire de files d'attente qui rejoint un cluster définit un canal émetteur de cluster, `CLUSSDR` vers l'un des référentiels. Il prend immédiatement connaissance des autres gestionnaires de files d'attente du cluster qui contiennent des référentiels complets. Il peut alors demander des informations à n'importe lequel des référentiels. Lorsque le gestionnaire de files d'attente envoie des informations au référentiel choisi, il envoie également les informations à une autre référentiel (s'il en existe un).

Un référentiel complet est mis à jour lorsque le gestionnaire de files d'attente qui l'héberge reçoit de nouvelles informations d'un des gestionnaires de files d'attente qui lui sont associés. Les nouvelles informations sont également envoyées à un autre référentiel pour réduire le risque de délai si un gestionnaire de files d'attente de référentiel est hors service. Comme toutes les informations sont envoyées deux fois, les référentiels doivent supprimer les doublons. Chaque élément d'information est associé à un numéro de séquence que les référentiels utilisent pour identifier les doublons. Tous les référentiels se synchronisent entre eux en échangeant des messages.

Files d'attente de cluster

Une file d'attente de cluster est une file d'attente hébergée par un gestionnaire de files d'attente de cluster et accessible aux autres gestionnaires de files d'attente dans le cluster.

Une file d'attente de cluster est annoncée aux autres gestionnaires de files d'attente de cluster. Ces autres gestionnaires de files d'attente de cluster peuvent insérer des messages dans une file d'attente de cluster sans qu'une définition de file d'attente éloignée correspondante soit nécessaire. Une file d'attente de cluster peut être annoncée dans plusieurs clusters à l'aide d'une liste de noms de cluster.

Lorsqu'une file d'attente est annoncée, les gestionnaires de files d'attente de cluster peuvent y insérer des messages. Pour insérer un message, le gestionnaire de files d'attente doit déterminer, à partir des référentiels complets, l'emplacement où la file d'attente est hébergée. Il ajoute alors des informations de routage au message et insère ce dernier dans une file d'attente de transmission du cluster.

Une file d'attente de cluster peut être une file d'attente partagée par les membres d'un groupe de partage de files d'attente dans IBM MQ for z/OS.

Tâches associées

Définition de files d'attente de cluster

Comparison between shared queues and cluster queues

This information is designed to help you compare shared queues and cluster queues, and decide which might be more suitable for your system.

Channel Initiator costs

In cluster queues, messages are sent by channels, so allow for channel initiator costs in addition to application costs. There are costs in the network because channels get and put messages. These costs are not present with shared queues, which therefore use less processing power than cluster queues when moving messages between queue managers in a queue sharing group.

Availability of messages

When putting to a queue, cluster queues send the message to one of the queue managers with active channels connected to your queue manager. On the remote queue manager, if applications used to process the messages are not working, the messages are not processed and wait until the applications start. Similarly, if a queue manager is shut down, any messages on the queue manager are not made available until the queue manager restarts. These instances show lower message availability than when using shared queues.

When using shared queues, any application in the queue sharing group can get messages that are sent. If you shut down one queue manager in the queue sharing group, messages are available to the other queue managers, providing higher message availability than when using cluster queues.

Capacity

A coupling facility is more expensive than a disk; therefore the cost of storing 1,000,000 messages in a local queue is lower than having a coupling facility with enough capacity to store the same number of messages.

Sending to other queue managers

Shared-queue messages are only available within a queue sharing group. If you want to use a queue manager outside of the queue sharing group, you must use channels. You can use clustering to workload balance between multiple remote distributed queue managers.

Workload balancing

You can use clustering to give weight to which channels and queue managers get a proportion of the messages sent. For example, you can send 60% of messages to one queue manager, and 40% of messages to another queue manager. This instance does not depend on the ability of the remote queue manager to process work. The system with the first queue manager might be overloaded, and the system with the second queue manager might be idle, but most of the messages still go to the first queue manager.

With shared queues, two CICS® systems can get messages. If one system is overloaded, the other system takes over most the workload.

Canaux de cluster

Dans chaque référentiel complet, vous définissez manuellement un canal récepteur de cluster et un ensemble de canaux émetteurs de cluster pour vous connecter à tout autre référentiel complet dans le cluster. Lorsque vous ajoutez un référentiel partiel, vous définissez manuellement un canal récepteur de

cluster et un canal émetteur de cluster unique qui se connecte à l'un des référentiels complets. D'autres canaux émetteurs de cluster sont définis automatiquement par le cluster, le cas échéant. Les canaux émetteurs de cluster définis automatiquement tirent leurs attributs de la définition de canal récepteur de cluster correspondante dans le gestionnaire de files d'attente de réception.

Canal récepteur de cluster : CLUSRCVR

Une définition de canal CLUSRCVR définit l'extrémité d'un canal, sur laquelle un gestionnaire de files d'attente de cluster peut recevoir des messages d'autres gestionnaires de files d'attente du cluster.

Vous devez définir au moins un canal CLUSRCVR pour chaque gestionnaire de files d'attente de cluster. En définissant le canal CLUSRCVR, le gestionnaire de files d'attente indique aux autres gestionnaires qu'il est disponible pour recevoir des messages.

Une définition de canal CLUSRCVR permet également aux autres gestionnaires de files d'attente de définir automatiquement les définitions de canal émetteur de cluster correspondantes. Voir la section «Canaux émetteurs de cluster définis automatiquement», à la page 64 de cet article.

Canal émetteur de cluster : CLUSSDR

Vous définissez manuellement un canal CLUSSDR entre chaque gestionnaire de files d'attente de référentiel complet et tout autre gestionnaire de files d'attente de référentiel complet dans le cluster. Toutes les mises à jour échangées par les référentiels complets sont transmises exclusivement sur ces canaux. En définissant manuellement ces canaux, vous contrôlez explicitement le réseau de référentiels complets.

Lorsque vous ajoutez un gestionnaire de files d'attente de référentiel partiel à un cluster, vous définissez manuellement un canal CLUSSDR unique qui se connecte à l'un des référentiels complets. Peu importe le référentiel complet choisi car une fois le premier contact établi, d'autres objets de gestionnaire de files d'attente de cluster, canaux CLUSSDR compris, sont définis automatiquement le cas échéant. Cela permet au gestionnaire de files d'attente d'envoyer des informations de cluster à n'importe quel référentiel complet et d'envoyer des messages à n'importe quel gestionnaire de files d'attente dans le cluster.

Comme indiqué dans la section de cet article, les canaux émetteurs définis automatiquement sont basés sur la configuration du canal récepteur de cluster. Ainsi, les propriétés de canal que vous définissez sur les canaux de cluster doivent être identiques sur les canaux CLUSSDR et les canaux récepteurs de cluster correspondants, ou définissez uniquement des canaux récepteurs de cluster.

Les canaux CLUSSDR doivent être définis manuellement uniquement pour les raisons indiquées précédemment. C'est-à-dire dans le but de connecter initialement un référentiel partiel à un référentiel complet ou de connecter entre eux deux référentiels complets. La configuration manuelle d'un canal CLUSSDR qui se connecte à un référentiel partiel ou à un gestionnaire de files d'attente qui n'est pas dans le cluster génère des messages d'erreur comme AMQ9427 et AMQ9428. Même s'il peut s'agir parfois d'une situation temporaire inévitable, par exemple, lors de la modification de l'emplacement d'un référentiel complet, la définition manuelle doit être supprimée dès que possible.

Canaux émetteurs de cluster définis automatiquement

Généralement, lorsque vous ajoutez un gestionnaire de files d'attente de référentiel complet à un cluster, vous définissez manuellement deux canaux de cluster seulement sur le gestionnaire de files d'attente :

- Un canal émetteur de cluster (CLUSSDR) à un gestionnaire de files d'attente de référentiel complet pour le cluster.
- Un canal récepteur de cluster (CLUSRCVR).

Le canal CLUSSDR que vous définissez permet au gestionnaire de files d'attente d'établir le premier contact avec le cluster. Une fois le premier contact établi, d'autres canaux CLUSSDR sont définis automatiquement par le cluster, le cas échéant.

Un canal CLUSSDR défini automatiquement tire ses attributs de la définition de canal CLUSRCVR correspondante sur le gestionnaire de files d'attente de réception. Même s'il existe un canal CLUSSDR

défini manuellement, les attributs du canal CLUSSDR défini automatiquement sont utilisés. Supposons par exemple que vous définissez un canal CLUSRCVR sans indiquer de numéro de port dans le paramètre **CONNAME** et que vous définissez manuellement un canal CLUSSDR qui n'indique pas de numéro de port. Lorsque le canal CLUSSDR défini automatiquement remplace celui qui est défini manuellement, le numéro de port (tiré du canal CLUSRCVR) reste à blanc. Le numéro de port par défaut est utilisé et le canal échoue.

Lorsqu'il existe des différences de configuration entre un canal CLUSSDR défini manuellement et une définition de canal CLUSRCVR correspondante, certaines prennent effet immédiatement (par exemple les paramètres d'équilibrage de charge) et d'autres prennent effet uniquement lors du redémarrage du canal (par exemple, la configuration TLS).

Pour éviter toute confusion, respectez si possible les consignes suivantes :

- Définissez manuellement des canaux CLUSSDR uniquement dans le but de pointer vers des référentiels complets.
- Après avoir défini manuellement des canaux CLUSSDR, configurez-les de sorte qu'ils soient identiques à la définition de canal CLUSRCVR correspondante sur le gestionnaire de files d'attente de réception.

Voir aussi Utilisation des canaux définis automatiquement.

Concepts associés

Utilisation de canaux définis automatiquement

Utilisation de files d'attente de transmission de cluster et de canaux émetteurs de cluster

Tâches associées

Configuration d'un nouveau cluster

Ajout d'un gestionnaire de files d'attente à un cluster

Rubriques de cluster

Les rubriques de cluster sont des rubriques d'administration avec l'attribut **`cluster`** défini. Des informations les concernant sont transmises à tous les membres d'un cluster et associées aux rubriques locales pour créer des portions d'un espace de sujet qui s'étend sur plusieurs gestionnaires de files d'attente. Cela permet aux messages publiés dans une rubrique sur un gestionnaire de files d'attente d'être distribués aux abonnements d'autres gestionnaires de files d'attente du cluster.

Lorsque vous définissez une rubrique de cluster dans un gestionnaire de files d'attente, la définition de la rubrique de cluster est envoyée aux gestionnaires de files d'attente des référentiels complets. Ces derniers propagent ensuite la définition de rubrique en cluster à tous les gestionnaires de files d'attente du cluster, ce qui rend la même rubrique de cluster disponible pour les diffuseurs de publications et les abonnés dans un gestionnaire de files d'attente du cluster. Le gestionnaire de files d'attente sur lequel vous créez une rubrique de cluster est appelé hôte de rubrique de cluster. La rubrique de cluster peut être utilisée par n'importe quel gestionnaire de files d'attente du cluster, mais les modifications à lui apporter doivent être effectuées sur celui où elle est définie (l'hôte). Ces modifications sont ensuite propagées à tous les membres du cluster par le biais des référentiels complets.

Pour toute information sur la configuration des rubriques de cluster de sorte qu'elles utilisent le *routage direct* ou le *routage via un hôte de rubrique* et sur l'héritage des rubriques de cluster et les abonnements génériques, voirDéfinition de rubriques de cluster.

Pour des informations sur les commandes à utiliser pour afficher les rubriques de cluster, voir les informations connexes.

Concepts associés

Utilisation des rubriques d'administration

Utilisation des abonnements

Référence associée

DISPLAYTOPIC

STATUT D'AFFICHAGE

DISPLAYSUB

Objets de cluster par défaut

Multi Sur Multiplatforms, les objets de cluster par défaut sont inclus dans l'ensemble d'objets par défaut créé automatiquement lorsque vous définissez un gestionnaire de files d'attente. z/OS Sous z/OS, vous trouverez les définitions d'objet de cluster par défaut dans les exemples de personnalisation.

Remarque : Vous pouvez modifier les définitions de canal par défaut de le même manière que toute autre définition de canal, en exécutant des commandes MQSC ou PCF. Ne modifiez pas les définitions de files d'attente par défaut, sauf SYSTEM.CLUSTER.HISTORY.QUEUE.

SYSTEM.CLUSTER.COMMAND.QUEUE
Chaque gestionnaire de files d'attente d'un cluster dispose d'une file d'attente locale appelée SYSTEM.CLUSTER.COMMAND.QUEUE qui est utilisée pour transférer des messages au référentiel complet. Les messages contiennent les informations nouvelles ou modifiées concernant le gestionnaire de files d'attente, ou les demandes d'informations à propos d'autres gestionnaires de files d'attente. SYSTEM.CLUSTER.COMMAND.QUEUE est vide normalement.

SYSTEM.CLUSTER.HISTORY.QUEUE
Chaque gestionnaire de files d'attente d'un cluster dispose d'une file d'attente locale appelée SYSTEM.CLUSTER.HISTORY.QUEUE. SYSTEM.CLUSTER.HISTORY.QUEUE est utilisée pour le stockage de l'historique des informations d'état de cluster pour les opérations de maintenance.

Dans les paramètres d'objet par défaut, SYSTEM.CLUSTER.HISTORY.QUEUE est défini sur PUT (ENABLED). Pour supprimer la collecte d'historique, remplacez le paramètre par PUT (DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
Chaque gestionnaire de files d'attente d'un cluster dispose d'une file d'attente locale appelée SYSTEM.CLUSTER.REPOSITORY.QUEUE. Cette file d'attente est utilisée pour stocker toutes les informations de référentiel complet. Cette file d'attente n'est pas vide normalement.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Chaque gestionnaire de files d'attente dispose d'une file d'attente locale appelée SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE est la file d'attente de transmission par défaut pour tous les messages vers toutes les files d'attente et tous les gestionnaires de files d'attente se trouvant dans des clusters. Vous pouvez remplacer la file d'attente de transmission par défaut de chaque canal émetteur de cluster par SYSTEM.CLUSTER.TRANSMIT.*ChannelName*en modifiant l'attribut de gestionnaire de files d'attente DEFCLXQ . Vous ne pouvez pas supprimer SYSTEM.CLUSTER.TRANSMIT.QUEUE. Il est également utilisé pour définir les vérifications d'autorisation si la file d'attente de transmission par défaut utilisée est SYSTEM.CLUSTER.TRANSMIT.QUEUE ou SYSTEM.CLUSTER.TRANSMIT. *ChannelName*.

SYSTEM.DEF.CLUSRCVR
Chaque cluster dispose d'une définition de canal CLUSRCVR par défaut appelée SYSTEM.DEF.CLUSRCVR. SYSTEM.DEF.CLUSRCVR permet de fournir les valeurs par défaut pour les attributs que vous ne spécifiez pas lorsque vous créez un canal récepteur de cluster sur un gestionnaire de files d'attente dans le cluster.

SYSTEM.DEF.CLUSSDR
Chaque cluster dispose d'une définition de canal CLUSSDR par défaut appelée SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDR permet de fournir les valeurs par défaut pour les attributs que vous ne spécifiez pas lorsque vous créez un canal émetteur de cluster sur un gestionnaire de files d'attente dans le cluster.

Concepts associés
Utilisation des objets de cluster par défaut

Messagerie de type publication/abonnement

La messagerie de type publication/abonnement vous permet de dissocier le fournisseur d'informations des consommateurs de ces informations. Il n'est pas nécessaire que l'application émettrice et l'application de réception se connaissent pour que les informations puissent être envoyées et reçues.

Pour qu'une application IBM MQ point-à-point puisse envoyer un message à une autre application, elle doit connaître cette application. Par exemple, elle doit connaître le nom de la file d'attente vers laquelle envoyer les informations et peut également indiquer un nom de gestionnaire de files d'attente.

Avec IBM MQ, votre application n'a pas besoin de connaître l'application cible. Toutes les application émettrices doivent exécuter les opérations suivantes :

- *Insérer* un message IBM MQ qui contient les informations nécessaire à l'application.
- Affecter le message à une rubrique indiquant le sujet des informations.
- Laisser IBM MQ traiter la distribution de ces informations.

De façon similaire, l'application cible peut ignorer la source des informations qu'elle reçoit.

La figure suivante illustre le système de publication/abonnement le plus simple. Il est composé d'un diffuseur de publications, d'un gestionnaire de files d'attente et d'un abonné. Un abonnement est effectué par l'abonné auprès du gestionnaire de files d'attente, une publication est envoyée depuis le diffuseur de publications vers le gestionnaire de files d'attente, puis est transmise par le gestionnaire de files d'attente à l'abonné.

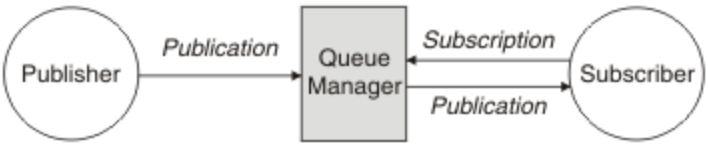


Figure 17. Configuration de publication/abonnement simple

Un système de publication/abonnement standard est composé de plusieurs diffuseurs depublications et d'abonnés à de nombreuses rubriques différentes, et souvent, de plusieurs gestionnaires de files d'attente. Une application peut être à la fois un éditeur et un abonné.

Une autre différence importante entre la messagerie de publication/abonnement et le mode point-à-point est qu'un message envoyé à une file d'attente point-à-point est traité uniquement par une seule application concernée. Un message publié dans une rubrique de publication/abonnement, pour laquelle plusieurs abonnés ont manifesté un intérêt, est traité par chaque abonné intéressé.

Composants de publication/abonnement

La fonction de publication/abonnement est le mécanisme par lequel les abonnés peuvent recevoir des informations de diffuseurs de publications sous la forme de messages. Les interactions entre les diffuseurs et les abonnés sont contrôlées par des gestionnaires de files d'attente à l'aide de fonctions IBM MQ standard.

Un système de publication/abonnement standard est composé de plusieurs diffuseurs depublications et d'abonnés à de nombreuses rubriques différentes, et souvent, de plusieurs gestionnaires de files d'attente. Une application peut être à la fois un éditeur et un abonné.

Le fournisseur des informations est appelé le *diffuseur de publications*. Les diffuseurs de publications fournissent des informations sur un sujet, sans avoir besoin de connaître les applications qu'elles intéressent. Ils génèrent ces informations sous la forme de messages, appelés *publications*, qu'ils souhaitent publier et définissent la rubrique de ces messages.

Le consommateur de ces informations est appelé un *abonné*. Les abonnés créent des *abonnements* qui décrivent la rubrique qui intéresse l'abonné. L'abonnement détermine donc les publications transmises à l'abonné. Les abonnés peuvent souscrire différents abonnements et recevoir des informations de différents diffuseurs de publications.

Les informations publiées sont envoyées dans un message IBM MQ, et leur sujet est identifié par sa *rubrique*. Lorsqu'il publie des informations, le diffuseur de publications les associe à une rubrique et l'abonné indique les rubriques à propos desquelles il souhaite recevoir des publications. L'abonné reçoit des informations concernant uniquement les rubriques auxquelles il s'est abonné.

C'est l'existence de rubriques qui permet aux fournisseurs et consommateurs d'informations d'être dissociés de la messagerie de type publication/abonnement, ce qui élimine le besoin d'inclure une destination spécifique dans chaque message, comme dans la messagerie point à point.

Les interactions entre les diffuseurs et les abonnés sont contrôlées par un gestionnaire de files d'attente. Le gestionnaire de files d'attente reçoit les messages envoyés par les diffuseurs de publication et les abonnements des abonnés (à un certain nombre de rubriques). Le travail du gestionnaire de files d'attente consiste à acheminer les messages publiés vers les abonnés susceptibles d'être intéressés par la rubrique des messages.

Des fonctions standard IBM MQ sont utilisées pour distribuer les messages afin que les applications puissent tirer parti de toutes les fonctionnalités disponibles pour les applications IBM MQ existantes. Cela signifie que vous pouvez utiliser des messages persistants pour bénéficier d'une distribution assurée une seule fois et que vos messages peuvent faire partie d'une unité d'oeuvre transactionnelle afin de garantir qu'ils sont distribués à l'abonné uniquement s'ils sont validés par le diffuseur de publications.

Diffuseurs de publications et publications

Dans la publication/l'abonnement IBM MQ, un diffuseur est une applications qui fournit des informations sur une rubrique à un gestionnaire de files d'attente sous la forme d'un message standard IBM MQ appelé publication. Un diffuseur de publications peut publier des informations sur plusieurs rubriques.

Les diffuseurs de publications utilisent l'instruction MQPUT pour placer un message (publication) dans une rubrique précédemment ouverte. Le gestionnaire de files d'attente local achemine ensuite la publication vers les abonnés qui possèdent un abonnement à la rubrique de la publication. Un message publié peut être utilisé par plusieurs abonnés.

Outre la distribution des publications à l'ensemble des abonnés locaux qui possèdent des abonnements appropriés, un gestionnaire de files d'attente peut également distribuer la publication à d'autres gestionnaires de files d'attente connectés, soit directement ou via un réseau de gestionnaires de files d'attente possédant des abonnements à la rubrique.

Dans un réseau de publication/abonnement IBM MQ, une application de publication peut être également un abonné.

Publications sous le contrôle d'un point de synchronisation

Les diffuseurs de publications peuvent émettre des appels MQPUT ou MQPUT1 sous un point de synchronisation afin d'inclure tous les messages distribués aux abonnés dans une unité d'oeuvre. Si l'option MQPMO_RETAIN, ou les options de distribution de rubrique NPMSGDLV ou PMSGDLV avec les valeurs ALL ou ALLDUR, sont spécifiées, le gestionnaire de files d'attente utilise des appels MQPUT ou MQPUT1 internes dans un point de synchronisation, dans la portée du diffuseur de publications MQPUT ou de l'appel MQPUT1.

Informations d'état et données sur l'événement

Les publications peuvent être catégorisées en publications d'état, comme le cours actuel d'une action, ou en publications d'événement, comme la vente de cette action.

Publications relatives à l'état

Les *publications d'état* contiennent des informations sur l'état actuel d'un élément, comme le cours d'une action ou le score actuel d'un match de football. Lorsqu'un événement se produit, (par exemple, le prix des articles est modifié ou le score d'un match change), les informations relatives à l'état précédent sont obsolètes et sont remplacées par les informations les plus récentes.

Un abonné souhaite recevoir la version actuelle des informations d'état au démarrage, puis recevoir les nouvelles informations chaque fois que l'état change.

Si une publication contient des informations d'état, elle est souvent publiée en tant que publication conservée. Un nouvel abonné souhaite généralement obtenir immédiatement les informations d'état

en cours sans attendre qu'un événement entraîne la une nouvelle publication des informations. Les abonnés recevront automatiquement la publication conservée d'une rubrique lorsqu'ils s'abonnent sauf s'ils utilisent les options MQSO_PUBLICATIONS_ON_REQUEST ou MQSO_NEW_PUBLICATIONS_ONLY.

Publications relatives aux événements

Les *publications d'événement* contiennent des informations sur des événements individuels qui se produisent, comme la vente d'une action ou le marquage d'un but. Chaque événement est indépendant des autres événements.

Un abonné souhaite recevoir des informations sur les événements au fur et à mesure qu'ils se produisent.

Publications conservées

Par défaut, après qu'une publication a été envoyée à tous les abonnés intéressés, elle est supprimée. Toutefois, un diffuseur de publications peut indiquer qu'une copie d'une publication est conservée pour être envoyée aux abonnés ultérieurs qui manifestent un intérêt pour cette rubrique.

La suppression de publications après leur envoi à tous les abonnés intéressés est adaptée aux informations d'événement, mais pas toujours aux informations d'état. Lorsqu'un message est conservé, les nouveaux abonnés ne doivent pas attendre que les informations d'état initial soient à nouveau publiées pour les recevoir. Par exemple, un abonné souscrivant un abonnement au cours d'une action recevra immédiatement le cours actuel de l'action, sans attendre que celui-ci change (et soit donc publié à nouveau).

Le gestionnaire de files d'attente ne peut conserver qu'une publication par rubrique pour que la publication conservée existante d'une rubrique soit supprimée dès qu'il reçoit une nouvelle publication conservée. Toutefois, la suppression de la publication existante peut ne pas avoir lieu de façon synchrone avec l'arrivée de la nouvelle publication conservée. Par conséquent, dans la mesure du possible, faites en sorte qu'un seul diffuseur de publications envoie les publications conservées d'une rubrique.

Les abonnés peuvent préciser qu'ils ne souhaitent pas recevoir de publications conservées en utilisant l'option d'abonnement MQSO_NEW_PUBLICATIONS_ONLY. Les abonnés existants peuvent demander à recevoir des copies en double des publications conservées.

Vous pouvez parfois ne pas souhaiter conserver des publications, même pour les informations d'état :

- Si tous les abonnements à une rubrique sont souscrits avant que des publications soient effectuées sur cette rubrique et que vous ne pas prévoyez pas ou n'autorisez pas de nouveaux abonnements, il n'est pas nécessaire de conserver les publications car celles-ci sont distribuées à l'ensemble des abonnés la première fois qu'elles sont publiées.
- Si des publications ont lieu fréquemment, par exemple, toutes les secondes, un nouvel abonné (ou un abonné en reprise après un échec) reçoit l'état actuel presque immédiatement après son abonnement initial et il n'est donc pas nécessaire de conserver ces publications.
- Si les publications sont volumineuses, vous risquez d'avoir besoin d'un espace de stockage considérable pour stocker les publication conservées pour chaque rubrique. Dans un environnement à plusieurs gestionnaires de files les publications conservées sont stockées par tous les gestionnaires de files d'attente du réseau qui ont un abonnement correspondant.

Lorsque vous décidez d'utiliser ou non les publications conservées, tenez compte de la manière dont les applications d'abonnement sont récupérées après un incident. Si le diffuseur de publications n'utilise pas de publications conservées, l'application d'abonné peut avoir besoin de stocker son état actuel en local.

Pour qu'une publication soit conservée, utilisez l'option d'insertion de message MQPMO_RETAIN. Si cette option est utilisée et que la publication ne peut pas être conservée, le message n'est pas publié et l'appel échoue avec MQRC_PUT_NOT_RETAINED.

Si un message est une publication conservée, cela est indiquée par la propriété de message MQIsRetained. La persistance d'un message est telle que lorsque celui-ci a été publié initialement.

Concepts associés

Remarques de conception pour les publications conservées dans les clusters de publication/abonnement

Publications sous le contrôle d'un point de synchronisation

Dans la fonction de publication/abonnement IBM MQ, le point de synchronisation peut être utilisé par des diffuseurs ou en interne par le gestionnaire de files d'attente.

Les diffuseurs utilisent le point de synchronisation lorsqu'ils émettent des appels MQPUT/MQPUT1 avec l'option MQPMO_SYNCPOINT. Tous les messages transmis aux abonnés font partie du nombre maximal de messages dans une unité d'oeuvre. L'attribut du gestionnaire de files d'attente MAXUMSGS spécifie cette limite. Si la limite est atteinte, l'éditeur reçoit le code anomalie 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED.

Lorsqu'un diffuseur émet des appels MQPUT/MQPUT1 à l'aide de MQPMO_NO_SYNCPOINT avec l'option MQPMO_RETAIN ou les options de distribution de rubrique NPMSGDLV/PMSGDLV avec des valeurs ALL ou ALLDUR, le gestionnaire de files d'attente utilise des points de synchronisation internes pour s'assurer que les messages sont transmis comme il convient. Le diffuseur de publications peut recevoir le code anomalie 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED si la limite est atteinte dans la portée de l'appel MQPUT/MQPUT1 de l'éditeur.

Abonnés et abonnements

Dans la publication/abonnement d'IBM MQ, un abonné est une application qui demande des informations sur une rubrique spécifique à un gestionnaire de files d'attente dans un réseau de publication/abonnement. Un abonné peut recevoir des messages sur la même rubrique ou des rubriques différentes, depuis plusieurs diffuseur de publications.

Des abonnements peuvent être créés manuellement via une commande MQSC ou par des applications. Ces abonnements sont envoyés au gestionnaire de files d'attente local et contiennent des informations sur les publications que l'abonné souhaite recevoir :

- La rubrique qui intéresse l'abonné ; il peut s'agir de plusieurs rubriques si des caractères génériques sont utilisés.
- Une chaîne de sélection facultative à appliquer aux messages publiés.
- Un descripteur vers une file d'attente (appelée *file d'attente de souscription*), dans laquelle les publications sélectionnées doivent être placées, et l'ID de corrélation (CorrelId) facultatif.

Le gestionnaire de files d'attente local stocke les informations d'abonnement. Lorsqu'il reçoit une publication, il analyse ces informations pour déterminer si un abonnement correspond à la rubrique et la chaîne de sélection de la publication. Pour chaque abonnement qui correspond, le gestionnaire de files d'attente dirige la publication vers la file d'attente de souscription de l'abonné. Les informations sur les abonnements stockées par gestionnaire de files d'attente peuvent être affichées à l'aide des commandes DIS SUB et DIS SBSTATUS.

Un abonnement est supprimé uniquement lorsque l'un des événements suivants se produit :

- L'abonné se désabonne à l'aide de l'appel MQCLOSE (si l'abonnement a été défini comme non durable).
- L'abonnement expire.
- L'abonnement est supprimé par l'administrateur système à l'aide de la commande DELETE SUB.
- L'application d'abonné s'arrête (si l'abonnement a été défini comme non durable).
- Le gestionnaire de files d'attente est arrêté ou redémarré (si l'abonnement a été défini comme non durable).

Lors de l'extraction des messages, utilisez les options appropriées dans l'appel MQGET. Si votre application traite uniquement les messages d'un abonnement, vous devez utiliser au moins `get-by-correlid`, comme illustré dans l'exemple de programme C `amqssbxa.c` et dans l'exemple Unmanaged MQ subscriber. Le **`CorrelId`** à utiliser est renvoyé par MQSUB dans le MQSD.Zone **`SubCorrelId`** .

Concepts associés

Abonnements clonés et partagés

Référence associée

Exemples de définition de la propriété sharedSubscription

Files d'attente gérées et publication/abonnement

Lorsque vous créez un abonnement vous pouvez choisir d'utiliser la mise en file d'attente gérée. Si vous utilisez une mise en file d'attente gérée, une file d'attente de souscription est créée automatiquement lorsque vous créez un abonnement. Les files d'attente gérées sont nettoyées automatiquement en fonction de la durabilité de l'abonnement. Avec l'utilisation de files d'attente gérées, vous n'avez pas besoin de créer des files d'attente pour recevoir des publications et les publications inutilisées sont supprimées automatiquement des files d'attente de souscription si une connexion d'abonnement non durable est fermée.

Si une application n'a pas besoin d'utiliser une file d'attente particulière comme file d'attente de souscription (destination des publications qu'elle reçoit), elle peut utiliser des *abonnements gérés* à l'aide de l'option d'abonnement MQSO_MANAGED. Si vous créez un abonnement géré, le gestionnaire de files d'attente renvoie un identificateur d'objet à l'abonné pour une file d'attente de souscription créée par le gestionnaire de files d'attente, dans laquelle les publications seront reçues. Cela est dû au fait qu'un *abonnement géré* a pour gestionnaire IBM MQ. L'identificateur d'objet de la file d'attente sera renvoyé pour vous permettre de rechercher, obtenir ou interroger la file d'attente. (Il n'est pas possible d'insérer des éléments dans une file d'attente gérée ou de définir des attributs pour celle-ci, sauf si on vous a accordé explicitement l'accès aux files d'attente dynamiques temporaires.)

La durabilité de l'abonnement détermine si la file d'attente gérée est conservée lorsque la connexion de l'application abonnée au gestionnaire de files d'attente est interrompue.

Les abonnements gérés sont particulièrement utiles dans le cas d'abonnements non durables, car sinon, lorsque la connexion de l'application est arrêtée, les messages inutilisés restent dans la file d'attente de l'abonné en y occupant indéfiniment de l'espace. Si vous utilisez un abonnement géré, la file d'attente gérée est file d'attente dynamique temporaire et est donc supprimée avec tous les messages inutilisés lorsque la connexion est interrompue pour l'une des raisons suivantes :

- MQCLOSE avec MQCO_REMOVE_SUB est utilisé et l'identificateur d'objet Hobj géré est fermé.
- Une connexion vers une application utilisant un abonnement non durable (MQSO_NON_DURABLE) est perdue.
- Un abonnement est supprimé car il est arrivé à expiration et l'identificateur Hobj géré est fermé.

Des abonnements gérés peuvent également être utilisés avec des abonnements durables mais il est possible que vous souhaitiez laisser les messages inutilisés dans la file d'attente de souscription pour pouvoir les récupérer lorsque la connexion est rouverte. C'est pourquoi les files d'attente gérées pour les abonnements durables prennent la forme d'une file d'attente dynamique permanente qui est conservée lorsque la connexion de l'application abonnée au gestionnaire de files d'attente est interrompue.

Vous pouvez définir un délai d'expiration pour votre abonnement si vous souhaitez utiliser une file d'attente gérée dynamique permanente qui, si elle sera conservée après l'interruption de la connexion, ne continuera pas d'exister indéfiniment.

Si vous supprimez la file d'attente gérée, vous recevez un message d'erreur.

Les files d'attente gérées qui sont créées sont nommées avec des nombres à la fin de leur nom (des horodatages) pour être uniques.

Durabilité des abonnements

Les abonnements peuvent être configurés pour être non durables ou durables. La durabilité des abonnements détermine ce qui se produit pour des abonnements lorsque des applications d'abonnement se déconnectent d'un gestionnaire de files d'attente.

Abonnements durables

Les abonnements durables continuent d'exister lorsque la connexion d'une application d'abonnement au gestionnaire de files d'attente est fermée. Si un abonnement est durable, lorsque l'application d'abonnement se déconnecte, l'abonnement est conservé et il peut être utilisé par l'application d'abonnement lorsque celle-ci se reconnecte et redemande l'abonnement en utilisant le même nom **SubName** que celui renvoyé lors de la création de l'abonnement.

Lors d'une souscription à un abonnement durable, un nom d'abonnement (**SubName**) est obligatoire. Les noms d'abonnement doivent être uniques dans un gestionnaire de files d'attente pour permettre d'identifier les abonnements. Cette méthode d'identification est nécessaire lorsque vous spécifiez un abonnement que vous souhaitez reprendre si vous avez volontairement fermé la connexion à l'abonnement (à l'aide de l'option MQCO_KEEP_SUB) ou si vous avez été déconnecté du gestionnaire de files d'attente. Vous pouvez reprendre un abonnement existant en utilisant l'appel MQSUB avec l'option MQSO_RESUME. Les noms d'abonnement sont également affichés si vous utilisez la commande DISPLAY SBSTATUS avec SUBTYPE ALL ou ADMIN.

Lorsqu'une application n'a plus besoin d'un abonnement durable, celui-ci peut être supprimé à l'aide de l'appel de fonction MQCLOSE avec l'option MQCO_REMOVE_SUB ou supprimé manuellement à l'aide de la commande MQSC DELETE SUB.

Vous pouvez utiliser l'attribut de rubrique **DURSUB** pour indiquer si des abonnements durables peuvent ou non être souscrits sur une rubrique.

Renvoyée à partir d'un appel MQSUB à l'aide de l'option MQSO_RESUME, le délai d'expiration d'abonnement est défini sur le délai d'expiration d'origine de l'abonnement et non sur le délai d'expiration restant.

Un gestionnaire de files d'attente continue d'envoyer des publications pour répondre à un abonnement durable même si l'application d'abonnement correspondante n'est pas connectée. Cela entraîne une accumulation de messages dans la file d'attente de l'abonné. Le moyen le plus simple pour éviter ce problème est d'utiliser un abonnement non durable chaque fois que cela convient. Cependant, lorsqu'il est nécessaire d'utiliser des abonnements durables, l'accumulation des messages peut être évitée si l'abonné souscrit à un abonnement à l'aide de l'option Publications conservées. Un abonné peut alors contrôler la réception des publications à l'aide de l'appel MQSUBRQ.

Abonnements non durables

Les abonnements non durables existent tant que la connexion entre l'application d'abonnement et le gestionnaire de files d'attente est ouverte. L'abonnement est supprimé lorsque l'application d'abonnement se déconnecte du gestionnaire de files d'attente volontairement ou suite à une perte de connexion. Lorsque la connexion est fermée, les informations concernant l'abonnement sont supprimées du gestionnaire de files d'attente, et n'apparaissent plus si vous affichez les abonnements à l'aide de la commande DISPLAY SBSTATUS. Aucun autre message n'est placé dans la file d'attente de souscription.

Ce qui se produit pour les publications non consommées dans une file d'attente de souscription pour des abonnements non durables est déterminé comme suit.

- Si une application d'abonnement utilise une destination gérée, toutes les publications qui n'ont pas été consommées sont automatiquement supprimées.
- Si l'application d'abonnement fournit un identificateur à sa propre file d'attente de souscription lorsqu'il s'abonne, les messages non consommés ne sont pas supprimés automatiquement. Il incombe à l'application de vider la file d'attente si c'est pertinent. Si la file d'attente est partagée par plusieurs abonnés ou d'autres applications point-à-point, cela peut ne pas être approprié de la vider complètement.

Bien que cela ne soit pas obligatoire pour les abonnements non durables, un nom d'abonnement, s'il est fourni, est utilisé par le gestionnaire de files d'attente. Les noms d'abonnement doivent être uniques dans le gestionnaire de files d'attente pour permettre d'identifier les abonnements.

Concepts associés

Abonnements clonés et partagés

Tâches associées

Utilisation d'abonnements partagés JMS 2.0

Référence associée

Exemples de définition de la propriété sharedSubscription

Chaînes de sélection

Une *chaîne de sélection* est une expression qui est appliquée à une publication pour déterminer si elle correspond à un abonnement. Les chaînes de sélection peuvent inclure des caractères génériques.

Lorsque vous vous abonnez, en plus d'une rubrique, vous pouvez spécifier une chaîne de sélection pour sélectionner les publications en fonction de leurs propriétés de message.

La chaîne de sélection est évaluée par rapport au message inséré par le diffuseur de publications avant d'être modifiée pour la distribution à chaque abonné. Faites attention lorsque vous utilisez des zones dans la chaîne de sélection, susceptibles d'être modifiées dans le cadre de l'opération de publication. Par exemple, les zones MQMD `UserIdentifier`, `MsgId` et `CorrelId`.

Les chaînes de sélection ne doivent pas faire référence à des zones de propriété de message ajoutées par le gestionnaire de files d'attente dans le cadre de l'opération de publication (voir Propriétés des messages de publication/abonnement), à l'exception de la propriété de message `MQTopicString`, qui contient la chaîne de rubrique pour la publication.

Concepts associés

Règles et restrictions des chaînes de sélection

Rubriques

Une rubrique est le sujet d'une information publiée dans un message de publication/abonnement.

Les messages dans les systèmes point-à-point sont envoyés à une adresse de destination spécifique. Les messages se trouvant dans des systèmes basés sur des sujets de publication/abonnement sont envoyés à des abonnés en fonction des sujets décrivant le contenu du message. Dans des systèmes basés sur le contenu, les messages sont envoyés aux abonnés en fonction du contenu du message.

Le système de publication/abonnement IBM MQ est un système de publication/abonnement basé sur les sujets. Un diffuseur de publication crée un message et le publie avec une chaîne de rubrique qui convient le mieux au sujet de la publication. Pour recevoir des publications, un abonné crée un abonnement avec un schéma correspondant à la chaîne de rubrique pour sélectionner les rubriques de publication. Le gestionnaire de files d'attente distribue les publications aux abonnés dont les abonnements correspondent à la rubrique de publication, et qui sont autorisés à recevoir les publications. L'article, «Chaînes de rubrique», à la page 74, décrit la syntaxe des chaînes de rubrique qui identifient le sujet d'une publication. Les abonnés créent également des chaînes de rubriques pour sélectionner les rubriques qu'ils souhaitent recevoir. Les chaînes de rubrique que les abonnés créent peuvent contenir un des deux schémas de caractères génériques alternatifs pour correspondre aux chaînes de rubrique des publications. La correspondance des schémas est décrite dans «Schémas de caractères génériques», à la page 75.

Dans une publication/un abonnement basé sur le sujet, les diffuseurs de publications ou les administrateurs, ont pour tâche de classifier les sujets dans les rubriques. Généralement, les sujets sont classés dans un ordre hiérarchique, dans des arborescences de rubriques, à l'aide du caractère `'/'` pour créer de sous-rubriques dans la chaîne de rubrique. Voir «Arborescence de rubriques», à la page 81 pour des exemples d'arborescences de rubriques. Les rubriques sont des noeuds de l'arborescence. Les rubriques sont des noeuds qui ne comportent aucune sous-rubrique, ou des noeuds intermédiaires avec des sous-rubriques.

Parallèlement à l'organisation de rubriques en arborescence hiérarchique de rubriques, vous pouvez associer des rubriques à des objets de rubriques d'administration. Vous affectez des attributs à une rubrique, par exemple un attribut indiquant si la rubrique est distribuée dans un cluster, en l'associant à un objet de rubrique d'administration. L'association est faite en nommant la rubrique à l'aide de l'attribut `TOPICSTR` de l'objet de rubrique d'administration. Si vous n'associez pas explicitement un objet de rubrique d'administration à une rubrique, elle hérite des attributs de son ancêtre le plus proche dans l'arborescence de rubriques que vous *avez* associé à un objet de rubrique d'administration. Si vous n'avez pas défini de rubrique parent, elle hérite de `SYSTEM.BASE.TOPIC`. Les objets de rubrique d'administration sont décrits dans «Objets de rubrique d'administration», à la page 82.

Remarque : Même si vous héritez de tous les attributs d'une rubrique de `SYSTEM.BASE.TOPIC`, définissez une rubrique principale pour vos rubriques qui hérite directement de `SYSTEM.BASE.TOPIC`.

Par exemple, dans l'espace de rubrique des États américains, `USA/Alabama` `USA/Alaska`, et ainsi de suite, USA est le sujet principal. L'objectif premier de la rubrique principale est de créer des espaces de sujet discrets et sans chevauchements afin d'éviter que des publications soient associées aux mauvais abonnements. Cela signifie également que vous pouvez modifier les attributs de votre rubrique principale pour affecter l'espace de sujet dans son intégralité. Par exemple, vous pouvez définir le nom de l'attribut **CLUSTER**.

Lorsque vous faites référence à une rubrique en tant que diffuseur de publications ou en tant qu'abonné, vous avez le choix entre fournir une chaîne de rubrique ou faire référence à un objet de rubrique. Vous pouvez également faire les deux, auquel cas la chaîne de rubrique que vous avez fournie définit une sous-rubrique de l'objet de rubrique. Le gestionnaire de files d'attente identifie la rubrique en ajoutant la chaîne de rubrique au préfixe de la chaîne de rubrique nommé dans l'objet de rubrique, en insérant un ' / ' additionnel entre les deux chaînes de rubriques, par exemple, *chaîne de rubrique/chaîne d'objet*. «Combinaison de chaînes de rubrique», à la page 79 décrit cela en plus de détails. La chaîne de rubrique résultante est utilisée pour identifier la rubrique et l'associer à un objet de rubrique d'administration. L'objet de rubrique d'administration n'est pas nécessairement le même objet de rubrique que l'objet de rubrique correspondant à la rubrique principale.

Dans une publication/un abonnement basé sur le contenu, vous définissez les messages que vous souhaitez recevoir en fournissant des chaînes de sélection recherchant le contenu de chaque message. IBM MQ fournit une forme intermédiaire de publication/abonnement basée sur le contenu à l'aide des sélecteurs de message qui analysent les propriétés du message plutôt que le contenu complet du message. Voir Sélecteurs. L'utilité archétype des sélecteurs de messages est de s'abonner à une rubrique et ensuite de qualifier la sélection avec une propriété numérique. Le sélecteur vous permet d'indiquer les valeurs qui vous intéresse uniquement dans une certaine plage ; quelque chose que vous ne pouvez pas faire en utilisant un caractère ou des caractères génériques basés sur des rubriques. Si vous devez filtrer en vous basant sur le contenu complet du message, vous devez utiliser IBM Integration Bus.

Chaînes de rubrique

Le libellé de la rubrique est publiée à l'aide d'une chaîne de rubrique. Abonnez-vous aux groupes de rubriques en utilisant des chaînes de caractères génériques de type caractère ou de type rubrique.

Rubriques

Une *chaîne de rubrique* est une chaîne de caractères qui identifie la rubrique d'un message de publication/abonnement. Vous pouvez utiliser n'importe quels caractères lorsque vous construisez une chaîne de rubrique.

Chaîne de rubrique

Trois caractères ont une signification spéciale dans la fonction de publication/abonnement de la IBM WebSphere MQ 7. Ils sont autorisés dans une chaîne de rubrique, mais utilisez-les avec précaution. L'utilisation des caractères spéciaux est expliquée dans «Schéma de caractères génériques basés sur des rubriques», à la page 75.

Barre oblique (/)

Séparateur de niveaux de rubrique. Utilisez le caractère ' / ' pour structurer la rubrique dans une arborescence de rubriques.

Evitez les niveaux de rubrique vides, ' // ' dans la mesure du possible. Ils correspondent à des noeuds de la hiérarchie de rubriques sans chaîne de rubrique. Une barre oblique ' / ' de début ou de fin dans une chaîne de rubrique correspond à un noeud vide de début ou de fin et doit être aussi évitée.

Signe dièse (#)

Combiné avec '/' pour construire un caractère générique multi-niveau dans des abonnements. Veillez à utiliser '#' à côté de '/' dans des chaînes de rubrique servant à nommer des rubriques publiées. «Exemples de chaînes de rubrique», à la page 75 montre une utilisation appropriée de '#'.

Les chaînes '.../#/...', '#/...' et '.../#' ont une signification particulière dans les chaînes de rubrique d'abonnement. Elles correspondent à toutes les rubriques à un ou plusieurs niveaux dans la hiérarchie de rubriques. Par conséquent, si vous avez créé une rubrique avec l'une de ces séquences, vous ne pouvez pas vous y abonner sans aussi vous abonner à toutes les rubriques à plusieurs niveaux de la hiérarchie de rubriques.

Signe plus (+)

Combiné avec '/' pour construire un caractère générique à un niveau dans des abonnements. Veillez à utiliser '+' à côté de '/' dans des chaînes de rubrique servant à nommer des rubriques publiées.

Les chaînes '.../+/...', '+/...' et '.../+' ont une signification particulière dans les chaînes de rubrique d'abonnement. Elles correspondent à toutes les rubriques à un niveau dans la hiérarchie de rubriques. Par conséquent, si vous avez créé une rubrique avec l'une de ces séquences, vous ne pouvez pas vous y abonner sans aussi vous abonner à toutes les rubriques à un niveau de la hiérarchie de rubriques.

Exemples de chaînes de rubrique

```
IBM/Business Area#/Results
IBM/Diversity/%African American
```

Référence associée

TOPIC

Schémas de caractères génériques

Deux schémas de caractères génériques sont utilisés pour s'abonner à plusieurs rubriques. Le choix du schéma est une option d'abonnement.

MQSO_WILDCARD_TOPIC
Sélection des rubriques auxquelles s'abonner à l'aide du schéma de caractères génériques basés sur les rubriques.

Il s'agit du schéma par défaut si aucun schéma générique n'est sélectionné par défaut.

MQSO_WILDCARD_CHAR
Sélection des rubriques auxquelles s'abonner à l'aide du schéma de caractères génériques de type caractère.

Définissez l'un des schémas en spécifiant le paramètre **wschema** sur la commande DEFINE SUB. Pour plus d'informations, voir DEFINE SUB.

Remarque : Les abonnements créés avant IBM WebSphere MQ 7.0 utilisent toujours le schéma de caractères génériques.

Exemples

```
IBM/+/Results
#/Results
IBM/Software/Results
IBM/*ware/Results
```

Schéma de caractères génériques basés sur des rubriques

Les caractères génériques basés sur des rubriques permettent de s'abonner à plusieurs rubriques en même temps.

Les caractères génériques basés sur les rubriques représentent une fonction puissante du système de rubriques dans la publication/l'abonnement IBM MQ. Les caractères génériques multi-niveau et à un niveau peuvent être utilisés pour les abonnements, mais ils ne peuvent pas l'être par le diffuseur d'un message dans une rubrique.

Le schéma de caractères génériques basés sur des rubriques vous permet de sélectionner des publications regroupées par niveau de rubrique. Vous pouvez décider, pour chaque niveau de la hiérarchie de rubriques, si la chaîne dans l'abonnement pour ce niveau de rubrique doit correspondre exactement à la chaîne dans la publication ou pas. Par exemple, l'abonnement `IBM/+/Results` sélectionne toutes les rubriques.

```
IBM/Software/Results
IBM/Services/Results
IBM/Hardware/Results
```

Il existe deux types de caractères génériques.

Caractère générique multi-niveau

- Le caractère générique multi-niveau est utilisé dans des abonnements. Lorsqu'il est utilisé dans une publication, il est traité comme un littéral.
- Le caractère générique multi-niveau `'#'` est utilisé pour correspondre aux divers niveaux d'une rubrique. Par exemple, dans l'exemple d'arborescence de rubriques, si vous vous abonnez à `'USA/Alaska/#'`, vous recevez des messages sur les rubriques `'USA/Alaska'` et `'USA/Alaska/Juneau'`.
- Le caractère générique multi-niveau peut représenter zéro, un ou plusieurs niveaux. Par conséquent, `'USA/#'` peut également correspondre à `'USA'`, `'#'` ne représentant alors aucun niveau. Le séparateur de niveaux de rubrique ne sert à rien dans ce contexte, car il n'y a aucun niveau à séparer.
- Le caractère générique multi-niveau ne s'applique que s'il est spécifié seul ou après le caractère séparateur de niveaux de rubrique. Par conséquent, `'#'` et `'USA/#'` sont des rubriques valides, dans lesquelles le caractère `'#'` est traité comme un caractère générique. Toutefois, même si `'USA#'` est également une chaîne de rubrique valide, le caractère `'#'` n'est pas considéré comme un caractère générique et ne possède pas de signification spéciale. Pour plus d'informations, voir «Cas où les caractères génériques basés sur des rubriques ne sont pas génériques», à la page 78.

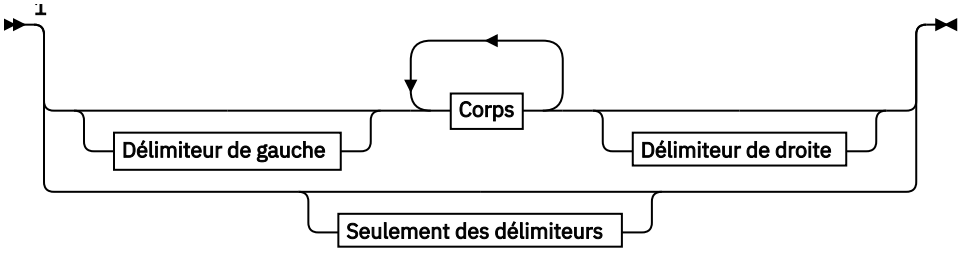
Caractère générique à un niveau

- Le caractère générique à un niveau est utilisé dans les abonnements. Lorsqu'il est utilisé dans une publication, il est traité comme un littéral.
- Le caractère générique à un niveau `'+'` ne correspond qu'à un seul niveau de rubrique. Par exemple, `'USA/+'` correspond à `'USA/Alabama'`, mais pas à `'USA/Alabama/Auburn'`. Le caractère générique à un niveau ne correspondant qu'à un seul niveau, `'USA/+'` ne correspond pas à `'USA'`.
- Le caractère générique à un niveau peut être utilisé à tout niveau de l'arborescence de rubriques et avec le caractère générique multi-niveau. Le caractère générique à un niveau doit être spécifié après le séparateur de niveaux de rubrique, sauf s'il est spécifié seul. Par conséquent, `'+'` et `'USA/+'` sont des rubriques valides, dans lesquelles le caractère `'+'` est traité comme un caractère générique. Toutefois, même si `'USA+'` est également une chaîne de rubrique valide, le caractère `'+'` n'est pas considéré comme un caractère générique et ne possède pas de signification spéciale. Pour plus d'informations, voir «Cas où les caractères génériques basés sur des rubriques ne sont pas génériques», à la page 78.

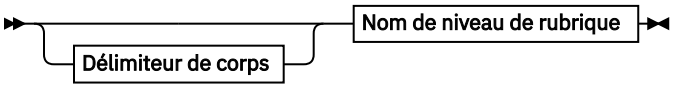
La syntaxe pour le schéma de caractères génériques basés sur des rubriques ne comporte pas de caractères d'échappement. Le fait que `'#'` et `'+'` soient traités ou non comme des caractères génériques dépend de leur contexte. Pour plus d'informations, voir «Cas où les caractères génériques basés sur des rubriques ne sont pas génériques», à la page 78.

Remarque : Le début et la fin d'une chaîne de rubrique sont traités de manière spéciale. Si vous utilisez '$' pour indiquer la fin de la chaîne, '$#/...' est un caractère générique à plusieurs niveaux et '$/#/..'. est un noeud vide à la racine, suivi par un caractère générique multi-niveau.

Chaîne de caractères génériques basés sur des rubriques

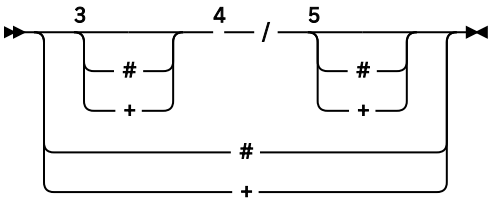


Corps



Nom de niveau de rubrique

Seulement des délimiteurs



Remarques :

[1] Une chaîne de rubrique de longueur NULL ou égale à zéro n'est pas valide

[2] Il est conseillé de ne pas utiliser de `*, ?, %` dans les chaînes de nom de niveau pour la compatibilité entre les schémas de caractères génériques basés sur les caractères et les caractères génériques basés sur les rubriques.

[3] Ces cas sont équivalents au masque de *délimiteur de gauche*.

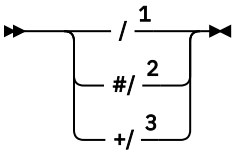
[4] / sans caractères génériques correspond à une seule rubrique vide.

[5] Ces cas sont équivalents au masque de *délimiteur de droite*.

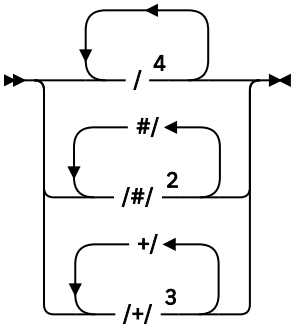
[6] Correspond à chaque rubrique.

[7] Correspond à chaque rubrique là où il n'y a qu'un seul niveau.

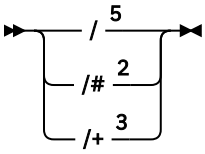
Délimiteur de gauche



Délimiteur de corps



Délimiteur de droite



Remarques :

[1] La chaîne de rubrique commence par une chaîne vide
[2] Correspond à zéro ou plusieurs niveaux. Plusieurs chaînes de correspondance multi-niveau produisent le même effet qu'une seule chaînes de correspondance multi-niveau.
[3] Correspond exactement à un niveau.
[4] // est une rubrique vide - un objet de rubrique sans chaîne de rubrique.
[5] La chaîne de rubrique se termine par une chaîne vide

Cas où les caractères génériques basés sur des rubriques ne sont pas génériques

Les caractères génériques '+' et '#' n'ont pas de signification spéciale lorsqu'ils sont mélangés à d'autres caractères (y compris eux-mêmes) dans un niveau de rubrique.

Cela signifie que les rubriques qui contiennent '+' ou '#' avec d'autres caractères dans un niveau de rubrique peuvent être publiées.

Etudions par exemple les deux rubriques suivantes :

1. level0/level1/+/level4/#
2. level0/level1/#+/level4/level#

Dans le premier exemple, les caractères '+' et '#' sont traités comme des caractères génériques et ne sont donc pas valides dans une chaîne de rubrique cible de la publication, mais ils sont valides dans un abonnement.

Dans le second exemple, les caractères '+' et '#' ne sont pas traités comme des caractères génériques et la chaîne de rubrique peut donc être la cible d'une publication et d'un abonnement.

Exemples

```
IBM/+/Results
#/Results
IBM/Software/Results
```

Schéma de caractères génériques (de type caractère)
Le schéma de caractères génériques (de type caractère) vous permet de sélectionner des rubriques en fonction de la correspondance de caractères standard.

Vous pouvez sélectionner toutes les rubriques à plusieurs niveaux dans une hiérarchie de rubriques à l'aide de la chaîne '*'. L'utilisation de '*' dans le schéma de caractères génériques (de type caractère) est équivalente à l'utilisation de la chaîne de caractères génériques basée sur des rubriques '#'.

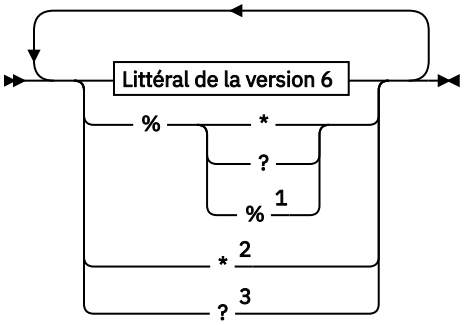
' *x*/*/*y* ' est équivalent à ' *x*/#/*y* ' dans le schéma basé sur les rubriques et sélectionne toutes les rubriques de la hiérarchie de rubriques entre les niveaux ' *x* et *y* ', où ' *x* ' et ' *y* ' sont des noms de rubrique qui ne figurent pas dans l'ensemble de niveaux renvoyés par le caractère générique.

'/+/' dans le schéma basé sur les rubriques n'a pas d'équivalent exact dans le schéma basé sur les caractères. 'IBM/*/Results' sélectionnera également 'IBM/Patents/Software/Results'. Vous pouvez construire des requêtes avec les deux schémas qui renvoient des correspondances identiques uniquement si l'ensemble de noms de rubrique à chaque niveau de la hiérarchie est unique.

En général, '*' et '?' utilisés dans le schéma basé sur des caractères n'ont pas d'équivalents dans le schéma basé sur des rubriques. Le schéma basé sur des rubriques n'exécute pas de correspondance partielle à l'aide de caractères génériques. L'abonnement générique basé sur des caractères 'IBM/*ware/Results' ne possède pas d'équivalent basé sur des rubriques.

Remarque : Les correspondances utilisant les abonnements génériques (de type caractère) sont plus lentes que les correspondances utilisant les abonnements basés sur des rubriques.

Chaîne de caractères génériques (de type caractère)



Littéral de la version 6

Remarques :

[1] Signifie "Insérer un caractère d'échappement devant le caractère suivant", de sorte qu'il soit traité en tant que littéral. '%' doit être suivi de '*', '?' ou '%'. Voir «Exemples de chaînes de rubrique», à la page 75.

[2] Signifie "Correspondance avec zéro ou plusieurs caractères" dans un abonnement.

[3] Signifie "Correspondance exacte avec un seul caractère" dans un abonnement.

Exemples

```
IBM/*/Results
IBM/*ware/Results
```

Combinaison de chaînes de rubrique

Lors de la création d'abonnement ou de l'ouverture de rubriques pour y publier des messages, la chaîne de rubrique peut être constituée en combinant deux chaînes de sous-rubrique distinctes, ou "sous-rubriques". Une sous-rubrique est fournie par l'application ou la commande d'administration comme chaîne de rubrique, et l'autre est la chaîne de rubrique associée à l'objet de rubrique. Vous pouvez utiliser la sous-rubrique comme chaîne de rubrique seule ou les combiner pour former un nouveau nom de rubrique.

Par exemple, vous pouvez définir un abonnement en utilisant la commande MQSC **DEFINE SUB**. La commande peut prendre **TOPICSTR** (chaîne de rubrique) ou **TOPICOBJ** (objet de rubrique) comme attribut ou les deux. Si seul **TOPICOBJ** est fourni, la chaîne de rubrique associée à l'objet de rubrique est utilisée comme chaîne de rubrique. Si seul **TOPICSTR** est fourni, il est utilisé comme chaîne de rubrique. Si les deux sont fournis, ils sont concaténés pour former une chaîne de rubrique unique de format **TOPICOBJ** / **TOPICSTR**, où la chaîne de rubrique configurée **TOPICOBJ** est toujours placée en premier et les deux parties de la chaîne sont toujours séparées par le caractère "/".

De même, dans programme MQI, le nom de rubrique complet est créé par MQOPEN. Il est composé de deux zones utilisées dans des appels MQI de publication/abonnement, dans l'ordre indiqué :

1. L'attribut **TOPICSTR** de l'objet de rubrique, nommé dans la zone **ObjectName**.
2. Le paramètre **ObjectString** définissant la sous-rubrique fournie par l'application.

La chaîne de rubrique résultante est renvoyée dans le paramètre **ResObjectString**.

Ces zones sont considérées comme étant présentes si le premier caractère de chaque zone n'est pas vide ou un caractère nul et si la longueur de la zone est supérieure à zéro. Si seule une zone est présente, elle est utilisée telle quelle comme nom de rubrique. Si aucune zone n'est renseignée, l'appel échoue avec le code anomalie MQRC_UNKNOWN_OBJECT_NAME ou MQRC_TOPIC_STRING_ERROR si le nom de rubrique complet n'est pas valide.

Si les deux zones sont présentes, un caractère "/" est inséré entre les deux éléments du nom de rubrique combiné résultant.

Le tableau suivant présente des exemples de concaténation de chaînes de rubrique:

Tableau 2. Exemples de concaténation de chaînes de rubrique

TOPICSTR de l'objet de rubrique	**Chaîne de rubrique fournie par l'application ou la commande DEFINE SUB**	**Nom complet de la rubrique**	**Commentaire**
Football/Scores	' '	Football/Scores	La chaîne TOPICSTR de l'objet de rubrique est utilisée seule.
' '	Football/Scores	Football/Scores	La chaîne ObjectString/ TOPICSTR est utilisée seule.
Football	Scores	Football/Scores	Le caractère "/" est ajouté au point de concaténation.
Football	/Scores	Football//Scores	Un 'noeud vide' est généré entre les deux chaînes. Différent de "Football/Scores".
/Football	Scores	/Football/Scores	La rubrique commence par un 'noeud vide'. Différent de "Football/ Scores".

Le caractère "/" est considéré comme un caractère spécial fournissant la structure au nom complet de la rubrique dans in«Arborescence de rubriques», à la page 81. Le caractère "/" ne doit pas être utilisé dans tous les autres cas, car la structure de l'arborescence des rubriques serait affectée. La rubrique "/Football" est différente de la rubrique "Football".

Remarque : Si vous utilisez un objet de rubrique lors de la création d'un abonnement, la valeur de la chaîne de rubrique de l'objet de rubrique est fixe dans l'abonnement à un moment donné. Toute modification consécutive de l'objet de rubrique n'affecte pas la chaîne de rubrique où l'abonnement est défini.

Caractères génériques dans les chaînes de rubrique

Les caractères génériques suivants sont des caractères spéciaux :

- signe plus (+)
- signe dièse (#)
- astérisque (*)
- point d'interrogation (?)

Les caractères génériques ont une signification spéciale uniquement lorsqu'ils sont utilisés par un abonnement. Ces caractères ne sont pas valides lorsqu'ils sont utilisés autre part, mais vous devez savoir comment ils sont utilisés, et il est peut-être préférable de ne pas les utiliser dans les chaînes de rubrique lors de la publication ou de la définition des objets de rubrique.

Si vous publiez une chaîne de rubrique avec # ou + combiné avec d'autres caractères (y compris eux-mêmes) dans un niveau de rubrique, la chaîne de rubrique peut faire l'objet d'un abonnement avec l'un ou l'autre des schémas de caractère générique.

Si vous publiez une chaîne de rubrique avec # ou + comme seul caractère entre deux caractères /, la chaîne de rubrique ne peut pas faire l'objet d'un abonnement explicitement par une application qui utilise MQSO_WILDCARD_TOPIC. Il en résulte que l'application reçoit davantage de publications que prévu.

N'utilisez pas de caractère générique dans la chaîne de rubrique d'un objet de rubrique défini. Dans le cas contraire, le caractère est traité comme caractère littéral lorsque l'objet est utilisé par un diffuseur, et comme caractère générique lorsqu'il est utilisé par un abonnement. Cela peut générer une confusion.

Exemple de fragment de code

Ce fragment de code, extrait de l'exemple de programme Exemple 2 : publication dans une rubrique variable, combine un objet de rubrique et une chaîne de rubrique variable :

```
MQOD   td = {MQOD_DEFAULT}; /* Object Descriptor      */
td.ObjectType = MQOT_TOPIC;  /* Object is a topic      */
td.Version = MQOD_VERSION_4;  /* Descriptor needs to be V4  */
strncpy(td.ObjectName, topicName, MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

Arborescence de rubriques

Chaque rubrique que vous définissez correspond à un élément ou à un noeud de l'arborescence de rubriques. Cette dernière peut être vide pour commencer ou contenir des rubriques définies précédemment à l'aide de commandes MQSC ou PCF. Vous pouvez définir une nouvelle rubrique à l'aide des commandes de création de rubrique ou en spécifiant la rubrique pour la première fois dans une publication ou un abonnement.

Vous pouvez utiliser toute chaîne de caractères pour définir la chaîne de rubrique d'une rubrique, mais il est conseillé de choisir une chaîne de rubrique qui s'intègre dans une structure d'arborescence hiérarchique. Une conception soignée des chaînes de rubrique et des arborescences de rubriques peut vous aider dans le cadre des opérations suivantes :

- Abonnement à plusieurs rubriques.
- Etablissement de règles de sécurité.

Vous pouvez construire une arborescence de rubriques sous la forme d'une structure plate et linéaire, mais il est recommandé de générer une arborescence de rubriques dans une structure hiérarchique comportant une ou plusieurs rubriques racine. Pour plus d'informations sur la planification de la sécurité et les rubriques, voir Sécurité de publication/abonnement.

La Figure 18, à la page 82 illustre un exemple d'arborescence de rubriques avec une rubrique racine.

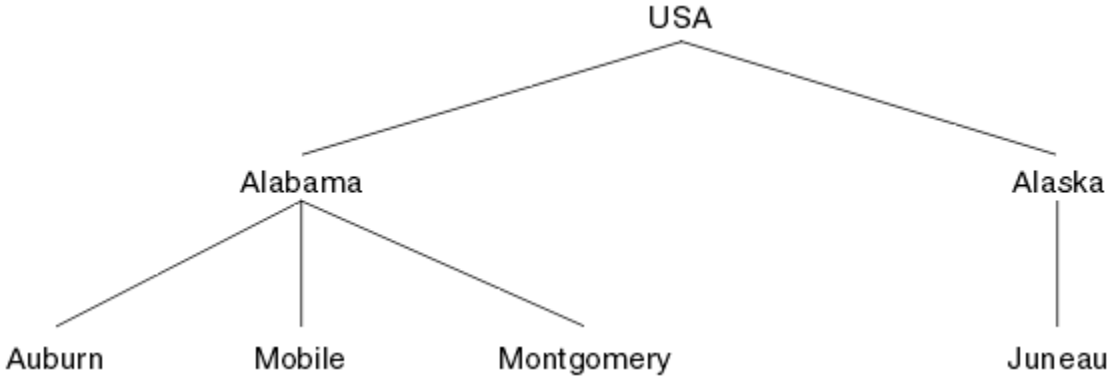


Figure 18. Exemple d'arborescence de rubriques

Chaque chaîne de caractères de la figure représente un noeud de l'arborescence de rubriques. Une chaîne de rubrique complète est créée en agrégeant les noeuds d'un ou plusieurs niveaux de l'arborescence de rubriques. Les niveaux sont séparés par le caractère "/". Le format d'une chaîne de rubrique intégralement spécifiée est le suivant : "racine/niveau2/niveau3".

Les rubriques valides de l'arborescence de rubriques illustrée dans la Figure 18, à la page 82 sont les suivantes :

"USA"
"USA/Alabama"
"USA/Alaska"
"USA/Alabama/Auburn"
"USA/Alabama/Mobile"
"USA/Alabama/Montgomery"
"USA/Alaska/Juneau"

Lorsque vous concevez des chaînes de rubrique et des arborescences de rubriques, n'oubliez pas que le gestionnaire de files d'attente n'interprète pas la chaîne de rubrique elle-même et qu'il n'essaye pas d'en déterminer la signification. Il utilise simplement la chaîne de rubrique pour envoyer des messages sélectionnés aux abonnés de cette rubrique.

Les principes suivants s'appliquent à la construction et au contenu d'une arborescence de rubriques :

- Le nombre de niveaux d'une arborescence de rubriques est illimité.
- La longueur du nom d'un niveau dans une arborescence de rubriques est illimitée.
- Le nombre de noeuds "racine" est illimité (le nombre d'arborescences de rubriques l'est donc également).

Tâches associées

Réduction du nombre de rubriques non souhaitées dans l'arborescence de rubrique

Objets de rubrique d'administration

A l'aide d'un objet de rubrique d'administration, vous pouvez affecter des attributs spécifiques et autres que ceux par défaut aux rubriques.

Figure 19, à la page 83 illustre comment la rubrique de haut niveau `Sport` qui est divisée en plusieurs rubriques couvrant divers sports peut être visualisée comme arborescence de rubriques :

Figure 19. Visualisation d'une arborescence de rubriques

Figure 20, à la page 83 illustre comment l'arborescence de rubriques peut être à nouveau divisée en sous-rubriques traitant de différents types d'informations sur chaque sport :

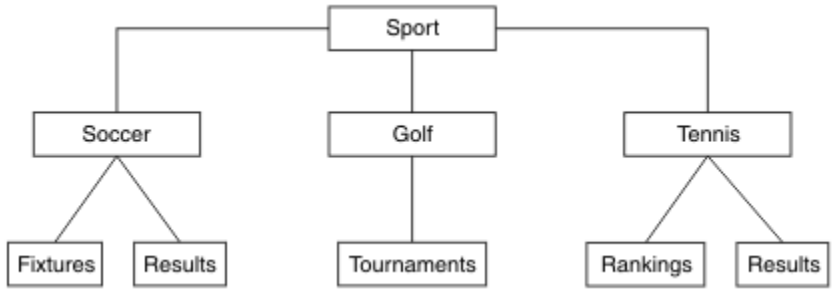


Figure 20. Arborescence de rubriques étendue

Pour créer l'arborescence de rubriques illustrée, il n'est pas nécessaire de définir des objets de rubrique d'administration. Chaque noeud figurant dans l'arborescence est défini par une chaîne de rubrique créée dans une opération de publication ou d'abonnement. Chaque rubrique de l'arborescence hérite des attributs de son parent. Les attributs sont hérités de l'objet de rubrique parent, car par défaut, tous les attributs sont définis sur ASPARENT. Dans cet exemple, toutes les rubriques possèdent les mêmes attributs que la rubrique `Sport`. La rubrique `Sport` n'a pas d'objet de rubrique d'administration et hérite de ses attributs de SYSTEM.BASE.TOPIC.

Notez qu'il est déconseillé d'octroyer des droits d'accès à des utilisateurs non mqm au niveau du noeud racine de l'arborescence de rubriques (SYSTEM.BASE.TOPIC), car les droits sont hérités et ne peuvent pas être restreints. De ce fait, en accordant des droits d'accès à ce niveau, vous octroyez des droits d'accès à la totalité de l'arborescence. Vous devez accorder les droits d'accès à un niveau de rubrique inférieur dans la hiérarchie.

Les objets de rubrique d'administration peuvent être utilisés pour définir des attributs spécifiques pour des noeuds particuliers dans l'arborescence de rubriques. Dans l'exemple suivant, l'objet de rubrique d'administration est défini pour attribuer la valeur NO à la propriété d'abonnements durables DURSUB de la rubrique Football :

```
DEFINE TOPIC(FOOTBALL.EUROPEAN)
TOPICSTR('Sport/Soccer')
DURSUB(NO)
DESCR('Administrative topic object to disallow durable subscriptions')
```

L'arborescence de rubriques peut être consultée sous forme de :

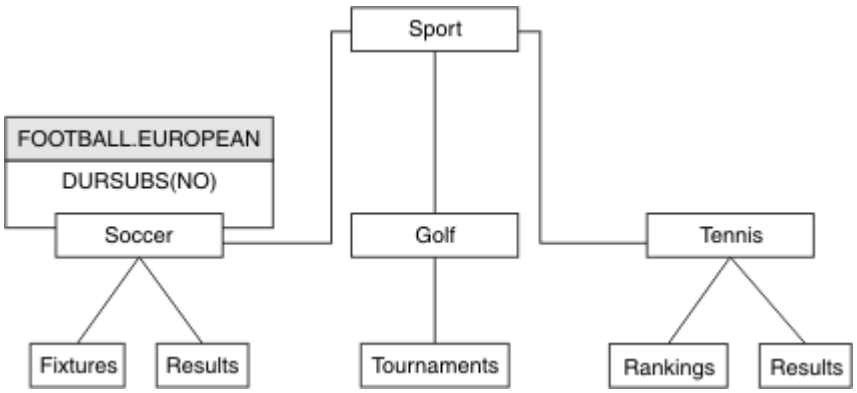


Figure 21. Visualisation d'un objet de rubrique d'administration associé à la rubrique Sport/Football

Toutes les applications qui s'abonnent aux rubriques sous Football dans l'arborescence peuvent toujours utiliser les chaînes de rubrique utilisées avant l'ajout de l'objet de rubrique d'administration. Cependant, une application peut être écrite pour s'abonner à l'aide du nom d'objet `FOOTBALL.EUROPEAN` au lieu de la chaîne `/Sport/Soccer`. Par exemple, pour s'abonner à `/Sport/Soccer/Results`, une application peut spécifier `MQSD.ObjectName` en tant que `FOOTBALL.EUROPEAN` et `MQSD.ObjectString` en tant que `Results`.

Avec cette fonction, vous pouvez masquer une partie de l'arborescence de rubriques vis-à-vis des développeurs d'applications. Définissez un objet de rubrique d'administration sur un noeud spécifique dans l'arborescence de rubriques, puis les développeurs peuvent définir leurs propres rubriques en tant que rubriques enfant du noeud. Les développeurs doivent uniquement connaître la rubrique parent, et non les autres noeuds de l'arborescence parent.

Héritage des attributs

Si une arborescence de rubriques comporte plusieurs objets de rubrique d'administration, par défaut chaque objet hérite ses attributs de la rubrique d'administration parent la plus proche. L'exemple précédent a été étendu dans Figure 22, à la page 84 :

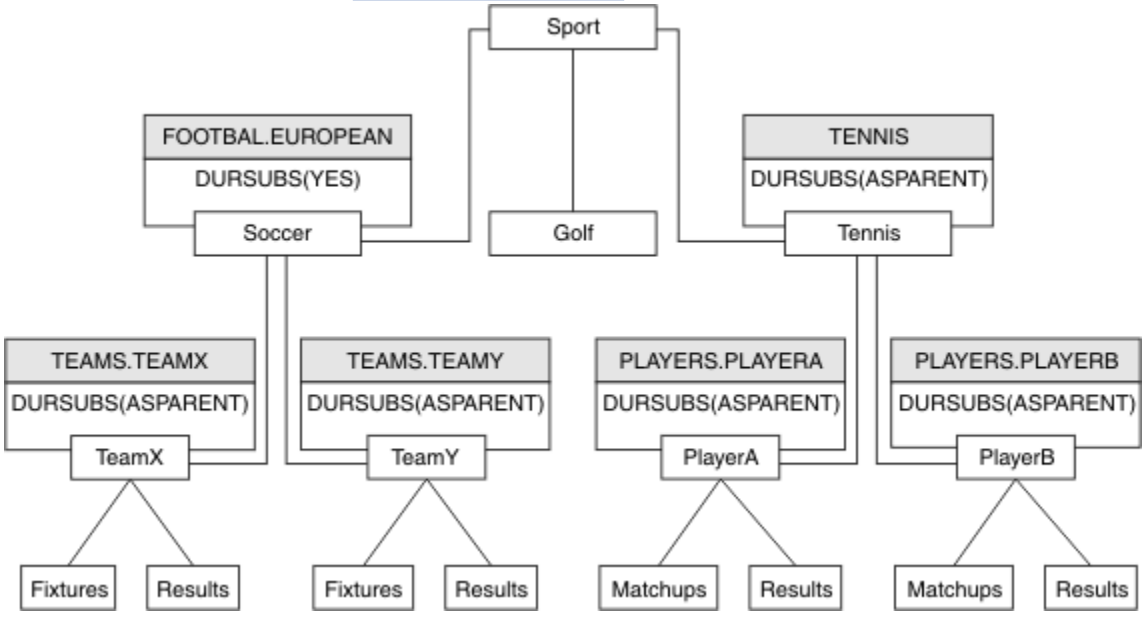


Figure 22. Arborescence de rubriques avec plusieurs objets de rubrique d'administration

Par exemple, utilisez la fonction d'héritage pour attribuer à toutes les rubriques enfant de `/Sport/Soccer` la propriété où les abonnements sont non durables. Remplacez l'attribut `DURSUB` de `FOOTBALL.EUROPEAN` par `NO`.

Cet attribut peut être défini à l'aide de la commande suivante :

```
ALTER TOPIC(FOOTBALL.EUROPEAN) DURSUB(NO)
```

La propriété DURSUB de tous les objets de rubrique d'administration des rubriques enfant de `Sport/Soccer` a pour valeur ASPARENT (valeur par défaut). Après avoir modifié la valeur de la propriété DURSUB de `FOOTBALL.EUROPEAN` en NO, les rubriques enfant de `Sport/Soccer` héritent de la valeur de propriété DURSUB NO. Toutes les rubriques enfant de `Sport/Tennis` héritent de la valeur de DURSUB de l'objet `SYSTEM.BASE.TOPIC` . `SYSTEM.BASE.TOPIC` est doté de la valeur YES.

La tentative de créer un abonnement durable à la rubrique `Sport/Soccer/TeamX/Results` va échouer ; cependant, il est possible de créer un abonnement durable à `Sport/Tennis/PlayerB/Results`.

Contrôle de l'utilisation de caractères génériques avec la propriété WILDCARD

Utilisez la propriété WILDCARD du paramètre **Topic** de MQSC ou la propriété `WildcardOperation` équivalente de la commande PCF Topic pour contrôler la distribution des publications sur les applications abonnées qui utilisent les noms de chaîne de rubrique générique. La propriété WILDCARD peut avoir deux valeurs possibles :

WILDCARD
Comportement des abonnements génériques par rapport à cette rubrique.

PASSTHRU
Les abonnements à une rubrique générique moins spécifique que la chaîne de rubrique dans cet objet rubrique reçoivent les publications effectuées dans cette rubrique et les chaînes de rubrique plus spécifiques que cette rubrique.

BLOCK
Les abonnements à une rubrique générique moins spécifique que la chaîne de rubrique dans cet objet rubrique ne reçoivent pas les publications effectuées dans cette rubrique ou les chaînes de rubrique plus spécifiques que cette rubrique.

La valeur de cet attribut est utilisée lorsque des abonnements sont définis. Si vous modifiez cet attribut, l'ensemble de rubriques couvert par les abonnements existants n'est pas affecté par la modification. Ce scénario s'applique également si la topologie est modifiée lorsque des objets rubrique sont créés ou supprimés. L'ensemble de rubriques correspondant aux abonnements créés à la suite de la modification de l'attribut WILDCARD est créé en utilisant la topologie modifiée. Si vous voulez forcer la réévaluation de l'ensemble de rubriques correspond pour les abonnements existants, vous devez redémarrer le gestionnaire de files d'attente.

Dans l'exemple, «Exemple : Créer le cluster de publication/abonnement Sport», à la page 89, vous pouvez suivre les étapes pour créer la structure de l'arborescence de rubriques affichée dans Figure 23, à la page 86.

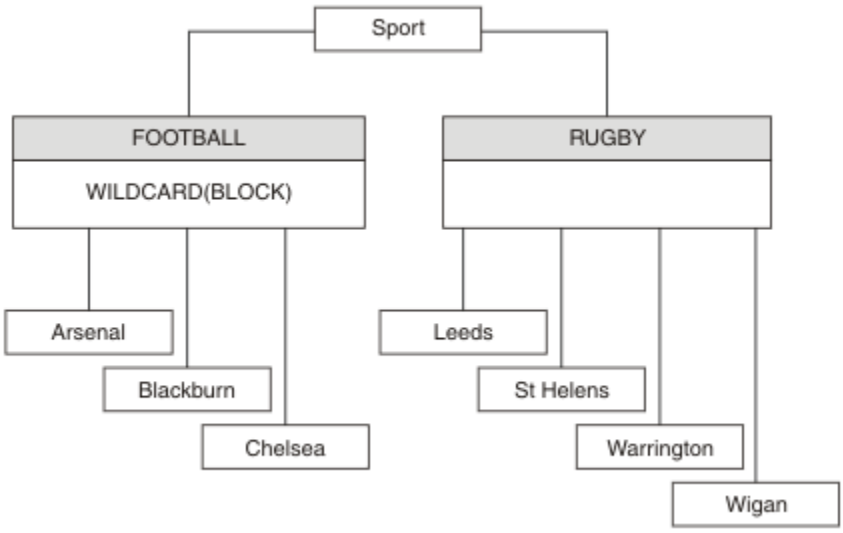


Figure 23. Arborescence de rubriques qui utilise la propriété WILDCARD, BLOCK

Un abonné utilisant la chaîne de rubrique générique `#` reçoit toutes les publications de la rubrique `Sport` et de la sous-arborescence `Sport/Rugby`. L'abonné ne reçoit aucune publication de la sous-arborescence `Sport/Football` car la valeur de propriété `WILDCARD` de la rubrique `Sport/Football` est `BLOCK`.

PASSTHRU correspond au paramètre par défaut. Vous pouvez définir la valeur PASSTHRU de la propriété `WILDCARD` sur les noeuds de l'arborescence `Sport`. Si les noeuds ne disposent pas de la valeur `BLOCK` de la propriété `WILDCARD`, la définition de PASSTHRU ne modifie pas le comportement constaté par les abonnés sur les noeuds de l'arborescence `Sports`.

Dans l'exemple, créez des abonnements pour observer comment le paramètre de caractère générique affecte les publications distribuées ; voir Figure 27, à la page 91. Exécutez la commande de publication dans Figure 30, à la page 92 pour créer quelques publications.

```
pub QMA
```

Figure 24. Publier dans QMA

Les résultats sont affichés dans Tableau 3, à la page 86. Notez comment la définition de la valeur `BLOCK` de la propriété `WILDCARD` empêche les abonnements avec caractères génériques de recevoir les publications des rubriques dans la portée du caractère générique.

Tableau 3. Publications reçues sur QMA

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Toutes les publications dans la sous-arborescence Football sont bloquées par `WILDCARD(BLOCK)` dans `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` sur `Sports/Football` empêche l'abonnement générique dans `Arsenal`

Tableau 3. Publications reçues sur QMA (suite)			
Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.

Remarque :

Supposons qu'un abonnement est doté d'un caractère générique qui fait correspondre un objet de rubrique à la valeur BLOCK de la propriété WILDCARD>. Si l'abonnement dispose également d'une chaîne de rubrique située à droite du caractère générique correspondant, il ne reçoit jamais de publication. Les publications qui ne sont pas bloquées sont celles dans les rubriques parent du caractère générique bloqué. Les publications dans les rubriques enfant de la rubrique dotée de la valeur de propriété BLOCK sont bloquées par le caractère générique. Par conséquent, les chaînes de rubrique d'abonnement incluant une rubrique située à droite du caractère générique ne reçoivent jamais de publications à des fins de correspondance.

La définition de la valeur de propriété WILDCARD sur BLOCK ne signifie pas que vous ne pouvez pas créer d'abonnement à l'aide d'une chaîne de rubrique qui comprend des caractères génériques. Un abonnement de ce type est normal. L'abonnement comprend une rubrique explicite qui correspond à la rubrique avec un objet de rubrique dont la propriété WILDCARD a pour valeur BLOCK. Il utilise les caractères génériques pour les rubriques parent ou enfant de la rubrique dont la propriété WILDCARD est définie sur BLOCK. Dans l'exemple figurant dans Figure 23, à la page 86, un abonnement tel que Sports/Football/# peut recevoir des publications.

Caractères génériques et rubriques de cluster

Les définitions de rubrique de cluster sont propagées à chaque gestionnaire de files d'attente d'un cluster. L'abonnement à une rubrique de cluster sur un gestionnaire de files d'attente d'un cluster entraîne la création d'abonnements proxy par le gestionnaire. Un abonnement proxy est créé sur chaque gestionnaire de files d'attente du cluster. Les abonnements utilisant les chaînes de rubrique contenant des caractères génériques et associés aux rubriques de cluster peuvent générer un comportement imprévisible. Le comportement est expliqué dans l'exemple qui suit.

Dans la configuration de cluster, par exemple, «Exemple : Créer le cluster de publication/abonnement Sport», à la page 89, QMB dispose du même ensemble d'abonnements que QMA, mais QMB n'a reçu aucune publication après que le diffuseur a publié dans QMA. Voir Figure 24, à la page 86. Même si les rubriques Sports/Football et Sports/Rugby sont des rubriques de cluster, les abonnements définis dans fullsubs.tst ne font pas référence à une rubrique de cluster. Aucun abonnement proxy n'est propagé depuis QMB dans QMA. Sans abonnement proxy, aucune publication sur QMA n'est réacheminée vers QMB.

Certains abonnements, tels Sports/#/Leeds, peuvent sembler référencer une rubrique de cluster, Sports/Rugby dans ce cas. L'abonnement Sports/#/Leeds est résolu en l'objet de rubrique SYSTEM.BASE.TOPIC.

La règle relative à la résolution de l'objet de rubrique référencé par un abonnement tel que Sports/#/Leeds est la suivante. Tronquez la chaîne de rubrique au premier caractère générique. Naviguez vers la gauche dans la chaîne de rubrique et recherchez la première rubrique comprenant un objet de rubrique d'administration associé. L'objet de rubrique peut spécifier un nom de cluster ou définir un objet de rubrique local. Dans l'exemple, Sports/#/Leeds, une fois tronquée, la chaîne de rubrique correspond à Sports, qui ne comprend pas d'objet de rubrique, et par conséquent Sports/#/Leeds hérite de SYSTEM.BASE.TOPIC, qui est un objet de rubrique local.

Pour visualiser comment l'abonnement aux rubriques en cluster peut modifier le fonctionnement de la propagation des caractères génériques, exécutez le script de commande upsubs.bat. Le script efface les files d'attente d'abonnement et ajoute les abonnements de rubrique de cluster dans fullsubs.tst. Exécutez puba.bat de nouveau pour créer un lot de publications ; voir Figure 24, à la page 86.

Tableau 4, à la page 88 affiche le résultat de l'ajout de deux nouveaux abonnements au même gestionnaire de files d'attente sur lequel les publications ont été publiées. Le résultat est comme prévu, les nouvelles publications reçoivent chacune une publication et le nombre de publications reçues par les autres abonnements est inchangé. Les résultats imprévus se produisent sur l'autre gestionnaire de files d'attente de cluster ; voir Tableau 5, à la page 88.

Tableau 4. Publications reçues sur QMA

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Toutes les publications dans la sous-arborescence Football sont bloquées par WILDCARD(BLOCK) dans Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) sur Sports/Football empêche l'abonnement générique dans Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal reçoit une publication car l'abonnement ne comporte pas de caractère générique.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds reçoit une publication dans tous les cas.

Tableau 5, à la page 88 affiche le résultat de l'ajout de deux nouveaux abonnements sur QMB et la publication dans QMA. Pour rappel, QMB n'a reçu aucune publication sans ces deux nouveaux abonnements. Comme prévu, les deux nouveaux abonnements reçoivent des publications car Sports/FootBall et Sports/Rugby sont tous deux des rubriques de cluster. QMB a transmis des abonnements proxy pour Sports/Football/Arsenal et Sports/Rugby/Leeds à QMA, qui a ensuite envoyé les publications à QMB.

Le résultat imprévu est que les deux abonnements Sports/# et Sports/#/Leeds qui ne recevaient pas de publications en reçoivent maintenant. Cette situation est due au fait que les publications Sports/Football/Arsenal et Sports/Rugby/Leeds transférées vers QMB pour les autres abonnements sont maintenant disponibles pour tout abonnement associé à QMB. Par conséquent, les abonnements aux rubriques locales Sports/# et Sports/#/Leeds reçoivent la publication Sports/Rugby/Leeds. Sports/#/Arsenal ne reçoit toujours pas de publication, car la propriété WILDCARD de Sports/Football est définie sur BLOCK.

Tableau 5. Publications reçues sur QMB

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Toutes les publications dans la sous-arborescence Football sont bloquées par WILDCARD(BLOCK) dans Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) sur Sports/Football empêche l'abonnement générique dans Arsenal

Tableau 5. Publications reçues sur QMB (suite)			
Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal reçoit une publication car l'abonnement ne comporte pas de caractère générique.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds reçoit une publication dans tous les cas.

Dans la plupart des applications, il n'est pas recommandé qu'un abonnement influence le comportement d'un autre abonnement. Un des avantages principaux de la propriété WILDCARD dotée de la valeur BLOCK est qu'elle permet aux abonnements de la même chaîne de rubrique contenant des caractères génériques d'avoir le même comportement. Les résultats de l'abonnement sont les mêmes, que l'abonnement soit sur le même gestionnaire de files d'attente que le diffuseur de publications ou sur un gestionnaire différent.

Caractères génériques et flux

Dans le cadre d'une nouvelle application écrite sur l'API de publication/abonnement, l'effet est qu'un abonnement à * ne reçoit aucune publication. Pour recevoir toutes les publications Sports, vous devez vous abonner à Sports/* ou Sports/# ; il en est de même pour les publications Business.

Le comportement d'une application de publication / abonnement en file d'attente existante ne change pas lorsque le courtier de publication / abonnement est migré vers une version ultérieure de IBM MQ. La propriété**StreamName** dans les commandes **Publish**, **Register Publisher** ou **Subscriber** est mappée sur le nom de la rubrique vers laquelle le flux a été migré.

Caractères génériques et points d'abonnement

Dans le cas d'une nouvelle application écrite sur l'API de publication/abonnement, l'effet de la migration est qu'un abonnement à * ne reçoit aucune publication. Pour recevoir toutes les publications Sports, vous devez vous abonner à Sports/* ou Sports/# ; il en est de même pour les publications Business.

Le comportement d'une application de publication / abonnement en file d'attente existante ne change pas lorsque le courtier de publication / abonnement est migré vers une version ultérieure de IBM MQ. La propriété **SubPoint** dans les commandes **Publish**, **Register Publisher** ou **Subscriber** est mappée sur le nom de la rubrique vers laquelle l'abonnement a été migré.

Exemple : Créer le cluster de publication/abonnement Sport

Les étapes suivantes créent un cluster, CL1, avec quatre gestionnaires de files d'attente : deux référentiels complets,CL1A et CL1B, et deux référentiels partiels, QMA et QMB. Les référentiels complets sont utilisés pour contenir uniquement les définitions de cluster. QMA est désigné comme hôte de rubrique de cluster. Les abonnements durables sont définis sur QMA et QMB.

Remarque : L'exemple est codé pour Windows. Vous devez coder à nouveau create qmgrs.bat et create pub.bat pour configurer et tester l'exemple sur les autres plateformes.

1. Créez les fichiers script.
   a. Créez topics.tst
   b. Créez wildsubs.tst
   c. Créez fullsubs.tst

d. Créez qmgrs.bat

e. Créez pub.bat

2. Exécutez qmgrs.bat pour créer la configuration.

```
qmgrs
```

Créez les rubriques dans Figure 23, à la page 86. Le script de la figure 5 crée les rubriques de cluster `Sports/Football` et `Sports/Rugby`.

Remarque : L'option REPLACE ne remplace pas les propriétés TOPICSTR d'une rubrique. TOPICSTR est une propriété dont la valeur varie dans l'exemple afin de tester les différentes arborescences de rubriques. Pour modifier les rubriques, supprimez d'abord la rubrique.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 25. Supprimer et créer des rubriques : topics.tst

Remarque : Supprimez les rubriques car REPLACE ne remplace pas les chaînes de rubrique.

Créez des abonnements avec des caractères génériques. Les caractères génériques faisant correspondre les rubriques aux objets de rubrique dans Figure 23, à la page 86. Créez une file d'attente pour chaque abonnement. Les files d'attente sont effacées et les abonnements supprimés lorsque le script est exécuté ou réexécuté.

Remarque : L'option REPLACE ne remplace pas les propriétés TOPICOBJ ou TOPICSTR d'un abonnement. TOPICOBJ ou TOPICSTR sont les propriétés dont la valeur varie dans l'exemple afin de tester les différents abonnements. Pour les modifier, supprimez d'abord l'abonnement.

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

Figure 26. Créer des abonnements génériques : wildsubs.tst

Créez des abonnements qui font référence aux objets de rubrique du cluster.

Remarque :

Le délimiteur, /, est automatiquement inséré entre la chaîne de rubrique référencée par TOPICOBJ et la chaîne de rubrique définie par TOPICSTR.

La définition DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) crée le même abonnement. TOPICOBJ permet de facilement référencer la chaîne de rubrique déjà définie. Lorsqu'il est créé, l'abonnement ne se rapporte plus à l'objet de rubrique.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

Figure 27. Supprimer et créer des abonnements : fullsubs.tst

Créez un cluster avec deux référentiels. Créez deux référentiels partiels pour la publication et l'abonnement. Réexécutez le script pour tout supprimer, puis recommencez. Le script crée également la hiérarchie de rubriques et les abonnements génériques initiaux.

Remarque :

Sur les autres plateformes, écrivez un script similaire ou tapez toutes les commandes. L'utilisation d'un script permet de tout supprimer rapidement et de recommencer avec une configuration identique.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figure 28. Créer des gestionnaires de files d'attente : qmgrs.bat

Mettez à jour la configuration en ajoutant des abonnements aux rubriques de cluster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figure 29. Mettre à jour les abonnements : upsubs.bat

Exécutez pub.bat, avec un gestionnaire de files d'attente comme paramètre pour publier les messages contenant la chaîne de rubrique de publication. Pub.bat utilise l'exemple de programme **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figure 30. Publier : pub.bat

Flux et rubriques

La publication/abonnement en file d'attente propose le concept de flux de publications qui n'existe pas dans le modèle de publication/abonnement intégré. Dans la publication/abonnement en file d'attente, les flux permettent de séparer les flux d'informations des différentes rubriques. Un flux est implémenté en tant que rubrique de niveau supérieur pouvant être mappée à un identificateur de rubrique différent sur le plan administratif.

Le flux par défaut SYSTEM.BROKER.DEFAULT.STREAM est configuré automatiquement pour tous les courtiers et les gestionnaires de files d'attente d'un réseau, et aucune configuration additionnelle n'est requise pour l'utiliser. Imaginez que le flux par défaut est un espace de sujet par défaut sans nom. Les rubriques publiées dans le flux par défaut sont immédiatement disponibles pour tous les gestionnaires de files d'attente connectés, avec publication / abonnement en file d'attente activé. Les flux nommés sont similaires aux espaces de sujet nommés séparés. Le flux nommé doit être défini sur chaque courtier sur lequel il est utilisé.

Si les diffuseurs de publications et les abonnés se trouvent sur des gestionnaires de files d'attente différents, une fois qu'ils sont connectés dans la même hiérarchie de courtiers, aucune configuration supplémentaire n'est requise pour que les publications et les abonnements circulent entre eux. La même interopérabilité fonctionne dans l'autre sens également.

Flux nommés

Un concepteur de solutions qui utilise le modèle de programmation de publication/abonnement en file d'attente peut décider de placer toutes les publications relatives au sport dans un flux nommé Sport. Pour que le flux soit disponible pour un gestionnaire de files d'attente qui s'exécute sur IBM MQ avec la fonction de publication / abonnement en file d'attente activée, le flux doit être ajouté manuellement.

Les applications de publication/abonnement en file d'attente qui s'abonnent à Soccer/Results dans le flux Sport fonctionnent sans modification. Les applications de publication/abonnement intégrées qui s'abonnent à la rubrique Sport à l'aide de MQSUB et fournissant la chaîne de rubrique Soccer/Results reçoivent également les mêmes publications.

La tâche d'ajout d'un flux est décrite dans la rubrique Ajout d'un flux. Il se peut que vous deviez ajouter des flux manuellement pour les deux raisons suivantes.

1. Vous poursuivez le développement de vos applications de publication/abonnement en file d'attente qui s'exécutent sur la version la plus récente du gestionnaire de files d'attente, plutôt que de migrer les applications vers l'interface MQI de publication/abonnement intégrée.
2. Le mappage par défaut des flux sur des rubriques entraîne une "collision" dans l'espace de sujet, et les publications dans un flux possèdent la même chaîne de rubrique que les publications d'un autre flux.

Droits

Par défaut, la racine de l'arborescence de rubriques contient plusieurs objets de rubrique : SYSTEM.BASE.TOPIC, SYSTEM.BROKER.DEFAULT.STREAM et SYSTEM.BROKER.DEFAULT.SUBPOINT. Les droits (pour la publication ou l'abonnement par exemple) sont déterminés par les droits dans SYSTEM.BASE.TOPIC ; tous les droits dans SYSTEM.BROKER.DEFAULT.STREAM ou SYSTEM.BROKER.DEFAULT.SUBPOINT sont ignorés. Si SYSTEM.BROKER.DEFAULT.STREAM ou SYSTEM.BROKER.DEFAULT.SUBPOINT est supprimé et recréé

avec une chaîne de rubrique non vide, les droits définis sur ces objets sont utilisés comme pour un objet de rubrique normal.

Mappage entre les flux et les rubriques

Un flux de publication / abonnement en file d'attente est imité dans IBM MQ en créant une file d'attente et en lui attribuant le même nom que le flux. La file d'attente est parfois appelée la file d'attente de flux car elle s'affiche comme telle pour les applications de publication/abonnement en file d'attente. Elle est identifiée par le moteur de publication/abonnement en l'ajoutant à la liste de noms spéciale appelée SYSTEM.QPUBSUB.QUEUE.NAMELIST. Vous pouvez ajouter le nombre de flux requis en ajoutant des files d'attente spéciales additionnelles à la liste de noms. Enfin, vous devez ajouter des rubriques ayant le même nom que les flux, et les mêmes chaînes de rubrique que le nom de flux pour pouvoir publier et vous abonner aux rubriques.

Dans des circonstances exceptionnelles toutefois, vous pouvez attribuer aux rubriques qui correspondent aux flux les chaînes de rubrique de votre choix lorsque vous définissez les rubriques. L'objectif d'une chaîne de rubrique est d'affecter à la rubrique un nom unique dans l'espace de sujet. En règle générale, le nom de flux remplit cette fonction parfaitement. Il arrive parfois qu'un nom de flux entre en conflit avec un nom de rubrique existant. Pour résoudre le problème, choisissez une autre chaîne de rubrique pour la rubrique associée au flux. Sélectionnez n'importe quelle chaîne de rubrique en vous assurant qu'elle est unique.

La chaîne de rubrique définie dans la définition de rubrique est ajoutée comme préfixe à la chaîne de rubrique fournie par les diffuseurs de publications et les abonnés à l'aide des appels MQI MQOPEN ou MQSUB. Les applications se référant aux rubriques à l'aide des objets de rubrique ne sont pas affectées par le choix d'une chaîne de rubrique en préfixe ; vous pouvez ainsi choisir une chaîne de rubrique qui rend les publications uniques dans l'espace de sujet.

Le remappage des différents flux sur les différentes rubriques dépend des préfixes utilisés pour rendre les chaînes de rubrique uniques, afin de distinguer un ensemble de rubriques d'un autre. Vous devez définir une convention d'attribution de nom de rubrique universelle qui est suivie à la lettre pour que le mappage fonctionne.

Dans IBM MQ, vous utilisez le mécanisme de préfixe pour remapper une chaîne de rubrique à un autre emplacement dans l'espace de sujet.

Remarque : Lorsque vous supprimez un flux, supprimez d'abord tous les abonnements du flux. Cette action est très importante si les abonnements proviennent d'autres courtiers de la hiérarchie.

Point d'abonnement et rubriques

Les points d'abonnement nommés sont émulés par les rubriques et les objets de rubrique.

Pour ajouter les points d'abonnement manuellement, voir Ajout d'un point d'abonnement.

Points d'abonnement dans IBM MQ

IBM MQ mappe les points d'abonnement sur différents espaces de sujet dans l'arborescence de rubriques d'IBM MQ. Les rubriques des messages de commande sans point d'abonnement sont associées à la racine de l'arborescence des rubriques IBM MQ et héritent des propriétés de SYSTEM.BASE.TOPIC.

Les messages de commande avec un point d'abonnement sont traités à l'aide de la liste des objets de rubrique dans SYSTEM.QPUBSUB.SUBPOINT.NAMELIST. Le nom de point d'abonnement dans le message de commande est comparé à la chaîne de rubrique de chaque objet de rubrique dans la liste. S'il existe une correspondance, le nom de point d'abonnement est ajouté en préfixe comme noeud de rubrique à la chaîne de rubrique. La rubrique hérite ses propriétés de l'objet de rubrique associé situé dans SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.

Les points d'abonnement sont utilisés pour créer un espace de sujet séparé pour chaque point d'abonnement. L'espace de sujet est associé à la racine d'une rubrique ayant le même nom que le point d'abonnement. Les rubriques dans chaque espace de sujet héritent de leurs propriétés de l'objet de rubrique ayant le même nom que le point d'abonnement.

Toutes les propriétés qui ne sont pas définies dans l'objet de rubrique correspondant sont héritées, de la manière normale, à partir de SYSTEM.BASE.TOPIC.

Les applications de publication/abonnement en file d'attente existantes qui utilisent les en-têtes de message MQRFH2 continuent de s'exécuter en définissant la propriété **SubPoint** dans les messages de commande Publish ou Register subscriber. Le point d'abonnement est associé à la chaîne de rubrique dans le message de commande et la rubrique obtenue est traitée de la même façon qu'une autre rubrique.

Les applications IBM MQ ne sont pas affectées par les points d'abonnement. Si une application utilise une rubrique qui hérite de l'un des objets de rubrique correspondants, elle interagit avec une application en file d'attente à l'aide du point d'abonnement correspondant.

Exemple

Une application de publication/abonnement WebSphere Message Broker existante (également appelée IBM Integration Bus) migrée vers IBM MQ a créé deux objets de rubrique, GBP et USD, avec les chaînes de rubrique correspondantes, 'GBP' et 'USD'.

Les éditeurs existants dans la rubrique NYSE/IBM/SPOT, migrés pour s'exécuter sous IBM MQ, qui utilisent le point d'abonnement USD créent des publications sur la rubrique USD/NYSE/IBM/SPOT. De même, les abonnés existants à NYSE/IBM/SPOT, à l'aide du point d'abonnement USD, créent des abonnements à USD/NYSE/IBM/SPOT.

Abonnez-vous au prix au comptant en dollars dans un programme de publication / abonnement IBM MQ en appelant MQSUB. Créez un abonnement à l'aide de l'objet de rubrique USD et de la chaîne de rubrique 'NYSE/IBM/SPOT', comme illustré dans le fragment de code 'C'.

```
strncpy(sd.ObjectName, "USD", MQ_TOPIC_NAME_LENGTH);
sd.ObjectString.VSPtr = "NYSE/IBM/SPOT";
sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
```

1. Définissez l'attribut CLUSTER des objets de rubrique USD et GBP sur l'hôte de rubrique de cluster.
2. Supprimez toutes les copies des objets de rubrique USD et GBP sur les autres gestionnaires de files d'attente du cluster.
3. Assurez-vous que USD et GBP sont définis dans SYSTEM.QPUBSUB.SUBPOINT.NAMELIST sur chaque gestionnaire de files d'attente du cluster.

Exemple de configuration de publication/abonnement à un seul gestionnaire de files d'attente

La Figure 31, à la page 95 illustre une configuration de publication/abonnement de base à un seul gestionnaire de files d'attente. L'exemple montre la configuration d'un service d'information dans lequel des informations sont disponibles à partir de diffuseurs de publications à propos de plusieurs rubriques.

- Diffuseur 1 publie des informations sur les résultats sportifs à l'aide d'une rubrique Sport
- Diffuseur 2 publie des informations sur le cours de la bourse à l'aide d'une rubrique Bourse
- Diffuseur 3 publie des informations sur des critiques de film à l'aide d'une rubrique Films et des programmes de télévision à l'aide d'une rubrique TV

Trois abonnés ont enregistré leur intérêt pour différentes rubriques. Le gestionnaire de files d'attente leur envoie donc les informations qui les intéressent :

- Abonné 1 reçoit les résultats sportifs et les cours de la bourse
- Abonné 2 reçoit les critiques de film
- Abonné 3 reçoit les résultats sportifs

Aucun des abonnés n'a enregistré son intérêt dans les programmes de télévision. Ces informations ne sont donc pas distribuées.

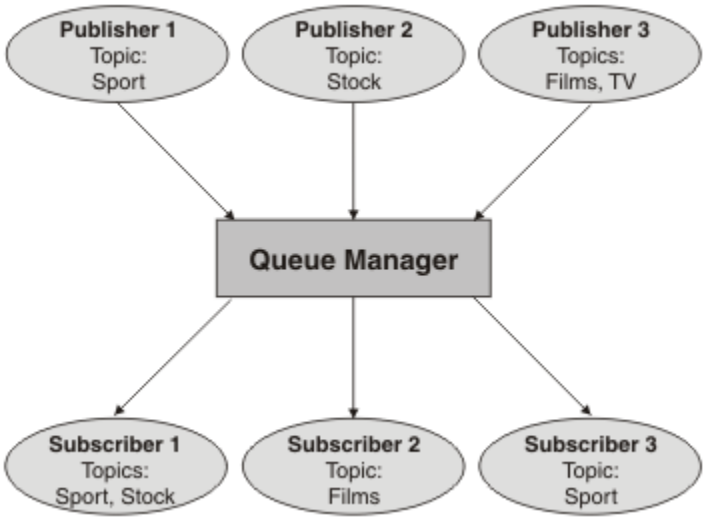


Figure 31. Exemple de publication/abonnement avec un seul gestionnaire de files d'attente

Réseaux de publication/abonnement répartis

Chaque gestionnaire de files d'attente fait concorder les messages publiés dans une rubrique avec les abonnements créée localement qui se sont abonnés à cette rubrique. Vous pouvez configurer un réseau de gestionnaires de files d'attente pour que les messages publiés par une application connectée à un gestionnaire de files d'attente soient distribués aux abonnements correspondants créés sur d'autres gestionnaires de files d'attente du réseau. Cela nécessite une configuration supplémentaire via des canaux simples entre les gestionnaires de files d'attente.

Une configuration de publication/abonnement répartie est un ensemble de gestionnaires de files d'attente connectés les uns avec les autres. Les gestionnaires de files d'attente peuvent tous se trouver sur le même système physique, ou être répartis sur plusieurs systèmes physiques. Lorsque vous connectez des gestionnaires de files d'attente ensemble, les abonnés peuvent s'abonner à un gestionnaire de files d'attente et recevoir des messages initialement publiés sur un autre gestionnaire de files d'attente. La figure suivante illustre cela en ajoutant un second gestionnaire de files d'attente à la configuration décrite dans «Exemple de configuration de publication/abonnement à un seul gestionnaire de files d'attente», à la page 94.

- Le gestionnaire de files d'attente 2 est utilisé par le diffuseur de publications 4 pour publier des informations relatives aux prévisions météo à l'aide d'une rubrique appelée Météo et des informations sur les conditions de trafic sur les routes à priorité à l'aide d'une rubrique appelée Trafic.
- L'abonné 4 utilise également ce gestionnaire de files d'attente et s'abonne aux informations sur les conditions de trafic à l'aide de la rubrique Trafic.
- L'abonné 3 s'abonne également aux informations relatives à la météo, bien qu'il utilise un gestionnaire de files d'attente différent de celui du diffuseur de publications. Cela est possible car les gestionnaires de files d'attente sont reliés entre eux.

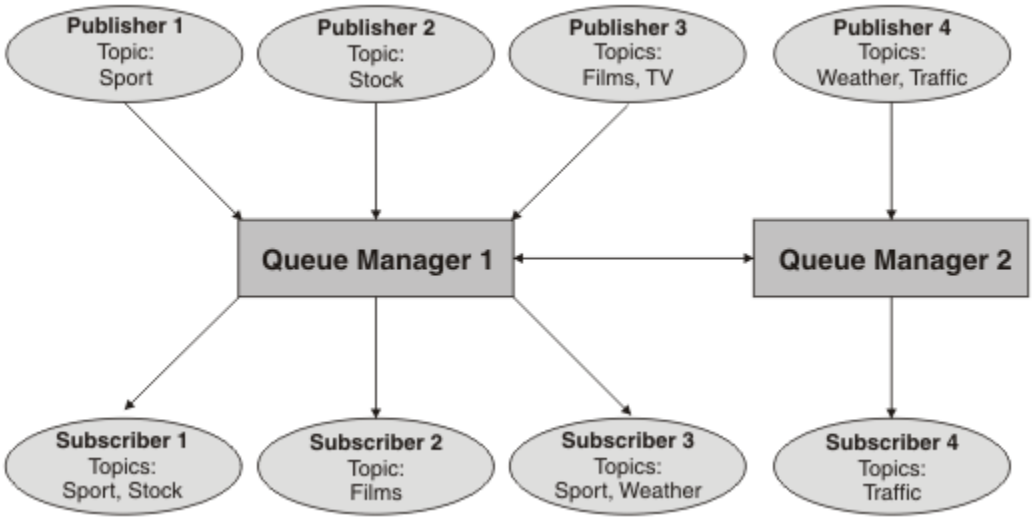


Figure 32. Exemple de publication/abonnement avec deux gestionnaires de files d'attente

Vous pouvez connecter manuellement les gestionnaires de files d'attente dans une hiérarchie parent et enfant, ou créer un cluster de publication/abonnement et laisser IBM MQ définir la plupart des informations de connexion automatiquement. Vous pouvez aussi combiner les deux topologies, par exemple en joignant plusieurs clusters dans une hiérarchie.

Présentation des clusters de publication/abonnement

Un cluster de publication/abonnement est un cluster standard comprenant un ou plusieurs objets de rubrique. Lorsque vous définissez un objet de rubrique d'administration sur n'importe quel gestionnaire de files d'attente d'un cluster, et que vous le mettez en cluster en indiquant un nom de cluster, les diffuseurs de publications et les abonnés à la rubrique peuvent se connecter à n'importe quel gestionnaire de files d'attente du cluster, et les messages publiés sont routés aux abonnés via les canaux du cluster entre les gestionnaires de files d'attente.

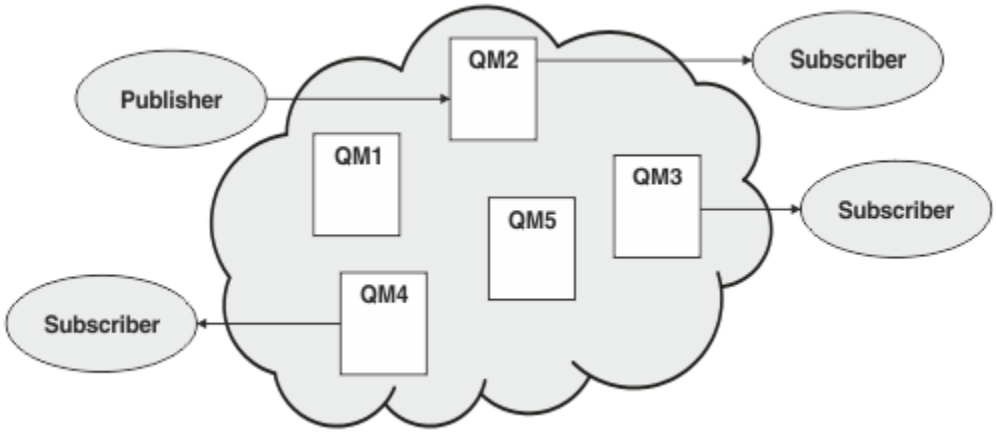


Figure 33. Cluster de publication/abonnement

Vous pouvez configurer le routage des messages de publication/abonnement dans un cluster de deux façons :

- routage direct
- routage via un hôte de rubrique

Lorsque vous configurez une rubrique en cluster à routage direct, les messages publiés sur le gestionnaire de files d'attente sont envoyés directement depuis ce gestionnaire de files d'attente à chaque abonnement sur n'importe quel gestionnaire de files d'attente du cluster. Le chemin le plus direct

pour les publications peut ainsi être obtenu, mais dans ce cas, tous les gestionnaires de files d'attente du cluster ont connaissance de tous les autres gestionnaires de files d'attente, pouvant ainsi établir des canaux de cluster entre eux.

Lorsque vous utilisez le routage via un hôte de rubrique, les messages publiés sur le gestionnaire de files d'attente sont envoyés à partir de ce dernier vers un gestionnaire de files d'attente qui héberge une définition d'objet de rubrique administré. Ce *gestionnaire de files d'attente hôte de rubrique* route le message à tous les abonnements des autres gestionnaires de files d'attente du cluster. Si les diffuseurs de publications ou les abonnés ne se trouvent pas sur les gestionnaires de files d'attente hôte de rubrique, la route des publications est plus longue. Cependant, l'avantage de cette méthode est que seuls les gestionnaires de files d'attente hôte de rubrique ont connaissance de tous les autres gestionnaires de files d'attente et peuvent ainsi établir des canaux de cluster avec eux.

Pour plus d'informations, voir «Clusters de publication/abonnement», à la page 98.

Présentation des hiérarchies de publication/abonnement

Une hiérarchie de publication/abonnement est un ensemble de gestionnaires de files d'attente connectés par des canaux dans une structure hiérarchique. Chaque gestionnaire de files d'attente identifie son gestionnaire de files d'attente *parent*n comme décrit dans Connexion d'un gestionnaire de files d'attente à une hiérarchie de publication/abonnement.

Les diffuseurs de publications et les abonnés à une rubrique peuvent se connecter à n'importe quel gestionnaire de files d'attente de la hiérarchie, et les messages sont transmis entre eux via la connectivité des gestionnaires de files d'attente hiérarchiques.

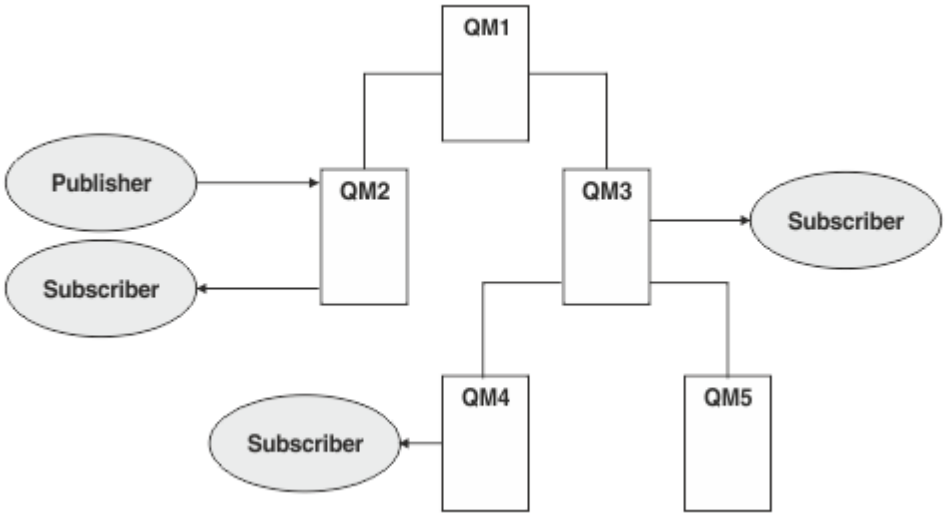


Figure 34. Hiérarchie de publication/abonnement

Dans la figure précédente, les publications distribuées aux abonnés sur QM3 et QM4 étaient routées depuis QM2 vers QM1, puis vers QM3 et finalement vers QM4.

Les hiérarchies permettent de contrôler directement les relations entre chaque gestionnaire de files d'attente de la hiérarchie. Le contrôle du routage des messages depuis les diffuseurs de publications vers les abonnés peut ainsi être plus affiné, surtout lorsque le routage entre les réseaux des gestionnaires de files d'attente s'effectue avec une connectivité restreinte. Accordez une attention particulière à la disponibilité et à la capacité de chaque gestionnaire de files d'attente par lequel un message est routé d'un diffuseur de publications vers les abonnés.

Pour plus d'informations, voir «Hiérarchies de publication/abonnement», à la page 101.

Distribution des publications entre les gestionnaires de files d'attente

Outre les choix liés au routage, deux approches permettent de distribuer les publications dans le réseau des gestionnaires de files d'attente :

- Envoyez les publications d'un gestionnaire de files d'attente vers les gestionnaires de files d'attente qui hébergent un abonnement pour cette publication.
- Envoyez chaque publication à tous les gestionnaires de files d'attente, et laissez-les faire correspondre la publication à leurs abonnements.

Dans le premier cas, les messages de publication sont alors uniquement envoyés lorsque cela est nécessaire, mais la connaissance des abonnements doit être partagée entre les gestionnaires de files d'attente. Dans le second cas, la connaissance des abonnements ne doit pas être partagée, mais il se peut que des messages de publication soient envoyés inutilement entre les gestionnaires de files d'attente.

Par défaut, IBM MQ utilise la première méthode et n'envoie les publications qu'aux gestionnaires de files d'attente qui y sont abonnés. La connaissance des abonnements est propagée entre les gestionnaires de files d'attente sous la forme d'*abonnements de proxy*. L'efficacité de la méthode à employer dans une topologie de publication/abonnement dépend de la distribution et de la durée de vie des abonnements, ainsi que de la fréquence des publications. Voir Performances des abonnements dans les réseaux de publication/abonnement.

Concepts associés

«Arborescence de rubriques», à la page 81
Chaque rubrique que vous définissez correspond à un élément ou à un noeud de l'arborescence de rubriques. Cette dernière peut être vide pour commencer ou contenir des rubriques définies précédemment à l'aide de commandes MQSC ou PCF. Vous pouvez définir une nouvelle rubrique à l'aide des commandes de création de rubrique ou en spécifiant la rubrique pour la première fois dans une publication ou un abonnement.

Scénarios de hiérarchie de publication/abonnement

Tâches associées

Conception des clusters de publication/abonnement

Clusters de publication/abonnement

Un cluster de publication/abonnement est un cluster standard de gestionnaires de files d'attente interconnectés, sur lequel les publications sont automatiquement déplacées des applications de publication vers les abonnements qui existent sur n'importe quel gestionnaire de files d'attente du cluster. Il existe deux options de routage des publications dans un même cluster de publication/abonnement : le *routage direct* et le *routage via un hôte de rubrique*. Le routage que vous choisissez dépend de la taille et des modèles d'activité attendus du cluster.

Un cluster utilisé pour une messagerie de type publication/abonnement n'est pas différent d'un cluster IBM MQ standard. De ce fait, les gestionnaires de files d'attente du cluster de publication/abonnement peuvent exister sur des ordinateurs séparés physiquement et les gestionnaires de chaque paire de gestionnaires de files d'attente sont automatiquement connectés ensemble par des canaux de cluster si nécessaire. Pour plus d'informations, voir Clusters.

Pour configurer un cluster standard de gestionnaires de files d'attente pour la messagerie de publication/abonnement, vous devez définie un ou plusieurs objets gérés sur un gestionnaire de files d'attente du cluster. Pour faire d'une rubrique une rubrique de cluster, vous devez configurer la propriété **CLUSTER** avec le nom du cluster. Toute rubrique alors utilisée par un diffuseur de publications ou par un abonné à ce point ou plus bas dans l'arborescence de rubriques est partagée par tous les gestionnaires de files d'attente du cluster, et les messages publiés sur une branche de l'arborescence de rubriques sont automatiquement routés vers les abonnements sur d'autres gestionnaires de files d'attente du cluster.

Seule une copie de chaque message est envoyée entre le gestionnaire de files d'attente du diffuseur de publications et chaque autre gestionnaire de files d'attente, quel que soit le nombre d'abonnés au message sur le gestionnaire de files d'attente cible. A son arrivée sur un gestionnaire de files d'attente avec un ou plusieurs abonnements, le message est dupliqué sur tous les abonnements.

Tout gestionnaire de files d'attente rejoignant le cluster prend automatiquement connaissance des rubriques en cluster, et les diffuseurs de publications ainsi que les abonnés de ce gestionnaire de files d'attente participent automatiquement au cluster.

L'activité de publication/abonnement non mise en cluster peut avoir lieu dans un cluster de publication/abonnement, en utilisant des chaînes de rubrique qui n'entrent dans un objet de rubrique de cluster.

Il existe deux options de routage des publications dans un même cluster de publication/abonnement : le *routage direct* et le *routage via un hôte de rubrique*. Pour choisir le routage du message à utiliser dans le cluster, définissez la propriété **CLROUTE** sur l'objet de rubrique administré à l'une des valeurs suivantes :

- **DIRECT**
- **TOPICHOST**

Par défaut, le routage de rubrique est **DIRECT**. Lorsque vous configurez une rubrique de cluster routée directement sur un gestionnaire de files d'attente, tous les gestionnaires de files d'attente du cluster détectent les autres gestionnaires de files d'attente de ce cluster. Lorsqu'il effectue des opérations de publication et d'abonnement, chaque gestionnaire de files d'attente peut se connecter directement à n'importe quel autre gestionnaire de files d'attente du cluster.

A partir de IBM MQ 8.0, vous pouvez à la place configurer le routage de rubrique en tant que **TOPICHOST**. Lorsque vous utilisez le routage via un hôte de rubrique, tous les gestionnaires de files d'attente du cluster détectent les gestionnaires de files d'attente de cluster qui hébergent la définition de rubrique routée (à savoir les gestionnaires de files d'attente sur lesquels vous avez défini l'objet de rubrique). Lorsque vous effectuez des opérations de publication et d'abonnement, les gestionnaires de files d'attente du cluster ne se connectent qu'à ces gestionnaires de files d'attente hôte de rubrique et ne se connectent pas directement les uns aux autres. Les gestionnaires de files d'attente hôte de rubrique sont responsables du routage des publications depuis les gestionnaires de files d'attente sur lesquels les publications sont publiées vers les gestionnaires avec les abonnements correspondants.

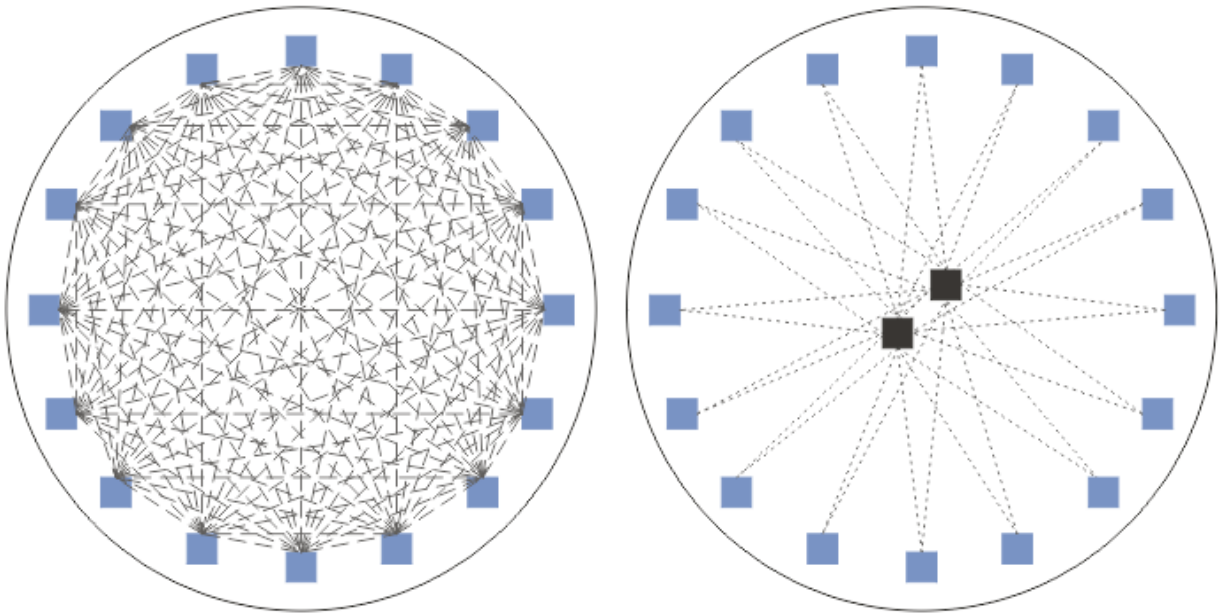


Figure 35. Routage direct et routage via un hôte de rubrique

Présentation du routage direct

Lorsqu'un objet de rubrique géré est configuré pour le routage direct, il suffit de le définir sur l'un des gestionnaires de files d'attente du cluster pour que tous les gestionnaires de files d'attente le sachent. Le choix du gestionnaire de files d'attente sur lequel la rubrique est définie n'affecte pas le comportement de la messagerie de publication/abonnement de la rubrique.

Chaque message est transmis directement depuis le gestionnaire de files d'attente du diffuseur de publications vers chaque abonnement sur les autres gestionnaires de files d'attente du cluster, et ne passe par aucun gestionnaire de files d'attente intermédiaire.

Par défaut, les messages sont envoyés uniquement aux autres gestionnaires de files d'attente du cluster qui hébergent un ou plusieurs abonnements.

- Cela implique que chaque gestionnaire de files d'attente informe directement tous les autres gestionnaires de files d'attente du cluster de la présence de toutes les rubriques dotées d'un ou de plusieurs abonnements qui lui sont associés. Tous les gestionnaires de files d'attente du cluster ont ainsi connaissance de toutes les rubriques associées à des abonnements, et de tous les gestionnaires de files d'attente qui hébergent un abonnement établissant un canal vers chaque autre gestionnaire de files d'attente. Cela ne dépend pas du fait que chaque gestionnaire de files d'attente ait in diffuseur de publications ou non.
- La connaissance de chaque rubrique abonnée individuelle sur tous les gestionnaires de files d'attente peut être supprimée par la modification d'un modèle d'envoi de toutes les publications à tous les gestionnaires de files d'attente du cluster, qu'ils soient dotés d'abonnements ou non. Cela réduit le trafic liée à la connaissance des abonnements, mais risque d'augmenter le trafic des publications et le nombre de canaux établis par chaque gestionnaire de files d'attente. Voir Performances des abonnements dans les réseaux de publication/abonnement.

Les flux de messages de publication/abonnement qui utilisent les rubriques en cluster à routage direct peuvent s'étendre sur plusieurs clusters grâce à l'ajout d'un gestionnaire de files d'attente de chaque cluster dans une hiérarchie de publication/abonnement. Voir Combinaison d'espaces de sujet de plusieurs clusters.

Pour plus d'informations sur le routage direct, voir Routage direct dans les clusters de publication/abonnement.

Présentation du routage via un hôte de rubrique

Lorsqu'un objet de rubrique administré est configuré pour le routage hôte de rubrique, les publications d'un gestionnaire de files d'attente dans le cluster sont routées via un gestionnaire de files d'attente, où l'objet de rubrique est configuré ("hôte de rubrique"), puis de ce point vers les gestionnaires de files d'attente où se trouvent les abonnements.

- Cela implique que chaque gestionnaire de files d'attente informe tous les hôtes de rubrique de la présence de chaque rubrique ayant un ou plusieurs abonnements qui lui sont associés. N'importe quel gestionnaire de files d'attente hébergeant un abonnement établit un canal vers chaque hôte de rubrique auquel l'abonnement est associé.
- Les gestionnaires de files d'attente qui n'hébergent pas de rubrique n'ont pas connaissance des autres gestionnaires de files d'attente n'hébergeant pas de rubrique dans le cluster à des fins de publication/abonnement et aucun canal n'est établi entre eux dans ce but.
- Si l'application de publication est connectée à un gestionnaire de files d'attente hébergeant la rubrique, les messages publiés sont routés directement aux gestionnaires de files d'attente dans lesquels des abonnements correspondants ont été créés, sans "tronçon" supplémentaire. De même, si les abonnements correspondants sont créés sur le seul gestionnaire de files d'attente hébergeant la rubrique, les messages publiés sur cette rubrique sont routés directement vers ce gestionnaire de files d'attente, sans tronçon supplémentaire.
- Les abonnements figurant sur le même gestionnaire de files d'attente que le diffuseur de publications sont satisfaits sans qu'il soit nécessaire d'effectuer un routage des publications vers les hôtes de l'objet de rubrique.

Tout comme pour les files d'attente de cluster, plusieurs gestionnaires de files d'attente peuvent configurer le même objet de rubrique géré. Cela offre une disponibilité plus élevée du routage des messages et une mise à l'échelle horizontale via l'équilibrage de la charge de travail. Dans le cas d'objets de rubrique routés via un hôte de rubrique, lorsque plusieurs gestionnaires de files d'attente configurent la même rubrique nommée pour la même branche de l'arborescence de rubriques, chaque hôte de

rubrique a connaissance des rubriques abonnées par chaque gestionnaire de files d'attente hébergeant un abonnement.

- Lorsqu'un message est publié, il est envoyé à l'un des gestionnaires de files d'attente hôte de rubrique pour être transmis à l'un des gestionnaires de files d'attente hébergeant l'abonnement. Le choix du gestionnaire de files d'attente hôte de rubrique suit les mêmes règles d'équilibrage de la charge de travail par défaut que pour les files d'attente point-à-point en cluster.
- Si un ou plusieurs gestionnaires de files d'attente hôte de rubrique ne peuvent pas être contacté par un gestionnaire de files d'attente de publication, les messages sont routés vers les gestionnaires de files d'attente hébergeant des rubriques restants disponibles.

Chaque publication vers une rubrique dans une branche routée de l'arborescence de rubriques est transmise à l'un des hôtes de rubrique, même s'il n'existe aucun abonnement à cette rubrique dans le cluster. Par défaut, les messages sont envoyés d'ici uniquement aux autres gestionnaires de files d'attente du cluster qui hébergent un ou plusieurs abonnements.

- Cela implique que chaque gestionnaire de files d'attente hôte de rubrique soit informé de la présence de toutes les chaînes de rubrique abonnées sur chaque gestionnaire de files d'attente du cluster.
- La connaissance de chaque rubrique abonnée individuelle peut être supprimée par la modification d'un modèle d'envoi de toutes les publications à un hôte de rubrique sur tous les gestionnaires de files d'attente du cluster, qu'ils soient ou non dotés d'abonnements. Cela réduit le trafic liée à la connaissance des abonnements, mais risque d'augmenter le trafic des publications et éventuellement le nombre de canaux établis avec chaque gestionnaire de files d'attente hébergeant des rubriques. Voir Performances des abonnements dans les réseaux de publication/abonnement.

Les flux de messages de publication/abonnement utilisant des rubriques en cluster routées via un hôte de rubrique **ne peuvent pas** s'étendre sur plusieurs clusters de publication/abonnement via l'utilisation d'une hiérarchie de publication/abonnement.

Pour plus d'informations sur le routage via un hôte de rubrique, voir Routage via un hôte de rubrique dans les clusters de publication/abonnement.

Hiérarchies de publication/abonnement

Pour générer une hiérarchie de publication/abonnement, vous devez lier les gestionnaires de files d'attente entre eux à l'aide de canaux, puis définir une relation parent-enfant entre les paires de gestionnaires de files d'attente. Un message est transmis d'un diffuseur de publications vers les abonnements via les relations directes d'une hiérarchie. Notez que le messages peut avoir à effectuer plusieurs "tronçons" pour arriver à destination.

Seule une copie du message est envoyée à une paire de gestionnaires de files d'attente, quel que soit le nombre d'abonnés du message sur le gestionnaire de files d'attente cible. A son arrivée sur un gestionnaire de files d'attente avec un ou plusieurs abonnements, le message est dupliqué sur tous les abonnements.

Par défaut, les messages sont envoyés uniquement aux autres gestionnaires de files d'attente de la hiérarchie qui se trouvent sur la route d'un abonnement sur un autre gestionnaire de files d'attente :

- Cela implique que chaque gestionnaire de files d'attente informe chaque relation directe de toutes les rubriques dotées d'un ou de plusieurs abonnements, soit sur ce gestionnaire de files d'attente, soit sur l'une de ses autres relations. Ainsi, tous les gestionnaires de files d'attente de la hiérarchie ont connaissance de toutes les rubriques dotées d'un abonnement.
- Ce comportement peut être modifié de telle sorte que les publications soient toujours envoyées à tous les gestionnaires de files d'attente de la hiérarchie, quel que soit le nombre d'abonnements existants. Il n'est alors plus nécessaire de propager les informations sur les abonnements dans la hiérarchie, mais cela peut augmenter le trafic des publications.

Lorsque vous créez un cluster, vous devez faire attention à ne pas créer de boucle entraînant un cycle infini des messages dans le réseau. Une telle boucle ne peut pas être créée dans une hiérarchie.

Chaque gestionnaire de files d'attente doit avoir un nom unique.

Les flux de message de publication/abonnement peuvent s'étendre sur plusieurs clusters de publication/abonnement. Pour ce faire, ajoutez un gestionnaire de files d'attente à partir de chaque cluster dans une hiérarchie de publication/abonnement.

Pour plus d'informations, voir Routage dans les hiérarchies de publication/abonnement.

Abonnements de proxy dans un réseau de publication/abonnement

Un abonnement de proxy est un abonnement souscrit par un gestionnaire de files d'attente pour des rubriques publiées sur un autre gestionnaire de files d'attente. Un abonnement de proxy est transmis entre les gestionnaires de files d'attente pour chaque chaîne de rubrique souscrite par un abonnement. Vous ne créez pas des abonnements de proxy de manière explicite. Le gestionnaire de files d'attente le fait automatiquement en votre nom.

Vous pouvez connecter ensemble des gestionnaire de files d'attente dans une hiérarchie de publication/abonnement ou dans un cluster de publication/abonnement. Les abonnements de proxy transitent entre les gestionnaires de files d'attente connectés. Avec les abonnements de proxy, les publications à une rubrique créées par un diffuseur de publications connecté à un gestionnaire de files d'attente sont reçues par les abonnés à cette rubrique connectés à d'autres gestionnaires de files d'attente. Voir «Réseaux de publication/abonnement répartis», à la page 95.

Dans des topologies de publication/abonnement avec plusieurs milliers d'abonnements à des chaînes de rubrique individuelles ou dans lesquels l'existence de ces abonnements peut rapidement changer, le temps système lié à la propagation d'abonnements de proxy doit être pris en compte. Outre l'agrégation automatique décrite dans le reste de cette rubrique, vous pouvez effectuer des modifications de configuration manuelles qui limitent davantage le flux des abonnements et des publications entre les gestionnaires de files d'attente connectés et qui réduisent le temps d'attente avant la propagation vers tous les gestionnaires de files d'attente connectés. Voir Performances des abonnements dans les réseaux de publication/abonnement.

Les abonnements de proxy ne contiennent aucun sélecteur utilisé par les abonnements locaux, et il est possible de simplifier les chaînes de rubrique d'abonnement qui contiennent des caractères génériques. Les publications peuvent alors correspondre à des abonnements de proxy et non aux abonnements réels, ce qui génère un flux de publication supplémentaire entre les gestionnaires de files d'attente. Le gestionnaire de files d'attente qui héberge les abonnements filtre ces différences pour que les publications supplémentaires ne soient pas renvoyées aux abonnements.

Agrégation d'abonnements de proxy

Les abonnements de proxy sont agrégés à l'aide d'un système d'élimination des doublons. Pour une chaîne de rubrique résolue particulière, un abonnement de proxy est envoyé pour le premier abonnement local ou le premier abonnement de proxy reçu. Les abonnements ultérieurs à la même chaîne de rubrique utilisent l'abonnement de proxy existant.

L'abonnement de proxy est annulé après que le dernier abonnement local ou le dernier abonnement de proxy reçu est annulé.

Agrégation de publications

Lorsqu'il existe plusieurs abonnements à la même chaîne de rubrique sur un gestionnaire de files d'attente, une seule copie de chaque publication correspondant à cette chaîne de rubrique est envoyée depuis d'autres gestionnaires de files d'attente dans une topologie de publication/abonnement. A l'arrivée du message, le gestionnaire de files d'attente local fournit une copie du message à chaque abonnement correspondant.

Il est possible que plusieurs abonnements de proxy correspondent à la chaîne de rubrique d'une seule publication lorsque les abonnements de proxy contiennent des caractères génériques. Si un message est publié sur un gestionnaire de files d'attente qui correspond à plusieurs abonnements de proxy créés par un seul gestionnaire de files d'attente connecté, une seule copie de la publication est transmise au gestionnaire de files d'attente éloignées pour satisfaire ces abonnements de proxy multiples.

Concepts associés

Détection de boucle dans un réseau de publication/abonnement distribué

Caractères génériques dans des abonnements de proxy

Les abonnements peuvent utiliser des caractères génériques dans les chaînes de rubrique pour les faire correspondre à plusieurs chaînes de rubrique des publications.

Il existe deux schémas de caractères génériques pouvant être utilisés par un abonnement : *basé sur les rubriques* et *basé sur les caractères*. Voir «Schémas de caractères génériques», à la page 75.

Dans IBM MQ , tous les abonnements de proxy pour les abonnements génériques sont convertis pour utiliser des caractères génériques basés sur des rubriques. Si un caractère générique basé sur les caractères est détecté, il est remplacé par un caractère *#*, placé près du caractère / le plus proche. Par exemple, `/aaa/bbb/c*d` est converti en `/aaa/bbb/#`. Avec la conversion, les gestionnaires de files d'attente éloignées envoient un peu plus de publications que celles auxquelles on s'est abonné explicitement. Les publications en trop sont éliminées par le gestionnaire de files d'attente local lorsque que celui-ci distribue les publications à ses abonnés locaux.

Contrôle de l'utilisation de caractères génériques avec la propriété WILDCARD

Utilisez la propriété WILDCARD du paramètre **Topic** de MQSC ou la propriété `WildcardOperation` équivalente de la commande PCF Topic pour contrôler la distribution des publications sur les applications abonnées qui utilisent les noms de chaîne de rubrique générique. La propriété WILDCARD peut avoir deux valeurs possibles :

WILDCARD
Comportement des abonnements génériques par rapport à cette rubrique.

PASSTHRU
Les abonnements à une rubrique générique moins spécifique que la chaîne de rubrique dans cet objet rubrique reçoivent les publications effectuées dans cette rubrique et les chaînes de rubrique plus spécifiques que cette rubrique.

BLOCK
Les abonnements à une rubrique générique moins spécifique que la chaîne de rubrique dans cet objet rubrique ne reçoivent pas les publications effectuées dans cette rubrique ou les chaînes de rubrique plus spécifiques que cette rubrique.

La valeur de cet attribut est utilisée lorsque des abonnements sont définis. Si vous modifiez cet attribut, l'ensemble de rubriques couvert par les abonnements existants n'est pas affecté par la modification. Ce scénario s'applique également si la topologie est modifiée lorsque des objets rubrique sont créés ou supprimés. L'ensemble de rubriques correspondant aux abonnements créés à la suite de la modification de l'attribut WILDCARD est créé en utilisant la topologie modifiée. Si vous voulez forcer la réévaluation de l'ensemble de rubriques correspond pour les abonnements existants, vous devez redémarrer le gestionnaire de files d'attente.

Dans l'exemple, «Exemple : Créer le cluster de publication/abonnement Sport», à la page 89, vous pouvez suivre les étapes pour créer la structure de l'arborescence de rubriques affichée dans Figure 23, à la page 86.

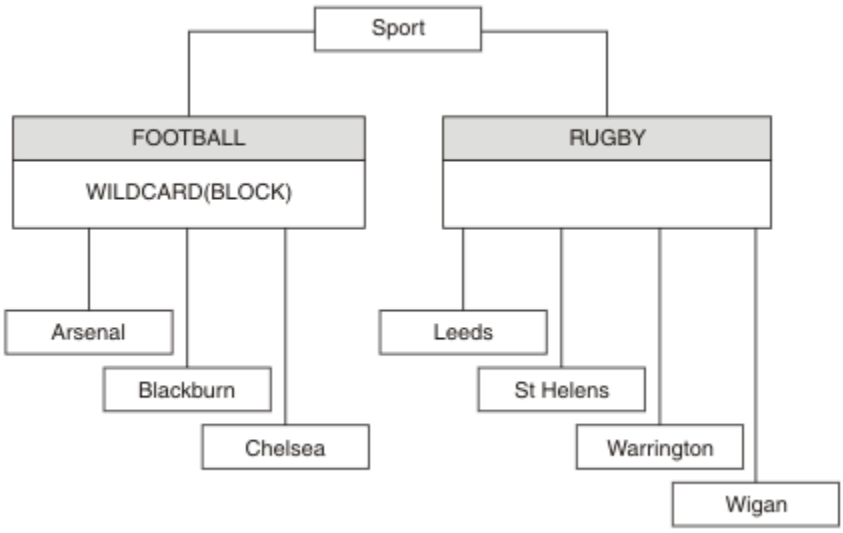


Figure 36. Arborescence de rubriques qui utilise la propriété WILDCARD, BLOCK

Un abonné utilisant la chaîne de rubrique générique `#` reçoit toutes les publications de la rubrique `Sport` et de la sous-arborescence `Sport/Rugby`. L'abonné ne reçoit aucune publication de la sous-arborescence `Sport/Football` car la valeur de propriété `WILDCARD` de la rubrique `Sport/Football` est `BLOCK`.

PASSTHRU correspond au paramètre par défaut. Vous pouvez définir la valeur PASSTHRU de la propriété `WILDCARD` sur les noeuds de l'arborescence `Sport`. Si les noeuds ne disposent pas de la valeur `BLOCK` de la propriété `WILDCARD`, la définition de PASSTHRU ne modifie pas le comportement constaté par les abonnés sur les noeuds de l'arborescence `Sports`.

Dans l'exemple, créez des abonnements pour observer comment le paramètre de caractère générique affecte les publications distribuées ; voir Figure 27, à la page 91. Exécutez la commande de publication dans Figure 30, à la page 92 pour créer quelques publications.

```
pub QMA
```

Figure 37. Publier dans QMA

Les résultats sont affichés dans Tableau 3, à la page 86. Notez comment la définition de la valeur `BLOCK` de la propriété `WILDCARD` empêche les abonnements avec caractères génériques de recevoir les publications des rubriques dans la portée du caractère générique.

Tableau 6. Publications reçues sur QMA

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**

SPORTS	`Sports/#`	`Sports`
		`Sports/Rugby`
		`Sports/Rugby/Leeds`
			Toutes les publications dans la sous-arborescence Football sont bloquées par `WILDCARD(BLOCK)` dans `Sports/Football`

SARSENAL	`Sports/#/Arsenal`	-	`WILDCARD(BLOCK)` sur `Sports/Football` empêche l'abonnement générique dans `Arsenal`

Tableau 6. Publications reçues sur QMA (suite)			
Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.

Remarque :

Supposons qu'un abonnement est doté d'un caractère générique qui fait correspondre un objet de rubrique à la valeur BLOCK de la propriété WILDCARD>. Si l'abonnement dispose également d'une chaîne de rubrique située à droite du caractère générique correspondant, il ne reçoit jamais de publication. Les publications qui ne sont pas bloquées sont celles dans les rubriques parent du caractère générique bloqué. Les publications dans les rubriques enfant de la rubrique dotée de la valeur de propriété BLOCK sont bloquées par le caractère générique. Par conséquent, les chaînes de rubrique d'abonnement incluant une rubrique située à droite du caractère générique ne reçoivent jamais de publications à des fins de correspondance.

La définition de la valeur de propriété WILDCARD sur BLOCK ne signifie pas que vous ne pouvez pas créer d'abonnement à l'aide d'une chaîne de rubrique qui comprend des caractères génériques. Un abonnement de ce type est normal. L'abonnement comprend une rubrique explicite qui correspond à la rubrique avec un objet de rubrique dont la propriété WILDCARD a pour valeur BLOCK. Il utilise les caractères génériques pour les rubriques parent ou enfant de la rubrique dont la propriété WILDCARD est définie sur BLOCK. Dans l'exemple figurant dans Figure 23, à la page 86, un abonnement tel que Sports/Football/# peut recevoir des publications.

Caractères génériques et rubriques de cluster

Les définitions de rubrique de cluster sont propagées à chaque gestionnaire de files d'attente d'un cluster. L'abonnement à une rubrique de cluster sur un gestionnaire de files d'attente d'un cluster entraîne la création d'abonnements proxy par le gestionnaire. Un abonnement proxy est créé sur chaque gestionnaire de files d'attente du cluster. Les abonnements utilisant les chaînes de rubrique contenant des caractères génériques et associés aux rubriques de cluster peuvent générer un comportement imprévisible. Le comportement est expliqué dans l'exemple qui suit.

Dans la configuration de cluster, par exemple, «Exemple : Créer le cluster de publication/abonnement Sport», à la page 89, QMB dispose du même ensemble d'abonnements que QMA, mais QMB n'a reçu aucune publication après que le diffuseur a publié dans QMA. Voir Figure 24, à la page 86. Même si les rubriques Sports/Football et Sports/Rugby sont des rubriques de cluster, les abonnements définis dans fullsubs.tst ne font pas référence à une rubrique de cluster. Aucun abonnement proxy n'est propagé depuis QMB dans QMA. Sans abonnement proxy, aucune publication sur QMA n'est réacheminée vers QMB.

Certains abonnements, tels Sports/#/Leeds, peuvent sembler référencer une rubrique de cluster, Sports/Rugby dans ce cas. L'abonnement Sports/#/Leeds est résolu en l'objet de rubrique SYSTEM.BASE.TOPIC.

La règle relative à la résolution de l'objet de rubrique référencé par un abonnement tel que Sports/#/Leeds est la suivante. Tronquez la chaîne de rubrique au premier caractère générique. Naviguez vers la gauche dans la chaîne de rubrique et recherchez la première rubrique comprenant un objet de rubrique d'administration associé. L'objet de rubrique peut spécifier un nom de cluster ou définir un objet de rubrique local. Dans l'exemple, Sports/#/Leeds, une fois tronquée, la chaîne de rubrique correspond à Sports, qui ne comprend pas d'objet de rubrique, et par conséquent Sports/#/Leeds hérite de SYSTEM.BASE.TOPIC, qui est un objet de rubrique local.

Pour visualiser comment l'abonnement aux rubriques en cluster peut modifier le fonctionnement de la propagation des caractères génériques, exécutez le script de commande upsubs.bat. Le script efface les files d'attente d'abonnement et ajoute les abonnements de rubrique de cluster dans fullsubs.tst. Exécutez puba.bat de nouveau pour créer un lot de publications ; voir Figure 24, à la page 86.

Tableau 4, à la page 88 affiche le résultat de l'ajout de deux nouveaux abonnements au même gestionnaire de files d'attente sur lequel les publications ont été publiées. Le résultat est comme prévu, les nouvelles publications reçoivent chacune une publication et le nombre de publications reçues par les autres abonnements est inchangé. Les résultats imprévus se produisent sur l'autre gestionnaire de files d'attente de cluster ; voir Tableau 5, à la page 88.

Tableau 7. Publications reçues sur QMA

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Toutes les publications dans la sous-arborescence Football sont bloquées par WILDCARD(BLOCK) dans Sports/Football
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) sur Sports/Football empêche l'abonnement générique dans Arsenal
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal reçoit une publication car l'abonnement ne comporte pas de caractère générique.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds reçoit une publication dans tous les cas.

Tableau 5, à la page 88 affiche le résultat de l'ajout de deux nouveaux abonnements sur QMB et la publication dans QMA. Pour rappel, QMB n'a reçu aucune publication sans ces deux nouveaux abonnements. Comme prévu, les deux nouveaux abonnements reçoivent des publications car Sports/FootBall et Sports/Rugby sont tous deux des rubriques de cluster. QMB a transmis des abonnements proxy pour Sports/Football/Arsenal et Sports/Rugby/Leeds à QMA, qui a ensuite envoyé les publications à QMB.

Le résultat imprévu est que les deux abonnements Sports/# et Sports/#/Leeds qui ne recevaient pas de publications en reçoivent maintenant. Cette situation est due au fait que les publications Sports/Football/Arsenal et Sports/Rugby/Leeds transférées vers QMB pour les autres abonnements sont maintenant disponibles pour tout abonnement associé à QMB. Par conséquent, les abonnements aux rubriques locales Sports/# et Sports/#/Leeds reçoivent la publication Sports/Rugby/Leeds. Sports/#/Arsenal ne reçoit toujours pas de publication, car la propriété WILDCARD de Sports/Football est définie sur BLOCK.

Tableau 8. Publications reçues sur QMB

Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Toutes les publications dans la sous-arborescence Football sont bloquées par WILDCARD(BLOCK) dans Sports/Football*
SARSENAL	Sports/#/Arsenal	-	WILDCARD(BLOCK) sur Sports/Football empêche l'abonnement générique dans Arsenal

Tableau 8. Publications reçues sur QMB (suite)			
Abonnement	**Chaîne de rubrique**	**Publications reçues**	**Remarques**
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*La valeur de propriété par défaut WILDCARD dans Sports/Rugby n'empêche pas l'abonnement générique dans Leeds.*
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal reçoit une publication car l'abonnement ne comporte pas de caractère générique.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds reçoit une publication dans tous les cas.

Dans la plupart des applications, il n'est pas recommandé qu'un abonnement influence le comportement d'un autre abonnement. Un des avantages principaux de la propriété WILDCARD dotée de la valeur BLOCK est qu'elle permet aux abonnements de la même chaîne de rubrique contenant des caractères génériques d'avoir le même comportement. Les résultats de l'abonnement sont les mêmes, que l'abonnement soit sur le même gestionnaire de files d'attente que le diffuseur de publications ou sur un gestionnaire différent.

Caractères génériques et flux

Dans le cadre d'une nouvelle application écrite sur l'API de publication/abonnement, l'effet est qu'un abonnement à * ne reçoit aucune publication. Pour recevoir toutes les publications Sports, vous devez vous abonner à Sports/* ou Sports/# ; il en est de même pour les publications Business.

Le comportement d'une application de publication / abonnement en file d'attente existante ne change pas lorsque le courtier de publication / abonnement est migré vers une version ultérieure de IBM MQ. La propriété**StreamName** dans les commandes **Publish**, **Register Publisher** ou **Subscriber** est mappée sur le nom de la rubrique vers laquelle le flux a été migré.

Caractères génériques et points d'abonnement

Dans le cas d'une nouvelle application écrite sur l'API de publication/abonnement, l'effet de la migration est qu'un abonnement à * ne reçoit aucune publication. Pour recevoir toutes les publications Sports, vous devez vous abonner à Sports/* ou Sports/# ; il en est de même pour les publications Business.

Le comportement d'une application de publication / abonnement en file d'attente existante ne change pas lorsque le courtier de publication / abonnement est migré vers une version ultérieure de IBM MQ. La propriété **SubPoint** dans les commandes **Publish**, **Register Publisher** ou **Subscriber** est mappée sur le nom de la rubrique vers laquelle l'abonnement a été migré.

Exemple : Créer le cluster de publication/abonnement Sport

Les étapes suivantes créent un cluster, CL1, avec quatre gestionnaires de files d'attente : deux référentiels complets,CL1A et CL1B, et deux référentiels partiels, QMA et QMB. Les référentiels complets sont utilisés pour contenir uniquement les définitions de cluster. QMA est désigné comme hôte de rubrique de cluster. Les abonnements durables sont définis sur QMA et QMB.

Remarque : L'exemple est codé pour Windows. Vous devez coder à nouveau create qmgrs.bat et create pub.bat pour configurer et tester l'exemple sur les autres plateformes.

1. Créez les fichiers script.
   a. Créez topics.tst
   b. Créez wildsubs.tst
   c. Créez fullsubs.tst

d. Créez qmgrs.bat

e. Créez pub.bat

2. Exécutez qmgrs.bat pour créer la configuration.

```
qmgrs
```

Créez les rubriques dans Figure 23, à la page 86. Le script de la figure 5 crée les rubriques de cluster `Sports/Football` et `Sports/Rugby`.

Remarque : L'option REPLACE ne remplace pas les propriétés TOPICSTR d'une rubrique. TOPICSTR est une propriété dont la valeur varie dans l'exemple afin de tester les différentes arborescences de rubriques. Pour modifier les rubriques, supprimez d'abord la rubrique.

```
DELETE TOPIC ('Sports')
DELETE TOPIC ('Football')
DELETE TOPIC ('Arsenal')
DELETE TOPIC ('Blackburn')
DELETE TOPIC ('Chelsea')
DELETE TOPIC ('Rugby')
DELETE TOPIC ('Leeds')
DELETE TOPIC ('Wigan')
DELETE TOPIC ('Warrington')
DELETE TOPIC ('St. Helens')

DEFINE TOPIC ('Sports')   TOPICSTR('Sports')
DEFINE TOPIC ('Football')  TOPICSTR('Sports/Football') CLUSTER(CL1) WILDCARD(BLOCK)
DEFINE TOPIC ('Arsenal')  TOPICSTR('Sports/Football/Arsenal')
DEFINE TOPIC ('Blackburn') TOPICSTR('Sports/Football/Blackburn')
DEFINE TOPIC ('Chelsea')  TOPICSTR('Sports/Football/Chelsea')
DEFINE TOPIC ('Rugby')   TOPICSTR('Sports/Rugby')  CLUSTER(CL1)
DEFINE TOPIC ('Leeds')   TOPICSTR('Sports/Rugby/Leeds')
DEFINE TOPIC ('Wigan')   TOPICSTR('Sports/Rugby/Wigan')
DEFINE TOPIC ('Warrington') TOPICSTR('Sports/Rugby/Warrington')
DEFINE TOPIC ('St. Helens') TOPICSTR('Sports/Rugby/St. Helens')
```

Figure 38. Supprimer et créer des rubriques : topics.tst

Remarque : Supprimez les rubriques car REPLACE ne remplace pas les chaînes de rubrique.

Créez des abonnements avec des caractères génériques. Les caractères génériques faisant correspondre les rubriques aux objets de rubrique dans Figure 23, à la page 86. Créez une file d'attente pour chaque abonnement. Les files d'attente sont effacées et les abonnements supprimés lorsque le script est exécuté ou réexécuté.

Remarque : L'option REPLACE ne remplace pas les propriétés TOPICOBJ ou TOPICSTR d'un abonnement. TOPICOBJ ou TOPICSTR sont les propriétés dont la valeur varie dans l'exemple afin de tester les différents abonnements. Pour les modifier, supprimez d'abord l'abonnement.

```
DEFINE QLOCAL(QSPORTS)  REPLACE
DEFINE QLOCAL(QSARSENAL) REPLACE
DEFINE QLOCAL(QSLEEDS)  REPLACE
CLEAR QLOCAL(QSPORTS)
CLEAR QLOCAL(QSARSENAL)
CLEAR QLOCAL(QSLEEDS)

DELETE SUB  (SPORTS)
DELETE SUB  (SARSENAL)
DELETE SUB  (SLEEDS)
DEFINE SUB  (SPORTS)  TOPICSTR('Sports/#')     DEST(QSPORTS)
DEFINE SUB  (SARSENAL) TOPICSTR('Sports/+/Arsenal') DEST(QSARSENAL)
DEFINE SUB  (SLEEDS)  TOPICSTR('Sports/+/Leeds')  DEST(QSLEEDS)
```

Figure 39. Créer des abonnements génériques : wildsubs.tst

Créez des abonnements qui font référence aux objets de rubrique du cluster.

Remarque :

Le délimiteur, /, est automatiquement inséré entre la chaîne de rubrique référencée par TOPICOBJ et la chaîne de rubrique définie par TOPICSTR.

La définition DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) crée le même abonnement. TOPICOBJ permet de facilement référencer la chaîne de rubrique déjà définie. Lorsqu'il est créé, l'abonnement ne se rapporte plus à l'objet de rubrique.

```
DEFINE QLOCAL(QFARSENAL) REPLACE
DEFINE QLOCAL(QRLEEDS)  REPLACE
CLEAR QLOCAL(QFARSENAL)
CLEAR QLOCAL(QRLEEDS)

DELETE SUB  (FARSENAL)
DELETE SUB  (RLEEDS)
DEFINE SUB  (FARSENAL) TOPICOBJ('Football') TOPICSTR('Arsenal') DEST(QFARSENAL)
DEFINE SUB  (RLEEDS)  TOPICOBJ('Rugby')  TOPICSTR('Leeds')  DEST(QRLEEDS)
```

Figure 40. Supprimer et créer des abonnements : fullsubs.tst

Créez un cluster avec deux référentiels. Créez deux référentiels partiels pour la publication et l'abonnement. Réexécutez le script pour tout supprimer, puis recommencez. Le script crée également la hiérarchie de rubriques et les abonnements génériques initiaux.

Remarque :

Sur les autres plateformes, écrivez un script similaire ou tapez toutes les commandes. L'utilisation d'un script permet de tout supprimer rapidement et de recommencer avec une configuration identique.

```
@echo off
set port.CL1B=1421
set port.CLIA=1420
for %%A in (CL1A CL1B QMA QMB) do call :createQM %%A
call :configureQM CL1A CL1B %port.CL1B% full
call :configureQM CL1B CL1A %port.CL1A% full
for %%A in (QMA QMB) do call :configureQM %%A CL1A %port.CL1A% partial
for %%A in (topics.tst wildsubs.tst) do runmqsc QMA < %%A
for %%A in (wildsubs.tst) do runmqsc QMB < %%A
goto:eof

:createQM
echo Configure Queue manager %1
endmqm -p %1
for %%B in (dlt crt str) do %%Bmqm %1
goto:eof

:configureQM
if %1==CL1A set p=1420
if %1==CL1B set p=1421
if %1==QMA set p=1422
if %1==QMB set p=1423
echo configure %1 on port %p% connected to repository %2 on port %3 as %4 repository
echo DEFINE LISTENER(LST%1) TRPTYPE(TCP) PORT(%p%) CONTROL(QMGR) REPLACE | runmqsc %1
echo START LISTENER(LST%1) | runmqsc %1
if full==%4 echo ALTER QMGR REPOS(CL1) DEADQ(SYSTEM.DEAD.LETTER.QUEUE) | runmqsc %1
echo DEFINE CHANNEL(TO.%2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('LOCALHOST(%3)') CLUSTER(CL1)
REPLACE | runmqsc %1
echo DEFINE CHANNEL(TO.%1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('LOCALHOST(%p%)')
CLUSTER(CL1) REPLACE | runmqsc %1
goto:eof
```

Figure 41. Créer des gestionnaires de files d'attente : qmgrs.bat

Mettez à jour la configuration en ajoutant des abonnements aux rubriques de cluster.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

Figure 42. Mettre à jour les abonnements : upsubs.bat

Exécutez pub.bat, avec un gestionnaire de files d'attente comme paramètre pour publier les messages contenant la chaîne de rubrique de publication. Pub.bat utilise l'exemple de programme **amqspub**.

```
@echo off
@rem Provide queue manager name as a parameter
set S=Sports
set S=6 Sports/Football Sports/Football/Arsenal
set S=6 Sports/Rugby Sports/Rugby/Leeds
for %%B in (6) do echo %%B | amqspub %%B %1
```

Figure 43. Publier : pub.bat

Concepts associés

Abonnements génériques et publications conservées

Portée de la publication

Lorsque vous configurez un cluster ou une hiérarchie de publication/abonnement, la portée d'une publication détermine également si les gestionnaires de files d'attente transmettent une publication aux gestionnaires de files d'attente éloignées. Utilisez l'attribut de rubrique **PUBSCOPE** pour administrer la portée des publications.

Si une publication n'est pas transmise à des gestionnaires de files d'attente éloignées, seuls les abonnés locaux la reçoivent.

Lorsque vous utilisez un cluster de publication/abonnement, la portée des publications est principalement contrôlée par la définition des objets de rubrique en cluster à certains points de l'arborescence de rubriques. La portée des publications doit être définie pour autoriser le flux des publications vers d'autres gestionnaires de files d'attente du cluster. Vous devez restreindre la portée des publications d'une rubrique en cluster uniquement lorsque vous avez besoin de contrôler avec une granularité fine des rubriques spécifiques sur certains gestionnaires de files d'attente.

Lorsque vous utilisez une hiérarchie de publication/abonnement, la portée des publications est généralement contrôlée par cet attribut, combiné avec l'attribut Portée des publications.

L'attribut **PUBSCOPE** permet de déterminer la portée des publications effectuées sur une rubrique spécifique. Vous pouvez définir cet attribut sur l'une des valeurs suivantes :

QMGR
: La publication est distribuée uniquement aux abonnés locaux. Ces publications sont appelées *publications locales*. Les publications locales sont pas transmises aux gestionnaires de files d'attente éloignées et sont ne donc pas reçues par les abonnés connectés à des gestionnaires de files d'attente éloignées.

TOUT
: La publication est distribuée aux abonnés locaux et aux abonnés connectés à des gestionnaires de files d'attente éloignées dans un cluster ou une hiérarchie de publication/abonnement. Ces publications sont appelées *publications globales*.

ASPARENT
: Utilisez le paramètre **PUBSCOPE** de la rubrique parent dans l'arborescence des rubriques.

Les diffuseurs de publications peuvent également indiquer si une publication est locale ou globale à l'aide de l'option de MQPMO_SCOPE_QMGR d'insertion de messages. Si cette option est utilisée, elle remplace le comportement qui a été défini à l'aide de l'attribut de rubrique **PUBSCOPE**.

Concepts associés

«Objets de rubrique d'administration», à la page 82
A l'aide d'un objet de rubrique d'administration, vous pouvez affecter des attributs spécifiques et autres que ceux par défaut aux rubriques.

Tâches associées

Configuration des réseaux de publication/abonnement réparti

Portée de l'abonnement

La portée d'un abonnement contrôle si un abonnement sur un gestionnaire de files d'attente reçoit des publications qui sont publiées sur un autre gestionnaire de files d'attente dans un cluster ou une hiérarchie de publication/abonnement, ou s'il reçoit uniquement les publications de diffuseurs de publications locaux.

La limitation de la portée d'un abonnement à un gestionnaire de files d'attente arrête la transmission des abonnements de proxy vers d'autres gestionnaires de files d'attente de la topologie de publication/abonnement. Cela réduit le trafic de messagerie de type publication/abonnement entre les gestionnaires de files d'attente.

Lorsque vous utilisez un cluster de publication/abonnement, la portée des abonnements est principalement contrôlée par la définition des objets de rubrique en cluster à certains points de l'arborescence de rubriques. La portée de l'abonnement doit être définie pour autoriser le flux des abonnements de proxy vers d'autres gestionnaires de files d'attente du cluster. Vous devez restreindre la portée de l'abonnement d'une rubrique en cluster uniquement lorsque vous avez besoin de contrôler avec une granularité fine des rubriques spécifiques sur certains gestionnaires de files d'attente.

Lorsque vous utilisez une hiérarchie de publication/abonnement, la portée des abonnements est généralement contrôlée par cet attribut, combiné avec l'attribut Portée de la publication.

L'attribut de rubrique **SUBSCOPE** est utilisé pour déterminer la portée des abonnements à une rubrique spécifique. Vous pouvez définir cet attribut sur l'une des valeurs suivantes :

QMGR
: Un abonnement reçoit uniquement les publications locales, et les abonnements proxy ne sont pas propagés aux gestionnaires de files d'attente éloignées.

TOUT
: Un abonnement de proxy est propagé vers les gestionnaires de files d'attente éloignées dans un cluster ou une hiérarchie de publication/abonnement, et l'abonné reçoit des publications locales et éloignées.

ASPARENT
: Utilisez le paramètre **SUBSCOPE** de la rubrique parent dans l'arborescence des rubriques.

Lorsque la portée de l'abonnement d'une rubrique est définie sur TOUT, directement ou via ASPARENT, vous pouvez limiter la portée des abonnements individuels à cette rubrique à QMGR en spécifiant MQSO_SCOPE_QMGR lors de la création de l'abonnement. Un abonnement à une rubrique dotée de la portée QMGR ne peut pas élargir la portée à ALL.

Concepts associés

«Objets de rubrique d'administration», à la page 82
A l'aide d'un objet de rubrique d'administration, vous pouvez affecter des attributs spécifiques et autres que ceux par défaut aux rubriques.

Tâches associées

Configuration des réseaux de publication/abonnement réparti

Espaces de sujet

Un espace de sujet est l'ensemble de rubriques auquel vous pouvez vous abonner et que vous pouvez publier. Un gestionnaire de files d'attente dans une topologie de publication/abonnement répartie a un espace de sujet qui inclut potentiellement des rubriques dotées d'abonnements et publiées sur les gestionnaires de files d'attente connectés de cette topologie.

Remarque : Pour avoir une présentation des rubriques d'un gestionnaire de files d'attente, comme les objets de rubrique d'administration, les chaînes de rubrique et les arborescences de rubriques, voir «Rubriques», à la page 73. Dans cet article, les autres références aux *rubriques* se rapportent aux *chaînes de rubrique* sauf indication contraire.

Les rubriques sont initialement créées de l'une des manières suivantes :

- de façon administrative, lorsque vous définissez un objet de rubrique ou un abonnement durable,

- dynamiquement, lorsqu'une application crée une publication ou un abonnement de manière dynamique à une nouvelle rubrique.

Les rubriques sont propagées à d'autres gestionnaires de files d'attente via les abonnements de proxy ou en créant des objets de rubrique de cluster administratifs. Des abonnements de proxy se traduisent par le réacheminement des publications depuis le gestionnaire de files d'attente auquel un diffuseur de publications est connecté vers les gestionnaires de files d'attente des abonnés.

Les abonnements de proxy sont propagés entre tous les gestionnaires de files d'attente qui sont connectés les uns aux autres par des relations parent-enfant dans une hiérarchie de gestionnaires de files d'attente. Par conséquent, vous pouvez vous abonner sur un gestionnaire à une rubrique définie sur un autre gestionnaire de files d'attente de la hiérarchie. Tant qu'il existe un chemin connecté entre les gestionnaires de files d'attente, la manière dont ceux-ci sont connectés n'a pas d'importance.

Les abonnements de proxy sont également propagés aux abonnements à des rubriques de cluster dans un cluster de publication/abonnement. Une rubrique de cluster est une rubrique qui est connectée à un objet de rubrique ayant l'attribut **CLUSTER** ou qui hérite de l'attribut de son parent. Les rubriques qui sont pas des rubriques de cluster sont appelées rubriques locales et ne sont pas répliquées vers le cluster. Aucun abonnement de proxy n'est propagé vers le cluster depuis des abonnements à des rubriques locales.

Pour résumer, des abonnements de proxy sont créés pour des abonnés dans deux situations.

1. Un gestionnaire de files d'attente est membre d'une hiérarchie, et un abonnement de proxy est réacheminé vers le parent et les enfants du gestionnaire de files d'attente.
2. Un gestionnaire de files d'attente est membre d'un cluster, et la chaîne de rubrique d'abonnement est résolue en une rubrique qui est associée à un objet de rubrique de cluster. Lorsqu'une rubrique est une rubrique de cluster *à routage direct*, les abonnements de proxy sont transmis à tous les membres du cluster. Lorsque la rubrique est une rubrique de cluster *routée via un hôte de rubrique*, les abonnements de proxy sont transmis uniquement aux gestionnaires de files d'attente du cluster qui ont défini l'objet de rubrique en cluster. Pour plus d'informations, voir «Clusters de publication/abonnement», à la page 98.

Si un gestionnaire de files d'attente est un membre d'un cluster et d'une hiérarchie, les abonnements de proxy sont propagés par les deux mécanismes sans distribution de publications en double à l'abonné.

Les espaces de sujet de trois topologies de publication/abonnement sont décrits dans la liste suivante :

- «Cas 1. Clusters de publication/abonnement», à la page 112.
- «Cas 2. Hiérarchies de publication/abonnement», à la page 113.

Dans des rubriques distinctes, les tâches de configuration suivantes expliquent comment combiner des espaces de sujet.

- Création d'un espace de sujet dans un cluster de publication/abonnement.
- Combinaison des espaces de sujet de plusieurs clusters.
- Combinaison et isolement des espaces de sujet dans plusieurs clusters.
- Publication et abonnement à des espaces de sujet dans plusieurs clusters.

Cas 1. Clusters de publication/abonnement

Dans cet exemple, supposons que le gestionnaire de files d'attente n'est pas connecté à une hiérarchie de publication/abonnement.

Si un gestionnaire de files d'attente est membre d'un cluster de publication/abonnement, son espace de sujet est constitué de rubriques locales et de rubriques de cluster. Les rubriques locales sont associées à des objets de rubrique sans l'attribut **CLUSTER**. Si un gestionnaire de files d'attente comporte des définitions d'objet de rubrique local, son espace de sujet est différent de celui d'un autre gestionnaire de files d'attente du cluster qui possède également ses propres objets de rubrique définis localement.

Dans un cluster de publication/abonnement, vous ne pouvez pas vous abonner à une rubrique définie sur un autre gestionnaire de files d'attente, sauf si la rubrique à laquelle vous vous abonnez est résolue en un objet de rubrique de cluster.

Lorsque les mêmes définitions nommées d'un objet de rubrique en cluster sont requises sur plusieurs gestionnaires de files d'attente, par exemple lors de l'utilisation du *routage via un hôte de rubrique*, il est essentiel que toutes les définitions correspondent là où cela est nécessaire. Pour plus d'informations, voir Création d'un espace de sujet dans un cluster de publication/abonnement.

Une définition locale d'un objet de rubrique, que cette définition soit pour une rubrique de cluster ou une rubrique locale, est prioritaire sur le même objet de rubrique défini ailleurs dans le cluster. La rubrique définie localement est utilisée, même si l'objet défini ailleurs est plus récent.

Il est important qu'un objet de rubrique de cluster soit associé à la même chaîne de rubrique partout dans le cluster. Vous ne pouvez pas modifier la chaîne de rubrique avec laquelle un objet de rubrique est associé. Pour associer le même objet de rubrique à une autre chaîne de rubrique, vous devez le supprimer et le recréer avec la nouvelle chaîne de rubrique. Si la rubrique est mise en cluster, cela revient à supprimer les copies de l'objet de rubrique stockées sur les autres membres du cluster, puis à créer des copies du nouvel objet de rubrique partout dans le cluster. Les copies de l'objet de rubrique font toutes référence à la même chaîne de rubrique.

Il est possible de créer par accident deux définitions du même objet de rubrique nommé sur deux gestionnaires de files d'attente différents, avec des chaînes de rubriques différentes. Cela peut être source de confusion car plusieurs définitions du même objet de rubrique avec des chaînes de rubrique différentes peuvent entraîner des résultats différents selon la manière dont la rubrique est référencée et à quel emplacement. Pour plus d'informations, voir Plusieurs définitions de rubrique de cluster du même nom.

Cas 2. Hiérarchies de publication/abonnement

Dans cet exemple, supposons que le gestionnaire de files d'attente n'est pas membre d'un cluster de publication/abonnement.

Dans IBM MQ, si un gestionnaire de files d'attente est membre d'une hiérarchie de publication / abonnement, son espace de sujet comprend toutes les rubriques définies en local et sur les gestionnaires de files d'attente connectés. L'espace de sujet de tous les gestionnaires de files d'attente d'une hiérarchie est le même. Il n'existe pas de séparation des rubriques en rubriques locales et en rubriques globales.

Définissez l'une des options **PUBSCOPE** et **SUBSCOPE** sur `QMGR` pour empêcher une publication sur une rubrique d'être transmises depuis un diffuseur de publications vers différents gestionnaires de files d'attente dans la hiérarchie.

Supposons que vous définissez un objet de rubrique `Alabama` avec la chaîne de rubrique `USA/Alabama` sur le gestionnaire de files d'attente `QMA`. Le résultat est le suivant :

1. L'espace de sujet dans `QMA` inclut désormais l'objet de rubrique `Alabama` et la chaîne de rubrique `USA/Alabama`.
2. Une application ou un administrateur peut créer un abonnement à l'adresse `QMA` à l'aide du nom d'objet de rubrique `Alabama`.
3. Une application peut créer un abonnement à n'importe quel sujet, y compris `USA/Alabama`, dans n'importe quel gestionnaire de files d'attente de la hiérarchie. Si `QMA` n'a pas été défini localement, la rubrique `USA/Alabama` est résolue par l'objet de rubrique `SYSTEM.BASE.TOPIC`.

Concepts associés

«Portée de la publication», à la page 110
Lorsque vous configurez un cluster ou une hiérarchie de publication/abonnement, la portée d'une publication détermine également si les gestionnaires de files d'attente transmettent une publication aux gestionnaires de files d'attente éloignées. Utilisez l'attribut de rubrique **PUBSCOPE** pour administrer la portée des publications.

«Portée de l'abonnement», à la page 111

La portée d'un abonnement contrôle si un abonnement sur un gestionnaire de files d'attente reçoit des publications qui sont publiées sur un autre gestionnaire de files d'attente dans un cluster ou une hiérarchie de publication/abonnement, ou s'il reçoit uniquement les publications de diffuseurs de publications locaux.

Tâches associées

Configuration des réseaux de publication/abonnement réparti

Multidiffusion IBM MQ

IBM MQ Multicast fournit une messagerie multidiffusion fiable, à haute distribution et à faible latence.

La multidiffusion est une forme de messagerie de type publication/abonnement efficace car elle peut contenir un nombre élevé d'abonnés sans que cela n'ait un impact négatif sur les performances. IBM MQ permet d'utiliser une messagerie multidiffusion fiable en utilisant des accusés de réception, des accusés réception négatifs et des numéros de séquence pour disposer d'une messagerie à haute distribution et faible latence.

La distribution fiable IBM MQ permet de distribuer les messages de manière quasiment simultanément pour qu'aucun destinataire n'ait un avantage. Comme la multidiffusion IBM MQ utilise le réseau pour distribuer les messages, aucun moteur de publication/abonnement n'est nécessaire pour distribuer les données. Une fois qu'une rubrique est mappée sur un adresse de groupe, le gestionnaire de files d'attente n'est pas requis, car les diffuseurs de publications et les abonnés fonctionnent en mode d'égal à égal. Ceci permet la réduction de la charge sur les serveurs de gestionnaire de files d'attente ; le gestionnaire ne représente plus une source d'incident potentielle.

Concepts multidiffusion initiaux

La multidiffusion IBM MQ peut facilement s'intégrer dans les systèmes et les applications existants à l'aide de l'objet Informations de communication (COMMINFO). Deux zones d'objet TOPIC activent la configuration rapide des objets TOPIC existants pour prendre en charge ou pour ignorer le trafic multidiffusion.

Objets requis pour la multidiffusion

Les informations suivantes sont une brève présentation des deux objets requis pour la multidiffusion IBM MQ :

Objet COMMINFO

L'objet COMMINFO contient les attributs associés à une transmission multidiffusion. Pour plus d'informations sur les paramètres de l'objet COMMINFO, voir DEFINE COMMINFO.

L'unique zone COMMINFO qui DOIT être défini est le nom de l'objet COMMINFO. Ce nom est ensuite utilisé pour identifier l'objet COMMINFO par rapport à une rubrique. La zone **GRPADDR** de l'objet COMMINFO doit être vérifié afin de s'assurer que la valeur est une adresse de groupe multidiffusion valide.

Objet TOPIC

Une rubrique est l'objet des informations publiées dans un message de publication/abonnement, et une rubrique se définit en créant un objet TOPIC. Pour plus d'informations sur les paramètres de l'objet TOPIC, voir DEFINE TOPIC.

Les rubriques existantes peuvent être utilisées avec la multidiffusion en modifiant les valeurs des paramètres de l'objet TOPIC suivants : **COMMINFO** et **MCAST**.

- **COMMINFO** Ce paramètre indique le nom de l'objet d'information de communication multidiffusion.
- **MCAST** Ce paramètre indique si la multidiffusion est autorisée à ce niveau dans l'arborescence des rubriques. Par défaut, **MCAST** est défini sur ASPARENT, ce qui signifie que l'attribut de multidiffusion de la rubrique est hérité du parent. Définir **MCAST** sur ENABLED autorise le trafic multidiffusion sur ce noeud.

Rubriques et réseaux de multidiffusion

Les informations suivantes sont une présentation de ce qui se passe avec les abonnements avec des définitions d'abonnement et de rubrique différentes. Ces exemples supposent tous que le paramètre **COMMINFO** de l'objet TOPIC est défini sur le nom d'un objet COMMINFO valide :

Activation de la rubrique définie sur la multidiffusion
Si le paramètre **MCAST** de la chaîne de rubrique est défini sur ENABLED, les abonnements des clients capables de multidiffusion sont autorisés et une abonnement de multidiffusion est réalisé sauf :

- S'il s'agit d'un abonnement durable d'un client capable de multidiffusion.
- S'il s'agit d'un abonnement non géré d'un client capable de multidiffusion.
- S'il s'agit d'un abonnement d'un client non capable de multidiffusion.

Dans ces cas, un abonnement sans multidiffusion est défini et lesabonnements sont rétrogradés vers la publication/l'abonnement normal.

Désactivation de la rubrique définie sur la multidiffusion
Si le paramètre **MCAST** de la chaîne de rubrique est défini sur DISABLED, un abonnement non multidiffusion est toujours réalisé et lesabonnements sont rétrogradés vers la publication/ l'abonnement normal.

Rubrique défini sur la multidiffusion uniquement
Si le paramètre **MCAST** de la chaîne de rubrique est défini sur ONLY, les abonnements des clients capables de multidiffusion sont autorisés et un abonnement multidiffusion est réalisé sauf :

- Il s'agit d'un abonnement durable : les abonnements durables sont rejetés avec le code anomalie 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
- Il s'agit d'un abonnement non géré : les abonnements non gérés sont rejetés avec le code anomalie 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
- Il s'agit d'un abonnement d'un client ne pouvant pas prendre en charge la multidiffusion : ces abonnements sont rejetés avec le code anomalie 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
- Il s'agit d'un abonnement d'une application locale : ces abonnements sont rejetés avec le code anomalie 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY

Windows Linux AIX MQ Telemetry - Généralités

MQ Telemetry comprend un service de télémétrie (MQXR) qui fait partie d'un gestionnaire de files d'attente ainsi que de clients de télémétrie que vous pouvez écrire vous-même ou télécharger gratuitement, ainsi que des interfaces d'administration de l'explorateur ou de ligne de commande. La télémétrie représente l'administration et la collecte de données à partir d'un large éventail de périphériques distants. MQ Telemetry permet d'intégrer la collection de données et de contrôler les périphériques dotés d'applications Web.

MQ Telemetry est un composant de IBM MQ. La mise à niveau de ces versions consiste essentiellement à installer la dernière version d'IBM MQ.

Les exemples d'application continuent d'être disponibles gratuitement à partir d' Eclipse Paho et MQTT.org. Voir IBM MQ Telemetry Transport sample programs.

MQ Telemetry étant un composant de IBM MQ, MQ Telemetry peut être installé avec le produit principal ou après l'installation du produit principal. Pour des informations sur la migration, voir Migration de MQ Telemetry sous Windows et Migration de MQ Telemetry sous Linux.

Les composants suivants sont inclus dans MQ Telemetry :

Canaux de télémétrie
Utilisez les canaux de télémétrie pour gérer la connexion des clients MQTT à IBM MQ . Les canaux de télémétrie utilisent de nouveaux objets IBM MQ , tels que SYSTEM.MQTT.TRANSMIT.QUEUE, pour interagir avec IBM MQ.

Service de télémétrie (MQXR)
Les clients MQTT utilisent le service de télémétrie SYSTEM.MQXR.SERVICE pour se connecter aux canaux de télémétrie.

Prise en charge d'IBM MQ Explorer pour MQ Telemetry
MQ Telemetry peut être administré à l'aide de IBM MQ Explorer.

Documentation
La documentation MQ Telemetry est incluse dans la documentation standard du produit IBM MQ . La documentation relative au kit de développement des clients Java et C est fournie dans la documentation du produit, sous la forme Javadoc et dans le format HTML.

Concepts de télémétrie

Vous collectez des informations à partir de l'environnement qui vous entoure et décidez les actions à entreprendre. En tant que consommateur, vous vérifiez ce que vous avez en stock avant d'acheter votre nourriture. Vous voulez savoir la durée de votre voyage si vous partez maintenant, avant d'effectuer votre réservation. Vous vérifiez les symptômes avant de vous rendre chez le médecin. Vous vérifiez l'horaire du bus et pouvez ainsi décider d'attendre ou non. Les informations qui vous aident à prendre ces décisions sont fournies par des compteurs et des appareils, un mot écrit sur une feuille de papier ou sur un écran et par vous-même. Où que vous soyez, et quand vous le voulez, vous collectez des informations, les rassemblez, les analysez et les utilisez.

Si les sources d'information sont trop éparses ou inaccessibles, il est difficile et coûteux de retrouver les informations les plus appropriées. Si vous voulez effectuer beaucoup de modifications ou s'il est difficile d'effectuer ces modifications, elles ne seront pas effectuées ou le seront au moment le moins opportun.

Que penseriez-vous si le coût de la collecte d'informations et le contrôle d'unités largement disséminées était réduit de manière significative par la connexion de ces appareils à Internet via la technologie numérique ? Ces informations pourraient alors être analysées à l'aide des ressources d'Internet et des entreprises. Vous auriez alors de nombreuses opportunités de prendre des décisions et d'agir en connaissance de cause.

Les tendances technologiques, les pressions environnementales et économiques permettent d'effectuer ces changements :

1. Le coût de la connexion et du contrôle des détecteurs et des actionneurs diminue en raison de la standardisation et de la connexion à des processeurs numériques à faible coût.
2. Internet et les technologies liées à Internet sont de plus en plus utilisés pour la connexion des appareils. Dans certains pays, le nombre de connexions aux applications Internet réalisées via les téléphones mobiles dépasse celui effectué avec des ordinateurs portables. D'autres types d'unités suivront la probablement la même tendance.
3. Internet et les technologies liées à Internet facilitent l'accession aux données pour une application. Un accès plus facile aux données permet d'utiliser l'analyse de données pour transformer les données des détecteurs en informations utiles pour plusieurs solutions.
4. L'utilisation Intelligente des ressources est souvent un moyen moins cher et plus rapide de réduire les coûts et les émissions de carbone. Les alternatives sont les suivantes : trouver de nouvelles ressources ou développer de nouvelles technologies pour utiliser les ressources existantes peut être une solution à long terme. A court terme, le développement de nouvelles technologies ou la recherche de nouvelles ressources est souvent plus risqué, plus lent et plus coûteux que l'amélioration des solutions existantes.

Exemple

Un exemple illustre comment ces tendances créent de nouvelles opportunités pour interagir intelligemment avec l'environnement.

La convention internationale SOLAS (Safety of Life at Sea) nécessite le déploiement du Système d'identification automatique (SIA) sur de nombreux navires. Ce système est obligatoire sur les navires marchands de plus de 300 tonnes et les navires de passagers. Le système SIA est principalement un

système permettant d'éviter les collisions lors du cabotage. Il est utilisé par les autorités maritimes pour surveiller et contrôler les eaux côtières.

Des passionnés dans le monde entier déploient des stations de suivi SIA et envoient les informations relatives à la navigation de cabotage sur Internet. D'autres passionnés écrivent des applications combinant des informations provenant du système SIA avec d'autres informations provenant d'Internet. Les résultats sont mis sur des sites Web et publiés à l'aide de Twitter et de SMS.

Dans une application, les informations provenant de stations SIA près de Southampton sont combinées avec celles des propriétaires de navires et des informations géographiques. L'application fournit des informations en direct sur les arrivées et les départs de bateaux sur Twitter. Les voyageurs réguliers utilisant les bateaux entre Southampton et l'île de Wight s'abonnent à la distribution de nouvelles en utilisant Twitter ou les SMS. Si le fil indique que le bateau est en retard, les voyageurs peuvent retarder leur départ et prendre le bateau lorsqu'il accoste plus tard que son heure d'arrivée prévue.

Pour plus d'exemples, voir «Scénarios d'utilisation de la télémétrie», à la page 118.

Tâches associées

Installation de MQ Telemetry

Administration de MQ Telemetry

Migration d'MQ Telemetry sous Windows

Migration d'MQ Telemetry sous Linux

Développement d'applications pour MQ Telemetry

Traitement des incidents liés à MQ Telemetry

Référence associée

Référence MQ Telemetry

Windows Linux AIX Présentation de MQ Telemetry

Les personnes, les commerces et les gouvernements veulent de plus en plus utiliser MQ Telemetry pour interagir plus intelligemment avec l'environnement dans lequel nous vivons et nous travaillons. MQ Telemetry permet de connecter toutes sortes d'unités à Internet et aux entreprises, et de réduire les coûts de création d'applications pour les unités intelligentes.

Qu'est-ce que MQ Telemetry ?

- Il s'agit d'une fonction d'IBM MQ qui étend le réseau principal de la messagerie universelle fournie par IBM MQ à toute une gamme de détecteurs distants, de mécanismes d'accès et d'appareils de télémétrie. MQ Telemetry étend IBM MQ de sorte qu'il puisse connecter des applications intelligentes d'entreprises, des services et des décisionnaires aux réseaux d'appareils instrumentés.
- Les principaux composants de MQ Telemetry sont les suivants :

  Le service MQ Telemetry (MQXR)
  Ce service s'exécute dans le serveur IBM MQ et utilise le protocole IBM MQ Telemetry Transport (MQTT) pour communiquer avec les périphériques de télémétrie.

  Les applications MQTT que vous écrivez
  Ces applications contrôlent les informations transmises entre les dispositifs de télémétrie et le gestionnaire de files d'attente IBM MQ, ainsi que les actions effectuées en réponse à ces informations. Pour faciliter la création de ces applications, vous utilisez des bibliothèques client MQTT.

Intérêt

- MQTT est un transport de messagerie qui permet de créer des implémentations MQTT pour un large éventail de périphériques.
- Les clients MQTT peuvent s'exécuter sur des périphériques plus petits ayant des ressources limitées.
- MQTT fonctionne efficacement sur les réseaux à faible bande passante, où le coût d'envoi des données est élevé ou peut être fragile.

- La distribution des messages est assurée et découplée de l'application.
- Les programmeurs d'application n'ont pas besoin d'avoir des connaissances en programmation des communications.
- Des messages peuvent être échangés avec d'autres applications de messagerie. Ces applications peuvent être une autre application de télémétrie, une interface MQI, JMS ou une application de messagerie d'entreprise.

Utilisation

- Des exemples de scripts sont fournis avec un exemple d'application client IBM MQ Telemetry Transport v3 (`mqttv3app.jar`). Voir Programmes exemples IBM MQ Telemetry Transport.
- Utilisez IBM MQ Explorer et les outils associés pour administrer la fonction de télémétrie d'IBM MQ.
- Utilisez les bibliothèques client pour créer des applications MQTT qui connectent un gestionnaire de files d'attente et qui utilisent la messagerie de publication/abonnement.
- Distribuez l'application et la bibliothèque client sur le dispositif sur lequel l'application doit être exécutée.

Fonctionnement

- MQTT est un protocole de publication/abonnement. Une application client MQTT peut publier des messages vers un serveur MQTT ou s'abonner aux messages envoyés par les applications qui se connectent à un serveur MQTT.
- Les applications client MQTT utilisent les bibliothèques client qui implémentent le transport des messages MQTT.
- Une application client MQTT des base fonctionne comme un client MQ standard, mais elle peut s'exécuter sur un très large éventail de plateformes et de réseaux.
- Le service MQ Telemetry (MQXR) transforme un gestionnaire de files d'attente IBM MQ en serveur MQTT.
- Lorsqu'un gestionnaire de files d'attente IBM MQ fait office de serveur MQTT, les autres applications qui se connectent au gestionnaire de files d'attente peuvent s'abonner aux messages du client MQTT et les recevoir.
- Le gestionnaire de files d'attente agit comme un routeur qui distribue des messages des applications de publications aux applications d'abonnement.
- Les messages peuvent être distribués entre différents types d'applications client. Par exemple, entre les clients Telemetry et les clients JMS.

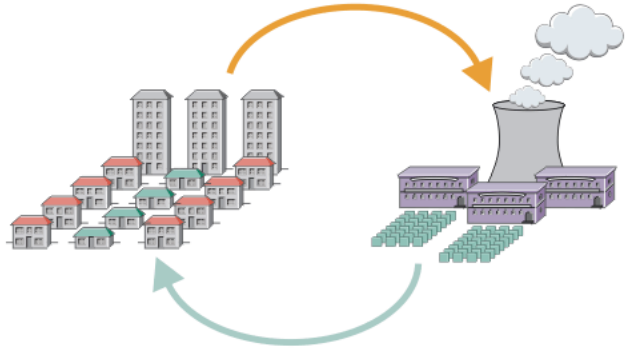
Remarque : MQ Telemetry remplace les noeuds SCADA qui ont été retirés de la version 7 de WebSphere Message Broker (également appelée IBM Integration Bus) et s'exécute sous Windows, Linux et AIX.

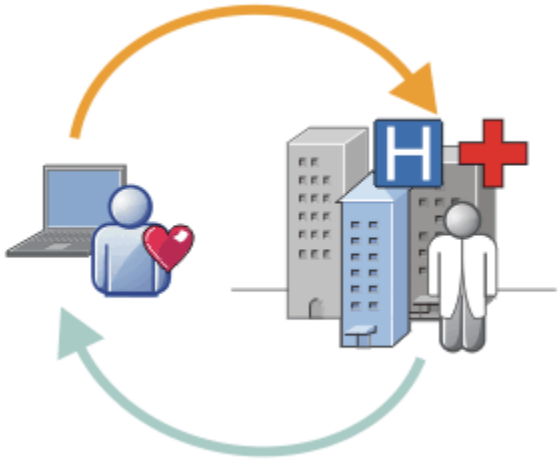
Windows Linux AIX Scénarios d'utilisation de la télémétrie

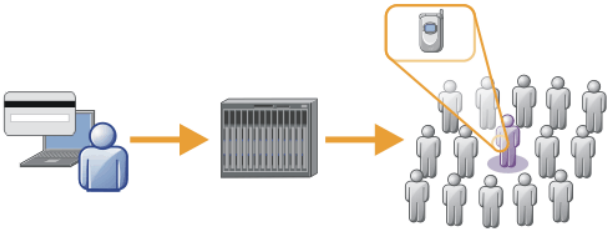
La télémétrie est la détection automatisée, la mesure des données et le contrôle des appareils à distance. L'accent est mis sur la transmission des données à partir des unités vers un point de contrôle central. La télémétrie comprend également l'envoi d'informations de configuration et de contrôle aux unités.

MQ Telemetry permet de connecter de petits périphériques utilisant le MQTT protocol et de connecter ces périphériques à d'autres applications à l'aide d'IBM MQ. MQ Telemetry comble le fossé qui existait entre les unités et Internet facilitant la création de "solutions intelligentes". Ces solutions permettent d'utiliser l'abondance d'informations disponibles sur Internet et dans les applications d'entreprise pour les mettre à disposition d'applications qui surveillent et contrôlent les unités.

Les diagrammes suivants illustrent quelques utilisations typiques de MQ Telemetry :

Télémétrie : Electricité intelligente	
	• Messages MQTT contenant les données relatives à la consommation d'énergie envoyées au fournisseur de services. • MQ Telemetry envoie des commandes de contrôle CONTROL COMMANDS basées sur l'analyse des données relatives à la consommation d'énergie. • Pour plus d'informations, voir le cas d'utilisation suivant : «Scénarios d'utilisation de la télémétrie : Surveillance et contrôle de l'énergie à domicile», à la page 121

Télémétrie : Services de santé intelligent	
• MQ Telemetry envoie les données de santé à votre hôpital et à votre médecin. • Des alertes de message MQTT peuvent être envoyées en fonction de l'analyse des données de santé. • Pour plus d'informations, voir le cas d'utilisation suivant : «Scénarios d'utilisation de la télémétrie : Surveillance de patient à domicile», à la page 119	

Télémétrie : One in a Crowd	
	• Une simple transaction de carte est envoyée au serveur de la banque. • MQ Telemetry identifie la personne concernée parmi des milliers, avertissant le consommateur que sa carte a été utilisée. • MQ Telemetry peut utiliser l'entrée d'information la plus simple et localiser cet individu.

Les scénarios d'utilisation décrits dans les sous-rubriques sont tirés d'exemples réels. Ils fournissent quelques exemples d'utilisation de la télémétrie et abordent les problèmes communs que la technologie devra résoudre.

Windows Linux AIX Scénarios d'utilisation de la télémétrie : Surveillance de patient à domicile

Dans le cadre de la collaboration entre IBM et un établissement de soins pour un système de surveillance d'un patient atteint de maladie cardiaque, un défibrillateur cardioverteur communique avec un hôpital. Les données du patient et de l'appareil implanté sont transférées à l'aide de la télémétrie de fréquence radio au périphérique MQTT au domicile d'un patient.

Le transfert a lieu généralement la nuit vers un émetteur situé a proximité du lit du patient. Cet émetteur transmet les données de manière sécurisée via le système téléphonique à l'hôpital où les données sont analysées.

Le système réduit le nombre de visites que le patient doit effectuer chez le médecin. Il permet de détecter lorsque le patient ou l'appareil doit être contrôlé et, en cas d'urgence, il alerte le médecin de garde.

Les caractéristiques de la collaboration entre IBM et l'établissement de soins sont communes dans plusieurs scénarios d'utilisation de la télémétrie :

Invisibilité

L'appareil ne nécessite aucune intervention de l'utilisateur autre que l'alimentation électrique, une ligne téléphonique et l'obligation pour l'utilisateur de se trouver à proximité de l'appareil une partie de la journée. Son utilisation est simple et fiable.

Pour éviter au patient d'avoir à configurer l'appareil, ce dernier est livré déjà configuré. Le patient doit juste le brancher sur le secteur. Le fait que l'appareil ne soit pas configuré par le patient simplifie son utilisation et diminue le risque de configuration erronée.

Le client MQTT est incorporé au périphérique. Le développeur du périphérique incorpore l'implémentation du client MQTT dans l'appareil et le développeur, ou fournisseur, configure le client MQTT dans le cadre de la configuration.

Le client MQTT est fourni sous la forme d'un fichier JAR Java SE, que le développeur inclut dans son application Java. Pour les environnements non-Java, tels que celui-ci, le développeur du périphérique peut implémenter un client dans une langue différente en utilisant les formats et le protocole MQTT publiés. Le développeur peut également utiliser l'un des clients C fournis en tant que bibliothèques partagées pour les plateformes Windows, Linux et ARM.

Connectivité inégale

La communication entre le défibrillateur et l'hôpital se fait via un réseau dont les caractéristiques sont inégales. Deux réseaux différents sont utilisés pour résoudre les différents problèmes liés à la collecte de données du patient et à l'envoi de ces données à l'hôpital. Entre le patient et l'unité MQTT un réseau de fréquence radio de faible puissance et de portée limitée est utilisé. L'émetteur se connecte à l'hôpital en utilisant une connexion TCP/IP de réseau privé virtuel via une ligne téléphonique à faible bande passante.

Il n'est pas toujours pratique de trouver un moyen de connecter tous les appareils directement à un réseau IP. La solution habituelle consiste à utiliser deux réseaux connectés à un concentrateur. Le périphérique MQTT est un simple concentrateur qui stocke les informations du patient et les transfère ensuite à l'hôpital.

Sécurité

Le médecin doit avoir la preuve de l'authenticité des données du patient et le patient doit pouvoir bénéficier du respect de la confidentialité des données le concernant.

Dans certaines situations, il suffit de chiffrer la connexion à l'aide du réseau privé virtuel ou de TLS. Dans d'autres, il est souhaitable de conserver les données en sécurité même après leur stockage.

Il arrive parfois que le dispositif de télémétrie ne soit pas sûr. Il peut se trouver dans un habitat partagé, par exemple. L'utilisateur du dispositif doit s'authentifier pour s'assurer que les données proviennent du patient approprié. Le dispositif lui-même peut être authentifié sur le serveur à l'aide de TLS, de même que le serveur s'authentifie sur le dispositif.

Le canal de télémétrie entre le dispositif et le gestionnaire de files d'attente prend en charge JAAS pour l'authentification utilisateur et TLS pour le chiffrement de la communication et l'authentification du dispositif. L'accès à une publication est contrôlé par le gestionnaire des droits d'accès aux objets dans IBM MQ.

L'identificateur utilisé pour authentifier l'utilisateur peut être mappé sur un identificateur différent, tel qu'une identité de patient commune. Un identificateur commun simplifie la configuration de l'autorisation d'accès aux rubriques de publication dans IBM MQ.

Connectivité

La connexion entre le périphérique MQTT et l'hôpital s'effectue via une ligne commutée et fonctionne sur une bande passante dont le débit est de 300 bauds.

Pour fonctionner efficacement à 300 bauds, le MQTT protocol ajoute uniquement quelques octets à un message en plus des en-têtes TCP/IP.

Le MQTT protocol fournit une transmission unique de messagerie de type *autonome après diffusion*, qui permet de maintenir un faible temps de latence. Il peut également utiliser plusieurs transmissions pour garantir la distribution *au moins une fois* et *une seule fois* si la distribution garantie est plus importante que le temps de réponse. Pour garantir la distribution, les messages sont stockés sur le dispositif jusqu'à ce qu'ils soient distribués. Si un dispositif est relié via une connexion sans fil, la distribution garantie est particulièrement utile.

Evolutivité

Les dispositifs de télémétrie sont normalement déployés en grande quantité, de dizaines de milliers à plusieurs millions d'unité.

La connexion de dispositifs sur un système entraîne des demandes importantes pour une solution. Les demandes sont d'ordre commercial, le coût des dispositifs et leur logiciel, et d'ordre administratif, la gestion des licences, des dispositifs et des utilisateurs. Les demandes d'ordre technique sont relatives à la charge sur le réseau et sur les serveurs.

L'ouverture des connexions utilise plus de ressources serveur que le maintien des connexions ouvertes. Mais dans un scénario tel que celui-ci, l'utilisation de lignes téléphoniques, le développement des connexions signifie que les connexions ne sont pas ouvertes plus longtemps que nécessaire. Les transferts de données sont largement effectués par lots. Les connexions peuvent être planifiées pendant la nuit de manière à éviter une augmentation soudaine des connexions à l'heure du coucher.

Au niveau du client, l'évolutivité des clients est favorisé par la configuration client minimale requise. Le client MQTT est incorporé au périphérique. Aucune étape de configuration ou d'acceptation de licence client MQTT ne doit être intégrée au déploiement des périphériques vers les patients.

Sur le serveur, MQ Telemetry a une cible initiale de 50 000 connexions ouvertes par gestionnaire de files d'attente.

Les connexions sont gérée à l'aide d'IBM MQ Explorer. IBM MQ Explorer filtre les connexions qui doivent être affichées pour un nombre gérable. Avec un schéma choisi d'identificateurs d'allocation aux clients, vous pouvez filtrer des connexions en fonction de l'emplacement géographique ou par ordre alphabétique de noms de patient.

Windows Linux AIX Scénarios d'utilisation de la télémétrie : Surveillance et contrôle de l'énergie à domicile

Les compteurs intelligents collectent plus d'informations sur la consommation d'énergie que les compteurs traditionnels.

Les compteurs intelligents sont souvent couplés à un réseau de télémétrie local pour surveiller et contrôler les appareils individuels dans une maison. Certains d'entre eux sont également connectés à distance pour la surveillance et le contrôle à distance.

La connexion à distance pourrait être configurée par un individu, une unité d'alimentation ou un point de contrôle central. Le point de contrôle à distance peut lire les données d'utilisation de l'alimentation et fournir des données d'utilisation. Il peut fournir des données pour influencer cette utilisation telles que la tarification continue et les informations météorologiques. Il peut limiter la charge pour améliorer l'efficacité globale de la production d'alimentation électrique.

Les compteurs intelligents commencent à se déployer à grande échelle. Au Royaume Uni, par exemple, le gouvernement est en pleine consultation pour l'installation de compteurs intelligents dans chaque foyer.

Les scénarios d'utilisation de compteurs à domicile ont plusieurs caractéristiques en commun :

Invisibilité

Le compteur ne doit nécessiter aucune intervention de l'utilisateur, sauf si ce dernier le souhaite pour économiser de l'énergie. Le compteur ne doit pas affecter la fiabilité de fourniture d'énergie aux appareils individuels.

Un client MQTT peut être intégré au logiciel déployé avec le compteur, et ne nécessite pas d'installation ou de configuration séparée.

Connectivité inégale

La communication entre les appareils et le compteur intelligent nécessite des normes différentes de connectivité plus importantes qu'entre le compteur et le point de connexion à distance.

La connexion entre le compteur intelligent et les appareils doit être hautement disponible et être conforme aux normes de réseau applicables à un réseau domestique.

Le réseau à distance est susceptible d'utiliser plusieurs connexions physiques. Certaines d'entre elles, comme les connexions cellulaires, ont un coût de transmission élevé et peuvent être intermittentes. La spécification MQTT version 3 est destinée aux connexions à distance et aux connexions entre les adaptateurs locaux et le compteur intelligent.

La connexion entre les prises d'alimentation et les appareils, et le compteur utilise un réseau domestique, tel que Zigbee. MQTT for sensor networks (MQTT-S), a été conçu pour fonctionner avec Zigbee et d'autres protocoles de réseau à faible bande passante. MQ Telemetry ne prend pas en charge directement MQTT-S. Il nécessite une passerelle pour connecter MQTT-S à MQTT v3.

Comme dans le cas de la surveillance de patient à domicile, les solutions pour la surveillance et le contrôle nécessitent plusieurs réseaux connectés en utilisant le compteur intelligent comme un concentrateur.

Sécurité

Il existe plusieurs problèmes de sécurité liés aux compteurs intelligents. Ces problèmes comprennent l'irréfutabilité des transactions, l'autorisation des actions de contrôle qui sont initiées et la confidentialité des données de consommation électrique.

Pour assurer la confidentialité des données, les données transférées entre le compteur et le point de contrôle à distance par MQTT peuvent être chiffrées avec TLS. Pour assurer l'autorisation des actions de contrôle, la connexion MQTT entre le compteur et le point de contrôle à distance peut être mutuellement authentifiée à l'aide de TLS.

Connectivité

La nature physique du réseau à distance peut varier considérablement. Le réseau peut utiliser une connexion à large bande existante ou un réseau mobile avec des coûts d'appel élevés et une disponibilité intermittente. En matière de coûts élevés et d'intermittence, les connexions MQTT constituent un protocole fiable et efficace ; voir «Scénarios d'utilisation de la télémétrie : Surveillance de patient à domicile», à la page 119.

Evolutivité

Finalement, les société de distribution d'électricité ou les points de contrôle centraux envisagent de déployer des dizaines de millions de compteurs intelligents. Initialement le nombre de compteurs prévu par déploiement se situait entre des dizaines et des centaines de milliers. Ce chiffre est comparable à la cible MQTT initiale de 50 000 connexions client ouvertes par gestionnaire de files d'attente.

Un aspect critique de l'architecture pour la surveillance et le contrôle de l'énergie à domicile est l'utilisation du compteur intelligent comme concentrateur de réseau. Chaque adaptateur d'appareil est un détecteur séparé. En les connectant à un concentrateur local utilisant MQTT, le concentrateur peut concentrer le flux de données sur une simple session TCP/IP avec le point de contrôle central, et peut également stocker les messages pendant une courte période pour surmonter l'indisponibilité de session.

Les connexions à distance doivent être laissées ouvertes dans les scénarios d'énergie à domicile pour deux raisons. Tout d'abord parce que l'ouverture des connexions peut prendre du temps lié à l'envoi des demandes. Le temps nécessaire à l'ouverture de nombreuses connexions pour envoyer des demandes de "limitation de charge" dans un intervalle court est trop long. Ensuite, pour recevoir des demandes de limitation de charge à partir de la société distributrice d'électricité, la connexion doit d'abord être ouverte par le client. Avec MQTT, les connexions sont toujours démarrées par le client, et pour recevoir les demandes de limitation de charge de la société distributrice d'électricité, la connexion doit restée ouverte.

Si le débit de l'ouverture de connexions est critique, ou que le serveur initie des demandes prioritaires, la solution consiste généralement à conserver plusieurs connexions ouvertes.

Windows Linux AIX Scénarios d'utilisation de télémétrie : Identification par radio fréquences (RFID)

La technologie RFID consiste en l'utilisation d'une étiquette RFID pour identifier et faire le suivi d'un objet via une liaison sans fil. Les étiquettes RFID peuvent être lues par un grand nombre de compteurs et, ceci même si l'étiquette n'est pas dans l'axe de visée du lecteur RFID. Les étiquettes passives sont activées par un lecteur RFID. Les étiquettes actives émettent sans activation externe. Les étiquettes actives doivent disposer d'une alimentation. Les étiquettes passives peuvent incorporer une alimentation afin d'en accroître la plage.

La technologie RFID est utilisée dans de nombreuses applications et les types de scénario d'utilisation peuvent varier énormément. Les scénarios d'utilisation RFID, ainsi que les scénarios d'utilisation de la surveillance de patient à domicile, et du contrôle et de la surveillance énergétique à domicile comportent des similitudes et des différences.

Invisibilité

Dans de nombreux scénarios d'utilisation, le lecteur RFID est déployé en grand nombre et doit fonctionner sans intervention de l'utilisateur. Le lecteur inclut un client MQTT intégré pour communiquer avec un point de contrôle central.

Par exemple, dans un entrepôt de distribution, un lecteur mobile pour détecter une palette. Le lecteur active l'étiquette RFID des articles se trouvant sur la palette et envoie les données et les demandes aux applications centrales. Les données sont utilisées pour la mise à jour de l'emplacement du stock. Les demandes contrôlent ensuite l'étape suivante pour la palette, comme par exemple son déplacement vers une baie particulière. Les compagnies aériennes, et les systèmes de gestion des bagages dans les aéroports utilisent la technologie RFID de la sorte.

Dans certains cas d'utilisation RFID, le lecteur dispose d'un environnement informatique standard, tel que Java Platform, Micro Edition (Java ME). Dans ces cas, le client MQTT peut être déployé dans une étape de configuration distincte, après la fabrication.

Connectivité inégale

Les lecteurs RFID doivent être séparés de l'unité de contrôle locale qui contient un client MQTT, ou chaque lecteur peut disposer d'un client MQTT intégré. Normalement, les facteurs de localisation géographique ou de communications indiquent le choix de la topologie.

Sécurité

La confidentialité et l'authenticité sont des préoccupations de sécurité lors de l'utilisation des étiquettes RFID. Les étiquettes RFID sont peu visibles, et peuvent être surveillées secrètement, espionnées ou trafiquées.

La solution aux problèmes de sécurité RFID accroît l'opportunité de déployer de nouvelles solutions RFID. Bien que l'exposition à la sécurité se trouve au sein même de l'étiquette RFID, et dans le lecteur local, l'utilisation du traitement d'informations centrales suppose des approches permettant de contrer les différentes menaces. Par exemple, le trafic d'étiquette peut être détecté en mettant en corrélation de manière dynamique les niveaux de stock et les livraisons et répartitions.

Connectivité

Les applications RFID comprenaient à la fois le stockage et la retransmission d'informations par lots collectées à partir des lecteurs RFID et les demandes immédiates. Dans le scénario d'entrepôt de distribution, le lecteur RFID est connecté en permanence. Lorsqu'une étiquette est lue, elle est publiée avec les informations relatives au lecteur. L'application de l'entrepôt renvoie la réponse au lecteur.

Dans l'application d'entreposage, le réseau est normalement fiable et les demandes immédiates peuvent utiliser des messages *autonomes après diffusion* pour des temps d'attente faibles. Les données par lot en mode différé peuvent utiliser la messagerie *une seule fois* pour minimiser les coûts de gestion associés à la perte de données.

Evolutivité

Si l'application RFID nécessite des réponses immédiates, de l'ordre d'une seconde ou deux, les lecteurs RFID doivent rester connectés.

Scénarios d'utilisation de la télémétrie : Détection d'environnement

La détection d'environnement utilise la télémétrie pour collecter des informations sur le niveau et la qualité des eaux de rivière, les polluants atmosphériques et d'autres données environnementales.

Les détecteurs sont souvent localisés dans des lieux distants sans accès aux communications filaires. La bande passante sans fil est onéreuse et sa fiabilité peut être faible. En général, plusieurs détecteurs d'environnement dans une zone géographique réduite sont connectés à une unité de surveillance locale dans un emplacement sûr. Les connexions locales peuvent être filaires ou sans fil.

Invisibilité

Les unités de détecteur risquent d'être moins accessibles, moins bien alimentées et déployées en plus grand nombre que l'unité de contrôle centrale. Les détecteurs sont parfois "muets", et le dispositif de surveillance locale inclut les adaptateurs pour transformer et stocker les données de détecteur. L'unité de surveillance est susceptible d'incorporer un ordinateur à usage général prenant en charge Java Platform, Standard Edition (Java SE) ou Java Platform, Micro Edition (Java ME). Il est peu probable que l'invisibilité soit une exigence majeure lors de la configuration du client MQTT.

Connectivité inégale

Les fonctions de capteurs et le coût de la bande passante de la connexion à distance se traduit de manière classique par l'utilisation d'un concentrateur de surveillance local connecté au serveur central.

Sécurité

La sécurité n'est pas une exigence majeure sauf si elle est utilisée dans un scénario d'utilisation militaire ou de défense.

Connectivité

Plusieurs utilisateurs ne nécessitent pas la surveillance continue ou la disponibilité des données. Les données d'exception, telles que l'alerte en cas d'inondation, doivent être transférées immédiatement. Les données de détecteur sont agrégées sur le moniteur local pour réduire les coûts de connexion et de communication, puis elles sont transférées à l'aide de connexions planifiées. Les données d'exception sont transférées dès qu'elles sont détectées au niveau du moniteur.

Evolutivité

Les détecteurs sont concentrés autour des concentrateurs locaux et les données de détecteur sont agrégées en paquets qui sont transmis suivant un planning. Ces deux facteurs réduisent la charge sur le serveur central qui serait imposée par l'utilisation directe de détecteurs connectés.

Windows Linux AIX Scénarios d'utilisation de la télémétrie : Applications mobiles

Les applications mobiles sont des applications qui s'exécutent sur des dispositifs sans fil. Ces dispositifs sont soit des plateformes d'application génériques, soit des dispositifs personnalisés.

Les plateformes habituelles comprennent des ordinateurs de poche tels que les téléphones et les assistants de données personnels et les appareils portables tels que les ordinateurs bloc-notes. Les unités personnalisées utilisent un matériel spécialement adapté à des applications spécifiques. Un dispositif permettant de livrer un colis contre signature est un exemple de dispositif mobile. Les applications se trouvant sur les dispositifs mobiles sont souvent conçues sur une plateforme logicielle générique.

Invisibilité

Le déploiement d'applications mobiles personnalisées est géré et peut inclure la configuration d'application client MQTT. Il est peu probable que l'invisibilité soit une exigence majeure lors de la configuration du client MQTT.

Connectivité inégale

Contrairement à la topologie de concentrateur local des scénarios précédents, les clients mobiles se connectent à distance. La couche d'application client se connecte directement à une application du concentrateur central.

Sécurité

Avec peu de sécurité physique, l'unité mobile et l'utilisateur mobile doivent être authentifiés. TLS est utilisé pour confirmer l'identité de l'unité, et JAAS pour authentifier l'utilisateur.

Connectivité

Si l'application mobile dépend de la couverture sans fil, elle doit pouvoir fonctionner hors ligne et faire face efficacement à une interruption de la connexion. Dans cet environnement, l'objectif est de rester connecté, mais l'application doit pouvoir stocker et transférés les messages. Les messages, qui sont souvent des commandes ou des confirmations de distribution, ont une importante valeur commerciale. Ils doivent être stockés et transférés de manière fiable.

Evolutivité

L'évolutivité n'est pas un élément majeur. Le nombre de clients d'application n'est pas censé dépasser des milliers ou des dizaines de milliers d'unités dans les scénarios d'application mobile.

Windows Linux AIX Connexion de dispositifs de télémétrie à un gestionnaire de files d'attente

Les dispositifs de télémétrie se connectent à un gestionnaire de files d'attente à l'aide du client MQTT version 3. Le client MQTT version 3 utilise TCP/IP pour se connecter à un programme d'écoute TCP/IP appelé service de télémétrie (MQXR).

Lorsque vous connectez un dispositif de télémétrie à un gestionnaire de files d'attente, le client MQTT démarre une connexion TCP/IP en utilisant la méthode `MqttClient.connect`. A l'instar des clients IBM MQ, un client MQTT doit être connecté au gestionnaire de files d'attente pour envoyer et recevoir des messages. La connexion est effectuée au niveau du serveur à l'aide du programme d'écoute TCP/IP installé avec MQ Telemetry, appelé service de télémétrie (MQXR). Chaque gestionnaire de files d'attente exécute au maximum un service de télémétrie (MQXR).

Le service de télémétrie (MQXR) utilise l'adresse de connecteur à distance définie par chaque client dans la méthode `MqttClient.connect` pour allouer la connexion à un canal de télémétrie. Une adresse de connecteur est une combinaison de nom d'hôte TCP/IP et de numéro de port. Des clients multiples qui utilisent la même adresse de connecteur éloigné sont connectés au même canal de télémétrie par le service de télémétrie (MQXR).

S'il existe plusieurs gestionnaires de files d'attente sur un serveur, partagez les canaux de télémétrie entre les gestionnaires de files d'attente. Affectez les adresses de connecteur distant entre les gestionnaires de files d'attente. Définissez chaque canal de télémétrie à l'aide d'une adresse de connecteur unique. Deux canaux de télémétrie ne doivent pas utiliser la même adresse de connecteur.

Si la même adresse de connecteur à distance est configurée pour les canaux de télémétrie sur plusieurs gestionnaires de files d'attente, le premier canal de télémétrie à connecter l'emporte. Les canaux suivants qui se connectent à la même adresse échouent.

S'il existe plusieurs adaptateurs réseau sur le serveur, séparez les adresses de connecteur à distance entre les canaux de télémétrie. L'affectation des adresses de connecteur est totalement arbitraire, dans la mesure où toute adresse de connecteur particulière n'est configurée que sur un seul canal de télémétrie.

Configurez IBM MQ pour connecter les clients MQTT en utilisant les assistants fournis dans le supplément MQ Telemetry pour IBM MQ Explorer. Vous pouvez également suivre les instructions indiquées dans les sections Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Linux et AIX et Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Windows pour configurer manuellement la télémétrie.

Référence associée

Propriétés de MQXR

Windows Linux AIX Protocoles de connexion pour la télémétrie

MQ Telemetry prend en charge IPv4 et IPv6, et TLS.

Windows Linux AIX Service de télémétrie (MQXR)

Le service de télémétrie (MQXR) est un programme d'écoute TCP/IP qui est géré comme un service IBM MQ. Créez le service à l'aide d'un assistant IBM MQ Explorer ou avec une commande **runmqsc**.

Le service MQ Telemetry (MQXR) est appelé SYSTEM.MQXR.SERVICE .

L'assistant Modèle de configuration de télémétrie fourni dans la fonction MQ Telemetry pour IBM MQ Explorer, crée le service de télémétrie et un exemple de canal de télémétrie. Voir Vérification de l'installation de MQ Telemetry en utilisant IBM MQ Explorer .

Créez le modèle de configuration à partir de la ligne de commande ; voir Vérification de l'installation de MQ Telemetry à l'aide de la ligne de commande.

Le service de télémétrie (MQXR) démarre et s'arrête automatiquement avec le gestionnaire de files d'attente. Contrôlez le service à l'aide du dossier de services figurant dans IBM MQ Explorer. Pour afficher le service, vous devez cliquer sur l'icône pour arrêter le IBM MQ Explorer filtrage des objets SYSTEM à partir de l'affichage.

Pour un exemple de création manuelle du service, voir

- Linux AIX Création du SYSTEM.MQXR.SERVICE sur Linux.
- Windows Création du SYSTEM.MQXR.SERVICE sur Windows.

Ces rubriques spécifient également la clé par défaut pour exiger le chiffrement des phrases passe pour les canaux TLS MQTT. Pour plus d'informations, voir Chiffrement des phrases secrètes pour les canaux TLS MQTT.

Windows Linux AIX Canaux de télémétrie

Créez des canaux de télémétrie pour créer des connexions avec des propriétés différentes, telles que l'authentification JAAS (Java Authentication and Authorization Service) ou TLS, ou pour gérer des groupes de clients.

Créez des canaux de télémétrie à l'aide de l'assistant **New Telemetry Channel** , fourni dans la fonction MQ Telemetry pour IBM MQ Explorer. Configurez un canal, en utilisant l'assistant, pour accepter les

connexions depuis les clients MQTT sur un port TCP/IP donné. Depuis IBM WebSphere MQ 7.1, vous pouvez configurer MQ Telemetry en utilisant le programme de ligne de commande **runmqsc**.

Créez plusieurs canaux de télémétrie sur différents ports pour faciliter la gestion d'un grand nombre de connexions client, en divisant les clients en groupes. Chaque canal de télémétrie possède un nom différent.

Vous pouvez configurer les canaux de télémétrie avec des attributs de sécurité différents pour créer des types de connexion différents. Créez plusieurs canaux pour accepter des connexions client sur des adresses TCP/IP différentes. Utilisez TLS pour chiffrer les messages et authentifier le canal de télémétrie et le client. Voir Configuration TLS des clients MQTT et des canaux de télémétrie. Spécifiez un ID utilisateur pour simplifier l'autorisation d'accès aux objets IBM MQ. Spécifiez une configuration JAAS pour authentifier l'utilisateur MQTT avec JAAS. Voir MQTT client identification, authorization, and authentication.

Windows Linux AIX Protocole IBM MQ Telemetry Transport

Le protocole IBM MQ Telemetry Transport (MQTT) v3 est conçu pour l'échange de messages entre des petits périphériques sur une faible bande passante, ou pour les connexions coûteuses et envoyer des messages de façon fiable. Il utilise TCP/IP.

Le MQTT protocol est publié ; voir Format et protocole IBM MQ Telemetry Transport. La version 3 du protocole utilise la publication/abonnement et prend en charge trois qualités de service : *autonome après diffusion*, *au moins une fois* et *une seule fois*.

La taille réduite des en-têtes de protocole et la charge de message du tableau d'octets permet de conserver la taille réduite du message. Les en-têtes comprennent un en-tête à deux octets fixes et jusqu'à 12 octets d'en-têtes variables supplémentaires. Le protocole utilise les en-têtes variables à 12 octets pour l'abonnement et la connexion, et uniquement 2 en-têtes variables de 2 octets pour la plupart des publications.

Avec trois qualités de service, vous pouvez établir un compromis entre la faible latence et la fiabilité. Voir Qualities of service provided by an MQTT client. La qualité de service *autonome après diffusion* n'utilise aucun stockage de dispositif persistant et ne prend en charge qu'une seule transmission pour envoyer ou recevoir une publication. Les qualités de service *au moins une fois* et *une seule fois* nécessitent le stockage persistant sur l'unité pour conserver l'état du protocole et sauvegarder un message jusqu'à la réception de l'accusé de réception.

Windows Linux AIX Clients MQTT

Une application client MQTT permet la collecte d'informations provenant du dispositif de télémétrie, la connexion au serveur et la publication d'informations sur le serveur. L'application peut également s'abonner à des rubriques, recevoir des publications et contrôler le dispositif de télémétrie.

Contrairement aux applications client IBM MQ, les clients MQTT ne sont pas des applications IBM MQ. Ils n'indiquent pas un gestionnaire de files d'attente auquel se connecter. Ils ne sont pas limités à l'utilisation d'interfaces de programmation IBM MQ. Au contraire, les clients MQTT implémentent le protocole MQTT 3. Vous pouvez écrire votre propre bibliothèque client pour l'interface avec le MQTT protocol dans le langage de programmation et sur la plateforme de votre choix. Voir Format et protocole IBM MQ Telemetry Transport.

Pour simplifier l'écriture d'applications client MQTT, utilisez les bibliothèques client C, Java et JavaScript qui encapsulent le MQTT protocol d'un certain nombre de plateformes. Si vous incorporez ces bibliothèques dans vos applications MQTT, un client MQTT complètement fonctionnel peut comporter seulement 15 lignes de code. Les bibliothèques client MQTT sont disponibles gratuitement sur Eclipse Paho et MQTT.org. Voir IBM MQ Telemetry Transport sample programs.

L'application client MQTT est toujours responsable de l'initiation d'une connexion avec un canal de télémétrie. Une fois connectée, l'application client MQTT ou une application IBM MQ peut commencer à échanger des messages.

Les applications client MQTT et les applications IBM MQ diffusent des publications et s'abonnent au même ensemble de rubriques. Une application IBM MQ peut également envoyer un message directement à une application client MQTT sans que cette dernière ait besoin de créer préalablement un abonnement. Voir Configurer la mise en file d'attente distribuée pour envoyer des messages aux clients MQTT.

Les applications client MQTT sont connectées à IBM MQ via un canal de télémétrie. Ce canal fait office de pont entre les différents types de message utilisés par MQTT et IBM MQ. Il crée des publications et des abonnements dans le gestionnaire de files d'attente pour le compte de l'application client MQTT. Le canal de télémétrie envoie des publications qui correspondent aux abonnements d'une application client MQTT d'un gestionnaire de files d'attente vers l'application client MQTT.

Windows Linux AIX Envoi d'un message à un client MQTT

Les applications IBM MQ peuvent envoyer des messages aux clients MQTT v3 en les publiant dans les abonnements créés par les clients, ou envoyer les messages directement. Les clients MQTT peuvent s'envoyer des messages en les publiant dans les rubriques auxquelles sont abonnés les autres clients.

Un client MQTT s'abonne à une publication qu'il reçoit d'IBM MQ.

Réalisez la tâche, «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130 pour envoyer une publication d'IBM MQ à un client MQTT.

Un client MQTT v3 reçoit des messages en créant généralement un abonnement à une rubrique ou groupe de rubriques. Dans l'exemple de fragment de code, Figure 44, à la page 129, le client MQTT s'abonne en utilisant la chaîne de rubrique `"MQTT Examples"`. Une application IBM MQ C, Figure 45, à la page 129, publie dans la rubrique à l'aide de la chaîne de rubrique `"MQTT Examples"`. Dans le fragment de code à la Figure 46, à la page 129, le client MQTT reçoit la publication dans la méthode de rappel, `messageArrived`.

Pour plus d'informations sur la façon de configurer IBM MQ pour envoyer des publications en réponse aux abonnements des clients MQTT, voir Publication d'un message en réponse à un abonnement client MQTT.

Une application IBM MQ envoie un message directement à un client MQTT.

Réalisez la tâche «Envoi d'un message à un client MQTT en utilisant IBM MQ Explorer», à la page 134 pour envoyer un message directement d'IBM MQ à un client MQTT.

Un message envoyé de cette manière à un client MQTT s'appelle un message non sollicité. Les clients MQTT v3 reçoivent des messages non sollicités comme publications avec un nom de rubrique défini. Le service de télémétrie (MQXR) détermine le nom de la rubrique sur le nom de la file d'attente éloignée.

Pour plus d'informations sur la configuration d'IBM MQ pour envoyer des messages directement aux clients MQTT, voir Envoi direct d'un message à un client.

Un client MQTT publie un message.

Un client MQTT v3 peut publier un message reçu par un autre client MQTT v3, mais il ne peut pas envoyer de message non sollicité. Le fragment de code à la Figure 47, à la page 130 montre comment un client MQTT v3 écrit en Java, publie un message.

En général, pour envoyer un message à un client MQTT v3, chaque client crée un abonnement à son propre identifiant `ClientIdentifier`. Réalisez la tâche «Publication d'un message vers un client MQTT v3», à la page 136 pour publier un message d'un client MQTT vers un autre client MQTT en utilisant `ClientIdentifier` comme chaîne de rubrique.

Exemples de fragments de code

Le fragment de code à la Figure 44, à la page 129 montre comment un client MQTT écrit en Java crée un abonnement. Il a également besoin d'une méthode de rappel, `messageArrived` pour recevoir les publications relatives à l'abonnement.

```
String    clientId = String.format("%-23.23s",
                     System.getProperty("user.name") + "_" +
                     (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient  client = new MqttClient("localhost", clientId);
String topicString = "MQTT Examples";
int             QoS = 1;
client.subscribe(topicString, QoS);
```

Figure 44. Abonné client MQTT v3

Le fragment de code à la Figure 45, à la page 129 montre comment une application IBM MQ écrite en C envoie une publication. Le fragment de code est extrait à partir de la tâche Création d'un diffuseur de publications sur une rubrique variable.

```
/* Define and set variables to.defaults */
/* Omitted lines declaring variables    */
char * topicName   = ""
char * topicString = "MQTT Examples"
char * publication = "Hello world!";
do {
  MQCONN(qMgrName, &Hconn, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  td.ObjectType = MQOT_TOPIC;    /* Object is a topic             */
  td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4     */
  strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
  td.ObjectString.VSPtr = topicString;
  td.ObjectString.VSLength = (MQLONG)strlen(topicString);
  MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
  MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
  if (CompCode != MQCC_OK) break;
  MQDISC(&Hconn, &CompCode, &Reason);
} while (0);
```

Figure 45. Abonné IBM MQ

Lorsque la publication arrive, le client MQTT appelle la méthode `messageArrived` de la classe `MqttCallback` du client d'application MQTT .

```
public class CallBack implements MqttCallback {
  public void messageArrived(MqttTopic topic, MqttMessage message) {
    try {
      System.out.println("Message arrived: \"" + message.toString()
          + "\" on topic \"" + topic.toString() + "\"");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
// ... Other callback methods
}
```

Figure 46. `messageArrived`méthode

La Figure 47, à la page 130 montre MQTT v3 publiant un message vers l'abonnement créé dans la Figure 44, à la page 129.

```
String          address = "localhost";
String         clientId = String.format("%-23.23s",
                          System.getProperty("user.name") + "_" +
                          (UUID.randomUUID().toString())).trim()).replace('-', '_');
MqttClient       client = new MqttClient(address, clientId);
String      topicString = "MQTT Examples";
MqttTopic         topic = client.getTopic(Example.topicString);
String     publication  = "Hello world";
MqttMessage     message = new MqttMessage(publication.getBytes());
MqttDeliveryToken token = topic.publish(message);
```

Figure 47. Abonné client MQTT v3

Windows Linux AIX Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer

Suivez les étapes de cette tâche pour publier un message en utilisant IBM MQ Explorer et vous abonner avec l'utilitaire client MQTT. Une tâche supplémentaire indique comment configurer un alias de gestionnaire de files d'attente plutôt que de paramétrer la file d'attente de transmission sur `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Avant de commencer

La tâche suppose que vous connaissiez IBM MQ et IBM MQ Explorer et qu'IBM MQ et la fonction MQ Telemetry soient installés.

L'utilisateur qui crée les ressources du gestionnaire de files d'attente doit disposer de droits suffisants pour cette opération. À des fins de démonstration, l'ID utilisateur IBM MQ Explorer est supposé être membre du groupe `mqm`.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous créez une rubrique dans IBM MQ et vous vous y abonnez à l'aide de l'utilitaire client MQTT. Lorsque vous publiez dans la rubrique en utilisant IBM MQ Explorer, le client MQTT reçoit la publication.

Procédure

Effectuez l'une des tâches suivantes :

- Vous avez installé MQ Telemetry, mais ne l'avez pas encore démarré. Effectuez la tâche suivante : «Démarrage de la tâche sans service de télémétrie (MQXR) défini», à la page 131
- Vous avez exécuté IBM MQ Telemetry auparavant, mais voulez utiliser un nouveau gestionnaire de files d'attente pour effectuer les démonstrations. Effectuez la tâche suivante : «Démarrage de la tâche sans service de télémétrie (MQXR) défini», à la page 131
- Vous souhaitez effectuer la tâche à l'aide d'un gestionnaire de files d'attente existant et dont les ressources de télémétrie ne sont pas définies. Vous ne souhaitez pas exécuter l'assistant **Définition d'un modèle de configuration**.

  a. Effectuez l'une des tâches suivantes pour configurer la télémétrie :

  - Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Linux et AIX
  - Configuration d'un gestionnaire de files d'attente pour la télémétrie sous Windows

  b. Effectuez la tâche suivante : «Démarrage de la tâche avec un service de télémétrie (MQXR) en cours d'exécution», à la page 132
- Si vous souhaitez effectuer la tâche à l'aide d'un gestionnaire de files d'attente existant dont les ressources de télémétrie sont déjà définies, suivez la procédure «Démarrage de la tâche avec un service de télémétrie (MQXR) en cours d'exécution», à la page 132.

Que faire ensuite

Effectuez la procédure «Envoi d'un message à un client MQTT en utilisant IBM MQ Explorer», à la page 134 pour envoyer un message directement à l'utilitaire client.

Démarrage de la tâche sans service de télémétrie (MQXR) défini

Créez un gestionnaire de files d'attente et exécutez l'assistant **Définition d'un modèle de configuration** pour paramétrer les ressources de télémétrie exemples du gestionnaire de files d'attente. Publiez un message à l'aide d'IBM MQ Explorer et abonnez-vous à cette publication à l'aide de l'utilitaire client MQTT.

Pourquoi et quand exécuter cette tâche

Lorsque vous définissez les ressources de télémétrie à l'aide de l'assistant **Définition d'un modèle de configuration**, ce dernier définit les droits de l'ID utilisateur invité. Envisagez soigneusement l'octroi des droits de cette manière à l'ID utilisateur invité. `guest` sous Windowset `nobody` sous Linuxsont autorisés à publier et à s'abonner à la racine de l'arborescence de rubriques et à placer des messages dans `SYSTEM.MQTT.TRANSMIT.QUEUE`.

L'assistant définit également la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`, ce qui peut interférer avec les applications exécutées sur un gestionnaire de files d'attente existant. Il est possible, mais laborieux, de configurer la télémétrie et de ne pas utiliser la file d'attente de transmission. Pour ce faire, effectuez la tâche «Utilisation d'un alias de gestionnaire de files d'attente», à la page 133. Cette tâche permet de créer un gestionnaire de files d'attente afin d'éviter toute interférence potentielle avec une file d'attente de transmission existante par défaut.

Procédure

1. A l'aide d'IBM MQ Explorer, créez et lancez un nouveau gestionnaire de files d'attente.
   a) Clic droit sur le dossier `Queue Managers` > **Nouveau** > **Gestionnaire de files d'attente...**. Entrez un nom de gestionnaire de files d'attente, puis cliquez sur **Terminer**.

      Créez un nom de gestionnaire de files d'attente ; par exemple, `MQTTQMGR`.
2. Créez et lancez le service de télémétrie (MQXR), puis créez un modèle de canal de télémétrie.
   a) Ouvrez le dossier `Queue Managers\`*`QmgrName`*`\Telemetry`.
   b) Cliquez sur **Définir la configuration de l'exemple...** > **Terminer**

      Laissez la case à cocher **Lancer l'utilitaire client MQTT** sélectionnée.
3. Créez un abonnement pour `MQTT Example` à l'aide de l'utilitaire client MQTT .
   a) Cliquez sur **Connexion**.

      L' **historique client** enregistre un événement `Connected` .
   b) Entrez `MQTT Example` dans la zone **Subscription \ Topic** > **Subscribe**.

   L' **historique client** enregistre un événement `Subscribed` .
4. Créez `MQTTExampleTopic` dans IBM MQ.
   a) Clic droit sur le dossier `Queue Managers\`*`QmgrName`*`\Topics` dans **Explorateur MQ**> **Nouveau** > **Rubrique**.
   b) Entrez `MQTTExampleTopic` dans la zone **Nom**, puis cliquez sur **Suivant**.
   c) Entrez `MQTT Example` comme **Chaîne de rubrique** > **Terminer**.
   d) Cliquez sur **OK** pour fermer la fenêtre d'accusé de réception.
5. Publiez `Hello World!` dans la rubrique `MQTT Example` à l'aide de IBM MQ Explorer.
   a) Cliquez sur le dossier `Queue Managers\`*`QmgrName`*`\Topics` dans le fichier IBM MQ Explorer.
   b) Clic droit sur `MQTTExampleTopic` > **Publication de test...**

c) Entrez `Hello World!` dans le champ **Données de message** > **Publier le message** > Passer à la fenêtre MQTT Client Utility.

L' **historique client** enregistre un événement `Received` .

Démarrage de la tâche avec un service de télémétrie (MQXR) en cours d'exécution

Créez un canal de télémétrie ainsi qu'une rubrique. Autorisez l'utilisateur à employer la rubrique et la file d'attente de transmission de télémétrie. Publiez un message à l'aide d'IBM MQ Explorer et abonnez-vous à cette publication à l'aide de l'utilitaire client MQTT.

Avant de commencer

Dans cette version de la tâche, un gestionnaire de files d'attente, *QmgrName*, est défini et en cours d'exécution. Un service de télémétrie (MQXR) est défini et en cours d'exécution. Ce service de télémétrie (MQXR) peut avoir été créé manuellement ou à l'aide de l'assistant **Définition d'un modèle de configuration**.

Pourquoi et quand exécuter cette tâche

Dans cette tâche, vous configurez un gestionnaire de files d'attente existant afin d'envoyer une publication à l'utilitaire client MQTT.

L'étape «1», à la page 132 de la tâche définit la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`, ce qui peut interférer avec les applications exécutées sur un gestionnaire de files d'attente existant. Il est possible, mais laborieux, de configurer la télémétrie et de ne pas utiliser la file d'attente de transmission. Pour ce faire, effectuez la tâche «Utilisation d'un alias de gestionnaire de files d'attente», à la page 133.

Procédure

1. Définissez `SYSTEM.MQTT.TRANSMIT.QUEUE` comme file d'attente de transmission par défaut.
   a) Clic droit sur `Queue Managers\`*`QmgrName`* `folder` > **Propriétés...**
   b) Cliquez sur **Communication** dans le navigateur.
   c) Cliquez sur **Sélectionner ...** > Sélectionner `SYSTEM.MQTT.TRANSMIT.QUEUE` > **OK** > **OK**.
2. Créez un canal de télémétrie `MQTTExampleChannel` pour connecter l'utilitaire client MQTT à IBM MQ, et démarrez l'utilitaire client MQTT.
   a) Clic droit sur le dossier `Queue Managers\`*`QmgrName`* `\Telemetry\Channels` dans **Explorateur MQ**> **Nouveau** > **Canal de télémétrie...**.
   b) Entrez `MQTTExampleChannel` dans la zone **Nom du canal**, puis cliquez sur **Suivant**, et à nouveau sur **Suivant**.
   c) Remplacez l'**ID utilisateur fixe** sur le panneau d'autorisation du client par l'ID utilisateur qui va publier et s'abonner à `MQTTExample` > **Suivant**.
   d) Laissez l'option **Lancer l'utilitaire client** sélectionnée, puis cliquez sur **Terminer**.
3. Créez un abonnement pour `MQTT Example` à l'aide de l'utilitaire client MQTT .
   a) Cliquez sur **Connexion**.

      L' **historique client** enregistre un événement `Connected` .
   b) Entrez `MQTT Example` dans la zone **Subscription \ Topic** > **Subscribe**.

   L' **historique client** enregistre un événement `Subscribed` .
4. Créez `MQTTExampleTopic` dans IBM MQ.
   a) Clic droit sur le dossier `Queue Managers\`*`QmgrName`*`\Topics` dans **Explorateur MQ**> **Nouveau** > **Rubrique**.
   b) Entrez `MQTTExampleTopic` dans la zone **Nom**, puis cliquez sur **Suivant**.
   c) Entrez `MQTT Example` comme **Chaîne de rubrique** > **Terminer**.

d) Cliquez sur **OK** pour fermer la fenêtre d'accusé de réception.

5. Si vous souhaitez qu'un utilisateur, et non le groupe `mqm`, publie et s'abonne à la rubrique `MQTTExample`, procédez de la manière suivante :

   a) Autorisez l'utilisateur à publier et à s'abonner à la rubrique `MQTTExampleTopic` :

   ```
   setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
   ```

   b) Autorisez l'utilisateur à entrer un message sur `SYSTEM.MQTT.TRANSMIT.QUEUE` :

   ```
   setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
   ```

6. Publiez `Hello World!` dans la rubrique `MQTT Example` à l'aide de IBM MQ Explorer.

   a) Cliquez sur le dossier `Queue Managers\`*QmgrName*`\Topics` dans le fichier IBM MQ Explorer.

   b) Clic droit sur `MQTTExampleTopic` > **Publication de test...**

   c) Entrez `Hello World!` dans le champ **Données de message** > **Publier le message** > Passer à la fenêtre MQTT Client Utility.

L' **historique client** enregistre un événement `Received` .

Utilisation d'un alias de gestionnaire de files d'attente

Publiez un message dans l'utilitaire client MQTT à l'aide de IBM MQ Explorer sans définir la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Cette tâche est la suite de la précédente. Elle utilise un alias de gestionnaire de files d'attente afin d'éviter le paramétrage de la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Avant de commencer

Effectuez la tâche «Démarrage de la tâche sans service de télémétrie (MQXR) défini», à la page 131 ou «Démarrage de la tâche avec un service de télémétrie (MQXR) en cours d'exécution», à la page 132.

Pourquoi et quand exécuter cette tâche

Lorsqu'un client MQTT crée un abonnement, IBM MQ envoie sa réponse en utilisant `ClientIdentifier` comme nom de gestionnaire de files d'attente éloignées. Dans cette tâche, il utilise `ClientIdentifier`, `MyClient`.

S'il n'y a pas de file d'attente de transmission ou d'alias de gestionnaire de files d'attente appelé `MyClient`, la réponse est placée dans la file d'attente de transmission par défaut. En définissant la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`, le client MQTT obtient la réponse.

Pour éviter de paramétrer la file d'attente de transmission sur `SYSTEM.MQTT.TRANSMIT.QUEUE`, utilisez les alias de gestionnaire de files d'attente. Vous devez configurer un alias de gestionnaire de files d'attente pour chaque `ClientIdentifier`. En général, le nombre trop important de clients rend difficile l'utilisation d'alias de gestionnaire de files d'attente. `ClientIdentifier` est souvent imprévisible, ce qui rend impossible la configuration de la télémétrie par cette méthode.

Il peut néanmoins arriver que vous ayez besoin de configurer la file d'attente de transmission par défaut sur une valeur autre que `SYSTEM.MQTT.TRANSMIT.QUEUE`. Les étapes de la procédure permettent de configurer un alias de gestionnaire de files d'attente au lieu de définir la file d'attente de transmission par défaut sur `SYSTEM.MQTT.TRANSMIT.QUEUE`.

Procédure

1. Supprimez `SYSTEM.MQTT.TRANSMIT.QUEUE` comme file d'attente de transmission par défaut.

   a) Clic droit sur `Queue Managers\`*QmgrName* `folder` > **Propriétés...**

b) Cliquez sur **Communication** dans le navigateur.

c) Retirez SYSTEM.MQTT.TRANSMIT.QUEUE de la zone **File d'attente de transmission par défaut**, puis cliquez sur **OK**.

2. Vérifiez que vous ne pouvez plus créer d'abonnement avec l'utilitaire client MQTT :

a) Cliquez sur **Connexion**.

L' **historique client** enregistre un événement Connected .

b) Entrez MQTT Example dans la zone **Subscription \ Topic** > **Subscribe**.

L' **historique client** enregistre un événement Subscribe failed et un événement Connection lost .

3. Créez un alias de gestionnaire de files d'attente pour ClientIdentifier, MyClient.

a) Cliquez avec le bouton droit de la souris sur le dossier Queue Managers*QmgrName*\Queues > **Nouveau** > **Définition de file d'attente à distance**.

b) Nommez la définition MyClient, puis cliquez sur **Suivant**.

c) Entrez MyClient dans la zone **Gestionnaire de files d'attente éloignées**.

d) Entrez SYSTEM.MQTT.TRANSMIT.QUEUE dans la zone **File d'attente de transmission**, puis cliquez sur **Terminer**.

4. Reconnectez l'utilitaire client MQTT.

a) Vérifiez que l'option **Identificateur du client** est définie sur MyClient.

b) **Connecter**

L' **historique client** enregistre un événement Connected .

5. Créez un abonnement pour MQTT Example à l'aide de l'utilitaire client MQTT .

a) Cliquez sur **Connexion**.

L' **historique client** enregistre un événement Connected .

b) Entrez MQTT Example dans la zone **Subscription \ Topic** > **Subscribe**.

L' **historique client** enregistre un événement Subscribed .

6. Publiez Hello World! dans la rubrique MQTT Example à l'aide de IBM MQ Explorer.

a) Cliquez sur le dossier Queue Managers*QmgrName*\Topics dans le fichier IBM MQ Explorer.

b) Clic droit sur MQTTExampleTopic > **Publication de test...**

c) Entrez Hello World! dans le champ **Données de message** > **Publier le message** > Passer à la fenêtre MQTT Client Utility.

L' **historique client** enregistre un événement Received .

Windows Linux AIX Envoi d'un message à un client MQTT en utilisant IBM MQ Explorer

Envoyez un message à l'utilitaire client MQTT en plaçant un message dans la file d'attente IBM MQ avec IBM MQ Explorer. Cette tâche indique comment configurer une définition de file d'attente éloignée pour envoyer un message directement à un client MQTT.

Avant de commencer

Exécutez la tâche «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130. Maintenez l'utilitaire client MQTT connecté.

Pourquoi et quand exécuter cette tâche

Cette tâche illustre l'envoi d'un message à un client MQTT à l'aide d'une file d'attente plutôt que par la publication dans une rubrique. Vous ne créez pas d'abonnement dans le client. L'étape «2», à la page 135 de la tâche indique que l'abonnement précédent a été supprimé.

Procédure

1. Supprimez les abonnements existants en déconnectant, puis en reconnectant l'utilitaire client MQTT.

   L'abonnement est supprimé, car (à moins que vous ne changiez les valeurs par défaut) l'utilitaire client MQTT se connecte avec une session propre. Voir Sessions propres.

   Pour faciliter cette tâche, entrez votre propre identifiant `ClientIdentifier` au lieu d'utiliser l'identifiant `ClientIdentifier` créé par l'utilitaire client MQTT.

   a) Cliquez sur l'option de **déconnexion** pour déconnecter l'utilitaire client MQTT du canal de télémétrie.

      L' **historique client** enregistre un événement `Disconnected`

   b) Remplacez l'identifiant **Identificateur de client** par `MyClient`.

   c) Cliquez sur **Connexion**.

   L' **historique client** enregistre un événement `Connected`
2. Vérifiez que l'utilitaire client MQTT ne reçoit plus de publication pour `MQTTExampleTopic`.

   a) Cliquez sur le dossier `Queue Managers\`*`QmgrName`*`\Topics` dans le fichier IBM MQ Explorer.

   b) Clic droit sur `MQTTExampleTopic` > **Publication de test...**

   c) Entrez `Hello World!` dans le champ **Données de message** > **Publier le message** > Passer à la fenêtre MQTT Client Utility.

   Aucun événement n'est enregistré dans l'**historique client**.
3. Créez une définition de file d'attente éloignée pour le client.

   Paramétrez l'identifiant `ClientIdentifier`, `MyClient`, en tant que nom de gestionnaire de files d'attente éloignées dans la définition de file d'attente éloignée. Utilisez le nom de votre choix en tant que nom de file d'attente éloignée. Ce nom est transmis à un client MQTT en tant que nom de rubrique.

   a) Cliquez avec le bouton droit de la souris sur le dossier `Queue Managers\`*`QmgrName`*`\Queues` > **Nouveau** > **Définition de file d'attente à distance**.

   b) Nommez la définition `MyClientRemoteQueue`, puis cliquez sur **Suivant**.

   c) Entrez `MQTTExampleQueue` dans la zone **File d'attente éloignée**.

   d) Entrez `MyClient` dans la zone **Gestionnaire de files d'attente éloignées**.

   e) Entrez `SYSTEM.MQTT.TRANSMIT.QUEUE` dans la zone **File d'attente de transmission**, puis cliquez sur **Terminer**.
4. Placez un message de test sur `MyClientRemoteQueue`.

   a) Cliquez avec le bouton droit de la souris sur **MyClientRemoteQueue** > **Insérer un message de test...**

   b) Entrez `Hello queue!` dans la zone de données Message > **Insérer un message** > **Fermer**

   L' **historique client** enregistre un événement `Received` .
5. Supprimez `SYSTEM.MQTT.TRANSMIT.QUEUE` comme file d'attente de transmission par défaut.

   a) Clic droit sur `Queue Managers\`*`QmgrName`* `folder` > **Propriétés...**

   b) Cliquez sur **Communication** dans le navigateur.

   c) Retirez `SYSTEM.MQTT.TRANSMIT.QUEUE` de la zone **File d'attente de transmission par défaut**, puis cliquez sur **OK**.
6. Réexécutez l'étape «4», à la page 135.

   `MyClientRemoteQueue` est une définition de file d'attente à distance qui désigne explicitement la file d'attente de transmission. Vous n'avez pas besoin de définir la file d'attente de transmission par défaut pour envoyer un message à `MyClient`.

Que faire ensuite

Lorsque la file d'attente de transmission par défaut n'est plus définie sur SYSTEM.MQTT.TRANSMIT.QUEUE, l'utilitaire client MQTT ne peut pas créer d'abonnement sauf si un alias de gestionnaire de files d'attente est défini pour ClientIdentifier, MyClient. Restaurez la définition de la file d'attente de transmission par défaut sur SYSTEM.MQTT.TRANSMIT.QUEUE.

Windows Linux AIX Publication d'un message vers un client MQTT v3

Publication d'un message d'un client MQTT v3 vers un autre en utilisant ClientIdentifier comme nom de rubrique et IBM MQ comme courtier de publication/abonnement.

Avant de commencer

Exécutez la tâche «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130. Maintenez l'utilitaire client MQTT connecté.

Pourquoi et quand exécuter cette tâche

Cette tâche décrit deux opérations :

1. Abonnement à une rubrique dans un client MQTT et réception d'une publication d'un autre client MQTT.
2. Définition d'abonnements "point-à-point" en utilisant l'identificateur de client en tant que chaîne de rubrique.

Procédure

1. Supprimez les abonnements existants en déconnectant, puis en reconnectant l'utilitaire client MQTT.

   L'abonnement est supprimé, car (à moins que vous ne changiez les valeurs par défaut) l'utilitaire client MQTT se connecte avec une session propre. Voir Sessions propres.

   Pour faciliter cette tâche, entrez votre propre identifiant ClientIdentifier au lieu d'utiliser l'identifiant ClientIdentifier créé par l'utilitaire client MQTT.

   a) Cliquez sur l'option de **déconnexion** pour déconnecter l'utilitaire client MQTT du canal de télémétrie.

      L' **historique client** enregistre un événement Disconnected

   b) Remplacez l'identifiant **Identificateur de client** par MyClient.

   c) Cliquez sur **Connexion**.

   L' **historique client** enregistre un événement Connected
2. Créez un abonnement à la rubrique MyClient.

   MyClient est l'identificateur de client de ce client.

   a) Entrez MyClient dans la zone **Abonnement\Rubrique**, puis cliquez sur **S'abonner**.

   L' **historique client** enregistre un événement Subscribed .
3. Démarrez un autre utilitaire client MQTT.

   a) Ouvrez le dossier Queue Managers*QmgrName*\Telemetry\channels.

   b) Cliquez avec le bouton droit de la souris sur le canal **PlainText**, puis sélectionnez **Exécuter l'utilitaire client MQTT...**

   c) Cliquez sur **Connexion**.

   L' **historique client** enregistre un événement Connected
4. Publiez Hello MyClient! dans la rubrique MyClient.

a) Copiez la rubrique d'abonnement, `MyClient`, à partir de l'utilitaire client MQTT exécuté avec le `ClientIdentifier`, `MyClient`.

b) Collez `MyClient` dans le champ **Publication\Topic** de chaque instance de l'utilitaire client MQTT.

c) Entrez `Hello MyClient!` dans la zone **Publication \ message** .

d) Cliquez sur **Publication** dans les deux instances.

Résultats

L'**historique du client** dans l'utilitaire client MQTT avec le `ClientIdentifier`, `MyClient`, enregistre deux événements **reçus** et un événement **publié**. L'autre instance d'utilitaire client MQTT enregistre un événement **Publié**.

Si un seul événement **Reçu** est affiché, les raisons peuvent en être les suivantes :

1. La file d'attente de transmission par défaut du gestionnaire de files d'attente est-elle définie sur `SYSTEM.MQTT.TRANSMIT.QUEUE` ?
2. Avez-vous créé des alias de gestionnaire de files d'attente ou des définitions de files d'attente à distance faisant référence à `MyClient` pour les autres exercices ? Si vous avez un problème de configuration, supprimez toutes les ressources qui font référence à `MyClient`, telles qu'un alias de gestionnaire de files d'attente ou des files d'attente de transmission. Déconnectez les utilitaires client, arrêtez et redémarrez le service de télémétrie (MQXR).

Windows Linux AIX Envoi d'un message à une application IBM MQ depuis un client MQTT

Une application IBM MQ peut recevoir un message depuis un client MQTT version 3 en s'abonnant à une rubrique. Le client MQTT se connecte IBM MQ en utilisant un canal de télémétrie et envoie un message à l'application IBM MQ en publiant dans la même rubrique.

Réalisez la tâche «Publication d'un message vers IBM MQ depuis un client MQTT», à la page 137 pour apprendre à envoyer une publication d'un client MQTT à un abonnement défini IBM MQ.

Si la rubrique est mise en cluster ou répartie à l'aide d'une hiérarchie publication/abonnement, l'abonnement peut se trouver dans un gestionnaire de files d'attente différent de celui auquel le client MQTT est connecté.

Windows Linux AIX Publication d'un message vers IBM MQ depuis un client MQTT

Créez un abonnement à une rubrique à l'aide d'IBM MQ Explorer et publiez des données dans la rubrique à l'aide de l'utilitaire client MQTT.

Avant de commencer

Exécutez la tâche «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130. Maintenez l'utilitaire client MQTT connecté.

Pourquoi et quand exécuter cette tâche

La tâche illustre la publication d'un message avec un client MQTT et la réception de la publication à l'aide d'un abonnement durable non géré créé en utilisant IBM MQ Explorer.

Procédure

1. Créez un abonnement durable à la chaîne de rubrique `MQTT Example`.

   Exécutez les étapes suivantes pour créer la file d'attente et l'abonnement en utilisant IBM MQ Explorer.

a) Cliquez avec le bouton droit de la souris sur le dossier Queue Managers*QmgrName*\Queues dans IBM MQ Explorer> **Nouveau** > **File d'attente locale...**.

b) Entrez MQTTExampleQueue comme nom de la file d'attente, puis cliquez sur **Terminer**.

c) Cliquez avec le bouton droit de la souris sur le dossier Queue Managers*QmgrName*\Subscriptions dans IBM MQ Explorer> **Nouveau** > **Abonnement...**.

d) Entrez MQTTExampleSubscription comme nom de la file d'attente, puis cliquez sur **Suivant**.

e) Cliquez sur **Sélectionner...** > MQTTExampleTopic > **OK**.

Vous avez déjà créé la rubrique MQTTExampleTopic à l'étape «4», à la page 131 de «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130.

f) Entrez MQTTExampleQueue comme nom de destination, puis cliquez sur **Terminer**.

2. En tant qu'étape facultative, définissez la file d'attente à utiliser par un autre utilisateur, sans autorisation mqm.

Si vous paramétrez la configuration pour les utilisateurs disposant de moins de droits que mqm, vous devez attribuer les droits put et get à MQTTExampleQueue. L'accès à la rubrique et à la file d'attente de transmission a été configuré à la section «Publication d'un message dans l'utilitaire client MQTT depuis IBM MQ Explorer», à la page 130.

a) Autoriser un utilisateur à placer des éléments dans la file d'attente MQTTExampleQueue et à y accéder :

```
setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
```

3. Publiez Hello IBM MQ! dans la rubrique MQTT Example à l'aide de l'utilitaire client MQTT .

Si vous n'avez pas quitté l'utilitaire client MQTT connecté, cliquez avec le bouton droit de la souris sur le canal **Texte plan** > **Exécuter MQTT Utilitaire client...** > **Connecter**.

a) Entrez MQTT Example dans la zone **Publication\Rubrique**.

b) Entrez Hello IBM MQ! dans le champ **Publication\Message** > **Publier**.

4. Ouvrez le dossier Queue Managers*QmgrName*\Queues et recherchez MQTTExampleQueue.

La valeur entrée dans la zone **Longueur de la file d'attente en cours** est 1.

5. Cliquez avec le bouton droit de la souris sur MQTTExampleQueue > **Parcourir les messages ...** et examinez la publication.

Windows Linux AIX Applications de publication/abonnement MQTT

Utilisez la publication/l'abonnement basée sur une rubrique pour écrire des applications MQTT.

Lorsque le client MQTT est connecté, les publications transitent dans l'une ou l'autre direction entre le client et le serveur. Les publications sont envoyées à partir du client où les informations sont publiées côté client. Les publications sont reçues du côté du client lorsque le message est publié sur une rubrique qui correspond à une description créée par le client.

Le courtier de publication/abonnement IBM MQ gère les rubriques et les abonnements créés par les clients MQTT. Les rubriques créées par les clients MQTT partagent le même espace de sujet que les rubriques créées par les applications IBM MQ.

Les publications qui correspondent à la chaîne de rubrique dans un abonnement de client MQTT sont placées sur SYSTEM.MQTT.TRANSMIT.QUEUE avec le nom du gestionnaire de files d'attente éloignées défini sur ClientIdentifier du client. Le service de télémétrie (MQXR) transfère les publications au client qui a créé l'abonnement. Pour identifier le client, il utilise l'identificateur de client défini en tant que nom du gestionnaire de files d'attentes éloignées.

En règle générale, SYSTEM.MQTT.TRANSMIT.QUEUE doit être défini en tant que file d'attente de transmission par défaut. Il est possible, mais onéreux, de configurer MQTT pour ne pas utiliser la file d'attente de transmission par défaut. Voir Configuration des files d'attente réparties pour envoyer des messages aux clients MQTT.

Un client MQTT peut créer une session persistante. Voir «Sessions avec état et sans état MQTT», à la page 142. Les abonnements créés dans une session persistante sont durables. Les publications qui arrivent pour un client avec une session persistante sont stockées dans `SYSTEM.MQTT.TRANSMIT.QUEUE`, et transmises au client lors de la reconnexion.

Un client MQTT peut également publier et s'abonner à des publications conservées. Voir Publications conservées et clients MQTT. Un abonné à une publication conservée reçoit la dernière publication dans la rubrique. Il la reçoit lorsqu'il crée un abonnement ou lorsqu'il se reconnecte à la session précédente.

Windows Linux AIX Applications de télémétrie

Ecrivez des applications de télémétrie à l'aide de flux de messages IBM MQ ou IBM Integration Bus.

Utilisez JMS, MQI ou d'autres interfaces de programmation IBM MQ pour programmer des applications de télémétrie dans IBM MQ.

Le service de télémétrie (MQXR) convertit entre les messages MQTT v3 et les messages IBM MQ. Il crée des abonnements et des publications pour les clients MQTT et envoie les publications aux clients MQTT. Une publication est la charge d'un message MQTT v3. La charge comprend les en-têtes de message et un tableau d'octets au format `jms-bytes`. Le serveur de télémétrie mappe les en-têtes entre un message MQTT v3 et un message IBM MQ. Voir «Intégration de MQ Telemetry aux gestionnaires de files d'attente», à la page 139.

Utilisez les noeuds Publication, MQInput et JMSInput pour envoyer et recevoir des publications entre IBM Integration Bus et les clients MQTT.

A l'aide des flux de messages, vous pouvez intégrer la télémétrie à des sites Web à l'aide de HTTP et à d'autres applications à l'aide d'adaptateurs IBM MQ et WebSphere .

Windows Linux AIX Intégration de MQ Telemetry aux gestionnaires de files d'attente

Le client MQTT est intégré à IBM MQ comme application de publication/abonnement. Il peut publier ou s'abonner à des rubriques dans IBM MQ, en créant des rubriques, ou en utilisant des rubriques existantes. Il reçoit les publications d'IBM MQ comme résultat des clients MQTT, y compris de lui-même, ou des autres applications IBM MQ qui publient dans les rubriques de ses abonnements. Les règles sont appliquées pour décider des attributs d'une publication.

La plupart des attributs associés aux rubriques, publications, abonnements et messages fournis par IBM MQ ne sont pas pris en charge. Les sections «Client MQTT vers un courtier de publication/abonnement IBM MQ», à la page 140 et «IBM MQ vers un client MQTT», à la page 141 décrivent comment sont définis les attributs des publications. Les paramètres varient selon que la publication est envoyée ou provient d'un courtier de publication/abonnement IBM MQ.

Dans IBM MQ, les rubriques de publication/abonnement sont associées à des objets de rubrique d'administration. Les rubriques créées par les clients MQTT ne sont pas différentes. Lorsqu'un client MQTT crée une chaîne de rubrique pour une publication, le courtier de publication/abonnement IBM MQ l'associe à un objet de rubrique d'administration. Le courtier mappe la chaîne de rubrique dans la publication avec le plus proche parent d'objet de rubrique d'administration. Le mappage est le même que celui utilisé pour les applications IBM MQ. S'il n'existe aucune rubrique créée par l'utilisateur, la publication est mappée à `SYSTEM.BASE.TOPIC`. Les attributs qui sont appliqués à la publication sont dérivés de l'objet de rubrique.

Lorsqu'une application IBM MQ ou un administrateur crée un abonnement, l'abonnement est nommé. Répertorie les abonnements à l'aide de IBM MQ Explorer, ou à l'aide de commandes **runmqsc** ou PCF. Tous les abonnements client MQTT sont nommés. Ils reçoivent un nom au format suivant: *`ClientIdentifier:Topic name`*

Client MQTT vers un courtier de publication/abonnement IBM MQ

Un client MQTT a envoyé une publication à IBM MQ. Le service de télémétrie (MQXR) convertit la publication en message IBM MQ. Le message IBM MQ contient trois parties :

1. MQMD
2. RFH2
3. Message

Les propriétés de MQMD sont définies à leurs valeurs par défaut, sauf lorsqu'elles sont inscrites dans le Tableau 9, à la page 140.

Tableau 9. Paramètres pour les propriétés MQMD

Zone MQMD	Tapez	Valeur
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	Prend l'une des valeurs : MqttClient.ClientIdentifier MqttConnectOptions.UserName ID utilisateur défini par l'administrateur IBM MQ pour le canal de télémétrie.
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF (valeur par défaut d'IBM MQ, qui est différente de celle de JMS, qui est 4.)
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

L'en-tête RFH2 ne contient pas de dossier <msd> définissant le type du message JMS. Le service de télémétrie crée le message IBM MQ comme message par défaut JMS. Le type de message par défaut JMS est un message jms-bytes. Une application peut accéder à d'autres informations d'en-tête en tant que propriétés de message ; voir Propriétés des messages.

Les valeurs RFH2 sont définies comme indiqué dans le Tableau 10, à la page 140. La propriété Format est définie dans l'en-tête fixe RFH2 et les autres valeurs sont définies dans les dossiers RFH2.

Tableau 10. Paramètres pour les propriétés RFH2

Propriété RFH2	Type/Dossier	En-tête
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	Copiez MqttClient.ClientIdentifier avec une longueur de 1...23 octets.
QoS	mqtt/qos	Copiez QoS depuis le message entrant MQTT.
ID message	mqtt/msgid	Copiez l'ID de message du message entrant MQTT si QoS a la valeur 1 ou 2.
MQIsRetained	mqps/Ret	Définissez cette propriété si la publication MQTT d'origine a été envoyée avec la propriété RETAIN définie et que le message est reçu comme publication conservée.
MQTopicString	mqps/Top	Rubrique dans laquelle le message MQTT a été publié.

La charge dans une publication MQTT est mappée au contenu du message IBM MQ :

Tableau 11. Mappage du contenu d'une publication MQTT au contenu d'un message IBM MQ

Contenu du message	**Tapez**	**Contenu du message IBM MQ**
Tampon	MQBYTE *n*	Copiez les octets du message entrant MQTT. La longueur peut être zéro.

IBM MQ vers un client MQTT

Un client s'est abonné à une rubrique de publication. Une application IBM MQ a publié dans la rubrique, et une publication a été envoyée à l'abonné MQTT par le courtier de publication/abonnement IBM MQ. Ou bien, une application IBM MQ a envoyé un message non sollicité directement à un client MQTT. Le Tableau 12, à la page 141 explique comment les en-têtes de message fixes sont définis dans le message envoyé au client MQTT. Les autres données dans l'en-tête de message IBM MQ ou les autres en-têtes sont supprimés. Les données dans le message IBM MQ sont envoyées comme charge de message dans le message MQTT sans modification. Le message MQTT est envoyé au client MQTT par le service de télémétrie (MQXR).

Tableau 12. Définition des en-têtes de message fixes dans un message IBM MQ envoyé au client MQTT

MQTTZone	**Tapez**	**Valeur**
DUP	Booléen	Défini si QoS = 1 ou 2, et que le message a été envoyé à ce client lors d'une transmission précédente, et que ce message n'a pas reçu d'accusé de réception après un certain laps de temps.
QoS	entier	La manière dont la valeur de QoS dans la publication sortante du courtier de publication/abonnement IBM MQ est définie dépend de la publication entrante. Elle varie selon que la publication entrante a été envoyée par un client MQTT ou une application IBM MQ. **MQTT** Valeur la plus faible de qualité de service QoS dans la publication entrante et dans la QoS demandé par l'abonné. **IBM MQ** Valeur la plus faible de QoS dérivée de la publication entrante : MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 et la QoS demandée par l'abonné. Si le message est envoyé au client sans un abonnement, QoS prend la valeur par défaut 2. Un client peut modifier cette valeur en s'abonnant à DEFAULT.QoS avec une QoS différente.
RETAIN	Booléen	Défini si la propriété retained a été définie pour la publication.

Le Tableau 13, à la page 142 explique comment les en-têtes de message variables sont définis dans le message MQTT envoyé au client MQTT.

Tableau 13. Définition des propriétés d'en-tête variable MQTT dans un message MQTT envoyé au client MQTT		
MQTTZone	**Tapez**	**Valeur**
Topic name	Chaîne	Chaîne de rubrique avec laquelle le message a été publié.
Message ID	Chaîne	Deux derniers octets de la propriété MQMD.MsgId de la publication lorsqu'elle est placée dans SYSTEM.MQTT.TRANSMIT.QUEUE.
Payload	octet[]	Copie directe des octets de la publication entrante dans le courtier de publication/abonnement. La longueur peut être zéro.

Windows Linux AIX Sessions avec état et sans état MQTT

Les clients MQTT peuvent créer une session avec état avec le gestionnaire de files d'attente. Lorsqu'un client MQTT avec état se déconnecte, le gestionnaire de files d'attente conserve les abonnements créés par le client ainsi que les messages en cours. Lorsque le client se reconnecte, il le message en cours. Il envoie les messages mis en file d'attente pour être distribués et reçoit les messages publiés pour ses abonnements pendant qu'il était déconnecté.

Lorsqu'un client MQTT se connecte à un canal de télémétrie, il démarre une nouvelle session ou relance une session. Dans le cas d'une nouvelle session, il n'existe aucun message en attente n'ayant pas reçu d'accusé de réception, aucun abonnement et aucune publication en attente de distribution. Lorsqu'un client se connecte, il indique si une session propre doit être démarrée ou s'il convient de relancer une session existante ; voir Sessions propres.

Si le client relance une session existante, celle-ci se poursuit comme si la connexion n'avait pas été interrompue. Les publications en attente de distribution sont envoyées au client et les réacheminements de messages qui n'avaient pas été validés sont terminés. Lorsqu'un client dans une session persistante se déconnecte du service de télémétrie (MQXR), les abonnements créés par le client sont conservés. Les publications des abonnements sont envoyées au client lorsqu'il se reconnecte. Si ce dernier se reconnecte sans relancer la précédente session, les publications sont supprimées par le service de télémétrie (MQXR).

Les informations d'état de session sont sauvegardées dans le gestionnaire de files d'attente SYSTEM.MQTT.PERSISTENT.STATE.

L'administrateur IBM MQ peut déconnecter et purger la session.

Windows Linux AIX Lorsqu'un client MQTT n'est pas connecté

Lorsqu'un client MQTT n'est pas connecté, le gestionnaire de files d'attente peut continuer à recevoir des publications en son nom. Elles sont réacheminées au client lorsque ce dernier se reconnecte. Un client peut créer une publication "Last will and testament" (Dernière volonté et testament) qui sera publiée par le gestionnaire de files d'attente au nom du client si ce dernier se déconnecte de manière inattendue.

Si vous voulez être prévenu lorsque le client se déconnecte de façon inattendue, vous pouvez enregistrer une publication "Last will and testament". Voir Publication Last will and testament (Dernière volonté et testament). Cette publication est envoyée au service de télémétrie (MQXR) s'il détecte que la connexion avec le client a été interrompue sans que le client en ait fait la demande.

Un client peut publier une publication conservée à tout moment ; voir Publications conservées et clients MQTT. Un nouvel abonnement à une rubrique peut nécessiter l'envoi d'une publication conservée associée à cette publication. Si vous créez la publication "last will and testament" en tant que publication conservée, vous pouvez l'utiliser pour contrôler le statut du client.

Par exemple, le client diffuse une publication conservée, lorsqu'il se connecte, indiquant qu'elle est disponible. Au même moment, il crée une publication conservée "last will and testament" qui annonce sa disponibilité. De plus, juste avant d'effectuer une déconnexion planifiée, il publie sa non disponibilité

en tant que publication conservée. Pour trouver si le client est disponible, vous devez vous abonner à la rubrique de la publication conservée. Vous devez toujours recevoir l'une des trois publications.

Si le client doit recevoir des messages publiés lorsqu'il est déconnecté, reconnectez le client à la session précédente ; voir «Sessions avec état et sans état MQTT», à la page 142. Ses abonnements sont actifs jusqu'à leur suppression ou jusqu'à ce que le client crée une session propre.

Windows Linux AIX Couplage non contraignant entre les clients MQTT et les applications IBM MQn

Le flux de publications entre les clients MQTT et les applications IBM MQ est couplé de façon non contraignante. Les publications peuvent provenir d'un client MQTT ou d'une application IBM MQ et dans un ordre non défini. Les diffuseurs de publications et les abonnés sont couplés de façon non contraignante. Ils interagissent de manière indirecte via les publications et les abonnements. Vous pouvez aussi envoyer directement des messages à un client MQTT depuis une application IBM MQ.

Les clients MQTT et les applications IBM MQ sont couplés de façon non contraignante dans deux sens :

1. Les diffuseurs de publication et les abonnés sont couplés de façon non contraignantes via l'association d'une publication et d'un abonnement avec une rubrique. Les diffuseurs de publication et les abonnés ne connaissent généralement pas l'adresse ou l'identité de l'autre source de publication ou d'abonnement.
2. Les clients MQTT publient, reçoivent et s'abonnent à des publications, et ils traitent les accusés de réception de distribution sur des unités d'exécution distincte.

Une application client MQTT n'attend pas la distribution d'une publication. Elle transmet un message au client MQTT, puis l'application continue dans sa propre unité d'exécution. Un jeton de distribution est utilisé pour synchroniser l'application avec la distribution d'une publication ; voir Jetons de distribution.

Une fois que le message a été transmis au client MQTT, l'application a le choix d'attendre le jeton de distribution. Au lieu d'attendre, le client peut fournir une méthode de rappel qui est appelée lorsque la publication est distribuée à IBM MQ. L'application peut également ignorer le jeton de distribution.

En fonction de la qualité de service associée au message, le jeton de distribution est renvoyé immédiatement à la méthode de rappel ou probablement après un délai très important. Le jeton de distribution peut même être renvoyé après déconnexion et reconnexion du client. Si la qualité de service est de type *autonome après diffusion*, le jeton de distribution est renvoyé immédiatement. Dans les deux autres cas, le jeton de distribution est renvoyé uniquement lorsque le client reçoit un accusé de réception indiquant que la publication a été envoyée aux abonnés.

Les publications envoyées au client MQTT à la suite d'un abonnement du client sont distribuées à la méthode de rappel `messageArrived`. `messageArrived` s'exécute sur une unité d'exécution différente de l'application principale.

Envoi de messages directement à un client MQTT

Vous pouvez envoyer un message à un client MQTT de l'une des deux façons suivantes.

1. Une application IBM MQ peut envoyer un message directement à un client MQTT sans abonnement. Voir Envoi direct d'un message à un client.
2. Vous pouvez également utiliser la convention de dénomination de votre `identificateur de client`. Demandez aux abonnés MQTT de créer des abonnements en utilisant leur identifiant `ClientIdentifier` unique comme rubrique. Diffusez la publication pour l'*identificateur de client*. La publication est envoyée au client qui s'est abonné à la rubrique *Identificateur de client*. Cette méthode permet d'envoyer une publication à un abonné MQTT.

Windows Linux AIX Sécurité MQ Telemetry

La sécurisation des dispositifs de télémétrie peut être cruciale étant donné que les dispositifs sont susceptibles d'être portables et utilisés dans des endroits qui ne sont pas correctement contrôlés. Vous

pouvez utiliser un réseau privé virtuel (VPN) pour sécuriser la connexion du périphérique MQTT vers le service de télémétrie (MQXR). MQ Telemetry fournit deux autres mécanismes de sécurité, TLS et JAAS.

TLS est principalement utilisé pour chiffrer les communications entre le dispositif et le canal de télémétrie, et pour s'assurer que le dispositif est connecté au serveur approprié ; voir Authentification du canal de télémétrie à l'aide de TLS. Vous pouvez également utiliser TLS pour vérifier que le dispositif client peut se connecter au serveur. Voir Authentification du client MQTT.

JAAS est principalement utilisé pour vérifier que l'utilisateur du dispositif peut utiliser une application serveur. Voir Authentification du client MQTT en utilisant un mot de passe. JAAS peut être utilisé avec LDAP pour vérifier un mot de passe à l'aide d'un répertoire de connexion unique.

TLS et JAAS peuvent être utilisés conjointement pour offrir deux facteurs d'authentification. Vous pouvez restreindre les chiffrements utilisés par TLS aux chiffrements respectant les normes FIPS.

Avec au moins dix ou des milliers d'utilisateurs, il n'est pas toujours pratique d'accorder des profils de sécurité individuels. Il n'est pas non plus pratique d'utiliser des profils pour autoriser les utilisateurs individuels à accéder aux objets IBM MQ. Regroupez plutôt les utilisateurs dans des classes pour l'autorisation de publication et d'abonnement aux rubriques et l'envoi de publications aux clients.

Configurez chaque canal de télémétrie pour mapper les clients à des ID utilisateur client communs. Servez-vous d'un ID utilisateur commun pour tout client qui se connecte sur un canal spécifique. Voir Identité et autorisation du client MQTT.

L'autorisation de groupes d'utilisateurs n'a pas d'incidence sur l'authentification de chaque individu. Chaque utilisateur individuel peut être authentifié, côté client ou serveur avec son `Nom d'utilisateur` et son `Mot de passe`, puis être authentifié sur le serveur à l'aide d'un ID utilisateur commun.

Windows Linux AIX Globalisation MQ Telemetry

La charge de message dans le protocole MQTT v3 est codée comme un tableau d'octets. Généralement, les applications qui gèrent du texte créent la charge du message au format UTF-8. Le canal de télémétrie décrit cette charge de message en UTF-8, mais n'effectue aucune conversion de page de code. La chaîne de rubrique doit être au format UTF-8.

L'application est responsable de la conversion des données alphabétiques dans la page de code appropriée et des données numériques dans le codage numérique correct.

Le client MQTT Java dispose d'une méthode `MqttMessage.toString` pratique. La méthode traite la charge de message comme étant codée dans le jeu de caractères par défaut de la plateforme locale qui est généralement UTF-8. Elle convertit la charge en chaîne Java. Java dispose d'une méthode Chaîne `getBytes` qui convertit une chaîne en tableau d'octets codé à l'aide du jeu de caractères par défaut de la plateforme locale. Deux programmes MQTT Java qui échangent du texte dans la charge de message, entre des plateformes ayant le même jeu de caractères par défaut, exécutent facilement et efficacement cette opération au format UTF-8.

Si le jeu de caractères par défaut de l'une des plateformes n'est pas UTF-8, les applications doivent alors établir une convention pour échanger les messages. Par exemple, le diffuseur de publications spécifie une conversion à partir d'une chaîne vers UTF-8 à l'aide de la méthode `getBytes("UTF8")`. Pour recevoir le texte d'un message, l'abonné part du principe que le message est codé avec le jeu de caractères UTF-8.

Le service de télémétrie (MQXR) décrit le codage de toutes les publications entrantes provenant des messages des clients MQTT comme étant au format UTF-8. Il définit `MQMD.CodedCharSetId` sur UTF-8et `RFH2.CodedCharSetId` sur `MQCCSI_INHERIT` ; voir «Intégration de MQ Telemetry aux gestionnaires de files d'attente», à la page 139. Le format de la publication prend la valeur `MQFMT_NONE`, ce qui signifie qu'aucune conversion ne peut être effectuée par les canaux ou par `MQGET`.

Windows Linux AIX Performance et évolutivité de MQ Telemetry

Tenez compte des éléments suivants lorsque vous gérez un grand nombre de clients et que vous voulez améliorer l'évolutivité de MQ Telemetry.

Planification de la capacité

Pour des informations sur les rapports de performances pour MQ Telemetry, voir MQ Performance documents.

Connexions

Les coûts induits par la connexion comprennent :

- le coût de configuration d'une connexion en soi, en terme d'utilisation et de temps du processeur ;
- les coûts liés au réseau ;
- la mémoire utilisée lorsqu'une connexion est ouverte et qu'elle n'est pas utilisée.

Une charge supplémentaire est à prévoir lorsque les clients restent connectés. Si une connexion est maintenue ouverte, les flux TCP/IP et les messages MQTT utilisent le réseau pour vérifier que la connexion est toujours présente. De plus, de la mémoire est utilisée au niveau du serveur pour chaque connexion client restée ouverte.

Si vous envoyez des messages à une fréquence supérieure à un message par minute, laissez votre connexion ouverte pour éviter le coût lié à l'établissement d'une nouvelle connexion. Si vous envoyez des messages à une fréquence inférieure à un message toutes les 10 ou 15 minutes, pensez à mettre fin à votre connexion afin d'éviter le coût engendré par son maintien. Il peut être préférable de conserver une connexion TLS ouverte, mais inactive, pendant de longues périodes car la configuration d'une telle connexion est plus onéreuse.

De plus, tenez compte de la capacité du client. S'il existe une fonction de stockage et retransmission sur le client, vous pouvez envoyer les messages par lots et interrompre la connexion entre l'envoi des lots. Toutefois, si le client est déconnecté, il ne peut pas recevoir de messages provenant du serveur. Vous prendrez donc votre décision en fonction de l'utilisation que vous faites de votre application.

Si, dans votre système, il existe un client qui envoie beaucoup de messages, par exemple des transferts de fichiers, alors n'attendez pas une réponse du serveur par message. Envoyez tous les messages et vérifiez à la fin de l'opération qu'ils ont tous été reçus. Vous pouvez également utiliser la qualité de service (QoS).

Vous pouvez modifier la qualité de service par message, en transmettant les messages non importants avec une qualité de service de 0 et les messages importants avec une qualité de service de 2. Le débit des messages peut être environ deux fois plus élevé avec une qualité de service de 0 qu'avec une qualité de service de 2.

Conventions de dénomination

Si votre application a été conçue pour un grand nombre de clients, implémentez une convention de dénomination efficace. Pour mapper chaque client à l'`identificateur de client` correct, choisissez un `identificateur de client` significatif. Une bonne convention de dénomination permet à l'administrateur de déterminer plus facilement les clients qui sont en cours d'exécution. Elle aide l'administrateur à filtrer la longue liste des clients dans IBM MQ Explorer, et facilite l'identification des problèmes. Voir Identificateur de client.

Débit

La longueur des noms de rubrique affecte le nombre d'octets transmis via le réseau. Lorsque vous diffusez une publication ou que vous vous abonnez à un message, le nombre d'octets dans un message peut s'avérer important. Par conséquent, limitez le nombre de caractères du nom de rubrique. Lorsqu'un client MQTT s'abonne à une rubrique, IBM MQ lui attribue un nom au format suivant :

```
ClientIdentifier: TopicName
```

Pour afficher tous les abonnements d'un client MQTT, vous pouvez utiliser la commande IBM MQ MQSC **DISPLAY** :

```
DISPLAY SUB(' ClientID1:*')
```

Définition des ressources dans IBM MQ à utiliser par les clients MQTT

Un client MQTT se connecte à un gestionnaire de files d'attente éloignées IBM MQ. Il existe deux méthodes de base pour qu'une application IBM MQ envoie des messages à un client MQTT : définissez la file d'attente de transmission par défaut sur SYSTEM.MQTT.TRANSMIT.QUEUE ou utilisez des alias de gestionnaire de files d'attente. S'il existe un grand nombre de clients MQTT, définissez la file d'attente de transmission par défaut d'un gestionnaire de files d'attente. Lorsque vous utilisez le paramètre de file d'attente de transmission par défaut, les tâches d'administration sont plus simples. Voir Configuration des files d'attente réparties pour envoyer des messages aux clients MQTT.

Amélioration de l'évolutivité en évitant les abonnements

Lorsqu'un client MQTT version 3 s'abonne à une rubrique, un abonnement est créé par le service de télémétrie (MQXR) dans IBM MQ. L'abonnement achemine les publications du client vers SYSTEM.MQTT.TRANSMIT.QUEUE. Le nom du gestionnaire de files d'attente éloignées dans l'en-tête de transmission de chaque publication correspond à l'identificateur `ClientIdentifier` du client MQTT qui a effectué l'abonnement. S'il existe de nombreux clients, chacun effectuant ses propres abonnements, de nombreux abonnements proxy sont gérés dans le cluster ou la hiérarchie de publication/abonnement IBM MQ. Pour plus d'informations sur l'utilisation d'une solution point-à-point plutôt que d'une solution de publication/abonnement, voir Envoi d'un message à un client directement.

Gestion d'un grand nombre de clients

Si vous souhaitez qu'un grand nombre de clients soient connectés simultanément, vous devez accroître la taille de la mémoire disponible pour le service de télémétrie (MQXR) en définissant les paramètres de la machine virtuelle Java **-Xms** et **-Xmx**. Effectuez les étapes suivantes :

1. Recherchez le fichier `java.properties` dans le répertoire de configuration du service de télémétrie ; voir Répertoire de configuration du service de télémétrie (MQXR) sous Windows ou Répertoire de configuration du service de télémétrie sous Linux.
2. Suivez les instructions indiquées dans le fichier. Un segment de mémoire de 1 Go est suffisant pour 50 000 clients connectés simultanément.

   ```
   # Heap sizing options - uncomment the following lines to set the heap to 1G
   #-Xmx1024m
   #-Xms1024m
   ```
3. Ajoutez d'autres arguments de ligne de commande à transmettre à la JVM exécutant le service de télémétrie (MQXR) dans le fichier `java.properties` ; voir Transmission des paramètres JVM au service de télémétrie (MQXR).

Pour augmenter le nombre de descripteurs de fichiers ouverts sous Linux, ajoutez les lignes suivantes à `/etc/security/limits.conf/` et connectez-vous à nouveau.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

Chaque connecteur nécessite un descripteur de fichier. Le service de télémétrie requiert des descripteurs de fichier supplémentaires. Ce nombre doit donc être supérieur au nombre de connecteurs ouverts requis.

Le gestionnaire de files d'attente utilise un descripteur d'objet pour chaque abonnement non durable. Pour prendre en charge plusieurs abonnements non durables actifs, augmentez le nombre maximal de descripteurs actifs dans le gestionnaire de files d'attente. Par exemple :

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

Figure 48. Modification du nombre maximum de descripteurs sur Windows

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

Figure 49. Modification du nombre maximum de descripteurs sur Linux

Autres considérations

Lorsque vous planifiez la configuration requise, tenez compte de la durée nécessaire au redémarrage du système. Le temps planifié d'indisponibilité peut avoir des implications sur le nombre de messages en attente de traitement. Configurez le système de telle sorte que les messages puissent être traités dans un laps de temps acceptable. Vérifiez l'espace disque, la mémoire et la puissance du processeur. Dans certaines applications client, il peut être possible de supprimer ces messages lorsque le client se reconnecte. Pour les supprimer, définissez `CleanSession` dans les paramètres de connexion du client (voir Sessions propres). Vous pouvez également publier et abonner en utilisant la meilleure qualité de service, `0`, dans un client MQTT. Voir Qualité de service. Utilisez des messages `non persistants` pour envoyer des messages depuis IBM MQ. Les messages associés à ces qualités de service ne sont pas récupérés lorsque le système ou la connexion redémarre.

Windows Linux AIX Périphériques compatibles avec MQ Telemetry

Les clients MQTT peuvent s'exécuter sur un large éventail de périphériques, allant des détecteurs et actionneurs aux dispositifs portatifs, en passant par les systèmes embarqués dans des véhicules.

Les clients MQTT sont petits et s'exécutent sur des périphériques avec une faible capacité de mémoire et ayant une puissance de traitement limitée. Le MQTT protocol est fiable et contient des petites en-têtes qui sont adaptés aux réseaux à faible bande passante, coûteux et dont la disponibilité est intermittente.

MQ Telemetry communique avec les périphériques de télémétrie via les applications client MQTT. Ces applications utilisent les ressources suivantes qui implémentent toutes le protocole MQTT version 3 :

- Bibliothèques client suivantes :
  - Le *MQTT client for Java*, qui est utilisé pour générer des applications natives pour (par exemple) les périphériques Android, OS X, Linux ou Windows. Les applications qui utilisent cette bibliothèque client peuvent s'exécuter sur toutes les variantes de Java de la plus petite configuration CLDC (Connected Limited Device Configuration)/MIDP (Mobile Information Device Profile) à CDC (Connected Device Configuration)/Foundation, J2SE (Java Platform, Standard Edition) et J2EE (Java Platform, Enterprise Edition). La bibliothèque de classe personnalisée IBM jclRM est également prise en charge. La plateforme Java ME est généralement utilisée sur les petits périphériques, tels que les actionneurs, les détecteurs, les téléphones mobiles et d'autres services intégrés. La plateforme Java SE est généralement installée sur des appareils haut de gamme et intégrés, tels que les ordinateurs de bureau et les serveurs.
  - *MQTT client for C*, qui est utilisé pour construire des applications natives pour (par exemple) les unités iOS, OS X, Linux ou Windows. Cette bibliothèque client fournit une implémentation de référence C avec le client natif préconfiguré pour les systèmes Windows et Linux. L'implémentation de référence C permet à MQTT d'être porté sur une large gamme de périphériques et de plateformes. Certains systèmes Windows sur Intel, notamment Windows 7, RedHat, Ubuntu, et certains systèmes Linux sur les plateformes ARM, tels qu'Eurotech Viper, implémentent des versions de Linux qui exécutent le client C, mais IBM ne fournit pas de support pour les plateformes. Vous devez reproduire les problèmes avec le client sur une plateforme prise en charge si vous envisagez d'appeler votre centre de support IBM.
  - Le *MQTT client for Java*, qui est utilisé pour la génération d'applications basées sur un navigateur.

Les bibliothèques client MQTT sont disponibles gratuitement sur Eclipse Paho et MQTT.org. Voir IBM MQ Telemetry Transport sample programs.

Sécurité dans IBM MQ

Dans IBM MQ, il existe plusieurs méthodes de sécurité : interface de service d'autorisation, exits de canal tiers ou écrits par l'utilisateur, sécurité de canal à l'aide du protocole TLS (Transport Layer Security), enregistrements d'authentification de canal et sécurité des messages.

Interface de service d'autorisation

L'autorisation de l'utilisation d'appels MQI, de commandes et d'accès aux objets est fournie par le **gestionnaire des droits d'accès aux objets** (OAM), qui est activé par défaut. L'accès aux entités IBM MQ est contrôlé via des groupes d'utilisateurs IBM MQ et le gestionnaire OAM. Les administrateurs peuvent utiliser une interface de ligne de commande pour octroyer ou révoquer des autorisations, selon les besoins.

Pour plus d'informations sur la création de composants de service d'autorisation, voir Configuration de la sécurité sur les systèmes AIX, Linux, and Windows.

Exits de canal tiers ou écrits par l'utilisateur

Les canaux peuvent utiliser des exits de canal tiers ou écrits par l'utilisateur. Pour plus d'informations, voir Programmes d'exit de canal pour les canaux de messagerie.

Sécurité de canal à l'aide de TLS

Le protocole TLS (Transport Layer Security) fournit une sécurité de canal standard assurant la protection contre les interceptions électroniques, les falsifications et l'usurpation.

TLS utilise des techniques de clé publique et symétrique pour assurer la confidentialité et l'intégrité des messages et l'authentification mutuelle.

Pour plus de détails sur la sécurité dans IBM MQ, y compris des informations détaillées sur TLS, voir Sécurisation. Pour une présentation de TLS, notamment des pointeurs des commandes décrites dans la présente section, voir Protocoles de sécurité cryptographiques TLS.

Enregistrements d'authentification de canal

Les enregistrements d'authentification de canal permettent d'exercer un contrôle précis sur l'accès octroyé aux systèmes de connexion au niveau d'un canal. Pour plus d'informations, voir Enregistrements d'authentification de canal.

Sécurité des messages

Utilisez Advanced Message Security, qui est un composant sous licence d'IBM MQ installé séparément, pour fournir une protection cryptographique aux messages envoyés et reçus via IBM MQ. Voir Advanced Message Security.

Tâches associées

Sécurisation

Planification de la sécurité

Prise en charge de TLS par le client géré IBM MQ.NET

Le client entièrement géré IBM MQ.NET fournit la prise en charge TLS (Transport Layer Security) basée sur le kit Microsoft.NET SSLStreams. Il est différent des autres clients IBM MQ , qui sont basés sur IBM Global Security Kit (GSKit).

Vous pouvez développer des applications IBM MQ.NET pour les exécuter en mode géré ou non géré.

- En mode géré, les applications .NET fonctionnent dans .NET CLR (Common Language Runtime) sans aucun appel interplateforme, tel que l'appel C MQI.

- En mode non géré, C MQI est appelé pour les opérations MQI sous-jacentes. Généralement, l'interface en mode non géré comprend des classes d'encapsuleur .NET en plus de C MQI.

Le client géré IBM MQ.NET utilise les bibliothèques Microsoft.NET Framework pour implémenter les protocoles de connexion sécurisée TLS. La classe System.NET.Security.SSLStream de Microsoft est utilisée pour implémenter la sécurité (TLS) dans IBM MQ.NET.

Le mode client IBM MQ.NET non géré prend déjà en charge la fonction TLS, qui est basée sur C MQI (et GSKit). Les opérations TLS sont donc traitées par C MQI. Dans ce cas, GSKit implémente les protocoles de connexion sécurisée TLS.

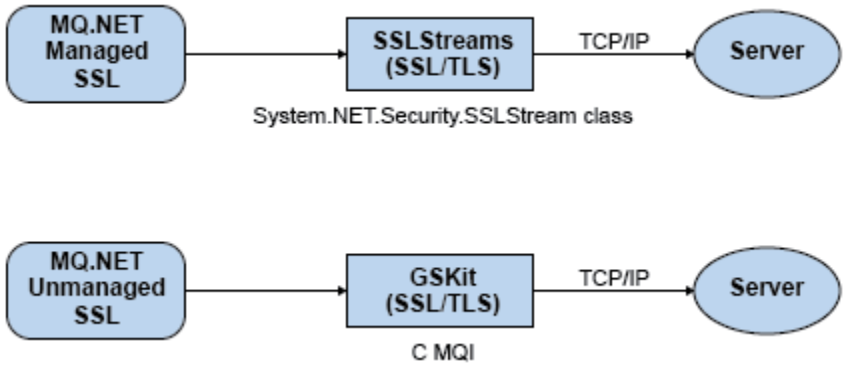


Figure 50. Comparaison de SSL/TLS géré et non géré par IBM MQ.NET

Le tableau suivant présente de manière synthétique les différences entre les implémentations gérées et non gérées :

Tableau 14. Différences entres les implémentations gérées et non gérées

Mode	Protocoles	Implémentation	Commentaires
SSL géré IBM MQ.NET	TLS	Classe System.NET.Security.SSLStream La classe SSLStream joue le rôle de flux via un socket TCP connecté	TLS 1.0 TLS 1.2 (avec Microsoft.NET Framework version 4.5 uniquement)
SSL non géré IBM MQ.NET	TLS	GSKit et C-MQI	Protocoles de connexion sécurisée TLS

Concepts associés

Support Secure Sockets Layer (SSL) et Transport Layer Security (TLS) pour .NET

IBM MQ MQI clients

Un IBM MQ MQI client est un composant du produit IBM MQ qui peut être installé sur un système sur lequel aucun gestionnaire de files d'attente n'est exécuté.

Un *client* IBM MQ MQI est un composant qui permet à une application exécutée sur un système d'émettre des appels MQI à un gestionnaire de files d'attente exécuté sur un autre système. Le résultat de l'appel est renvoyé au client, qui le renvoie à l'application.

En utilisant un IBM MQ MQI client, une application exécutée sur le même système que le client peut se connecter à un gestionnaire de files d'attente en cours d'exécution sur un autre système. L'application peut émettre des appels MQI à ce gestionnaire de files d'attente. Une application de ce type est appelée application IBM MQ MQI client et le gestionnaire de files d'attente est appelé *gestionnaire de files d'attente du serveur*..

Un IBM MQ serveur est un gestionnaire de files d'attente qui fournit des services de mise en file d'attente à un ou plusieurs clients. Tous les objets IBM MQ, par exemple les files d'attente, existent uniquement sur le poste du gestionnaire de files d'attente (le poste serveur IBM MQ) et non sur le client. Un serveur IBM MQ peut également prendre en charge les applications IBM MQ.

La différence entre un serveur IBM MQ et un gestionnaire de files d'attente ordinaire réside dans le fait qu'un serveur possède une liaison de communication dédiée avec chaque client. Pour plus d'informations sur la création de canaux pour les clients et les serveurs, voir Configuration de la mise en file d'attente répartie.

Une application IBM MQ MQI client et un gestionnaire de files d'attente du serveur communiquent ensemble à l'aide d'un *canal MQI*. Ce dernier démarre lorsque l'application client émet un appel **MQCONN** ou **MQCONNX** pour se connecter au gestionnaire de files d'attente et s'arrête lorsque l'application client émet un appel **MQDISC** pour se déconnecter du gestionnaire de files d'attente. Les paramètres d'entrée d'un appel MQI sont transmis dans une direction sur un canal MQI et les paramètres de sortie dans la direction opposée.

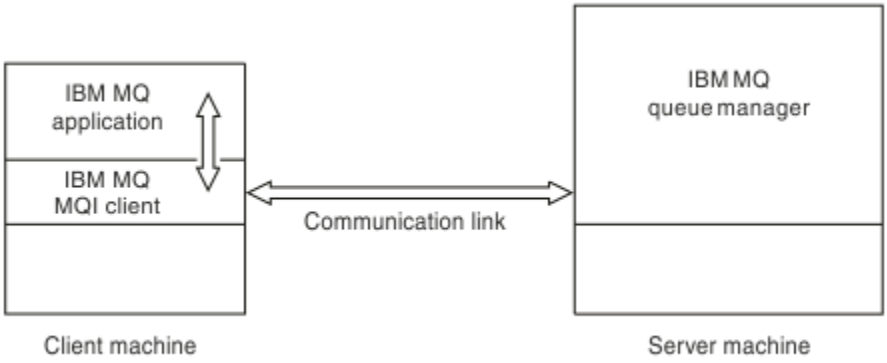


Figure 51. Liaison entre un client et un serveur

Les plateformes ci-dessous peuvent être utilisées. Les combinaisons sont fonction du produit IBM MQ que vous utilisez et sont décrites dans la rubrique «Plateformes prises en charge pour les clients IBM MQ», à la page 152.

IBM MQ MQI client	**IBM MQServeur**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

Le canal MQI est accessible aux applications exécutées sur la plateforme client ; les files d'attente et autres objets IBM MQ sont conservés dans un gestionnaire de files d'attente que vous avez installé sur un serveur.

Une application que vous souhaitez exécuter dans l'environnement IBM MQ MQI client doit auparavant être connectée à la bibliothèque client appropriée. Lorsque l'application émet un appel MQI, le IBM MQ MQI client achemine la demande vers un gestionnaire de files d'attente dans lequel elle est traitée et à partir duquel une réponse est renvoyée au IBM MQ MQI client.

La liaison entre l'application et le IBM MQ MQI client est établie de façon dynamique au moment de l'exécution.

Vous pouvez également développer des applications client à l'aide de IBM MQ classes for .NET, IBM MQ classes for Java ou IBM MQ classes for Java Message Service (JMS). Vous pouvez utiliser les clients Java et JMS sur les plateformes suivantes :

- IBM i

- 

AIX
- 

Linux
- 

Windows

L'utilisation de Java et JMS n'est pas décrite dans la présente rubrique. Pour plus de détails sur l'installation, la configuration et l'utilisation d'IBM MQ classes for Java et d'IBM MQ classes for JMS, voir Utilisation d'IBM MQ classes for Java et Utilisation d'IBM MQ classes for JMS.

Applications IBM MQ dans un environnement client-serveur

Une fois liées à un serveur, les applications IBM MQ client peuvent émettre la plupart des appels MQI de la même façon que les applications locales. L'application client émet un appel MQCONN pour se connecter à un gestionnaire de files d'attente spécifié. Tous les autres appels MQI spécifiant le descripteur de connexion renvoyé par la demande de connexion sont ensuite traités par ce gestionnaire de files d'attente.

Vous devez connecter vos applications aux bibliothèques client appropriées. Voir Génération d'applications pour les IBM MQ MQI clients.

Concepts associés

«Pourquoi utiliser des clients IBM MQ ?», à la page 151
L'utilisation de clients IBM MQ est un moyen efficace d'implémenter la messagerie et la mise en file d'attente IBM MQ.

«A quoi correspond un client transactionnel étendu ?», à la page 153
Un client transactionnel étendu IBM MQ peut mettre à jour les ressources gérées par un autre gestionnaire de ressources, sous le contrôle d'un gestionnaire de transactions externe.

«Mode de connexion du client au serveur», à la page 154
Un client se connecte à un serveur à l'aide de MQCONN ou de MQCONNX, et communique via un canal.

«Gestion et prise en charge des transactions», à la page 156
Présentation de la gestion des transactions et description de la prise en charge des transactions par IBM MQ.

«Extension des fonctions du gestionnaire de files d'attente», à la page 157
Vous pouvez étendre les fonctions du gestionnaire de files d'attente à l'aide d'exits utilisateur, d'exits API ou de services optionnels.

Information associée

Comment configurer un IBM MQ MQI client

Pourquoi utiliser des clients IBM MQ ?

L'utilisation de clients IBM MQ est un moyen efficace d'implémenter la messagerie et la mise en file d'attente IBM MQ.

Vous pouvez avoir une application qui utilise l'interface MQI exécutée sur une machine et le gestionnaire de files d'attente sur une machine différente (physique ou virtuelle). Voici les avantages :

- Mise en oeuvre complète d'IBM MQ sur la machine client.
- Configuration matérielle requise sur le système client réduite.
- Exigences de l'administration système réduites.
- Une application IBM MQ exécutée sur un client peut se connecter à plusieurs gestionnaires de files d'attente sur des systèmes différents.
- Des canaux secondaires utilisant des protocoles de transmission différents peuvent être utilisés.

Plateformes prises en charge pour les clients IBM MQ

IBM MQ sur toutes les plateformes serveur prises en charge accepte les connexions des IBM MQ MQI clients sur un certain nombre de plateformes.

IBM MQ installé sous la forme d'un *serveur et produit de base* sur toutes les plateformes serveur prises en charge peut accepter les connexions des IBM MQ MQI clients sur les plateformes suivantes :

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

L'identificateur du jeu de caractères codés (CCSID) et le protocole de communication peuvent varier selon les connexions.

Quelles applications s'exécutent sur un IBM MQ MQI client ?

L'interface MQI complète est prise en charge dans l'environnement client. Vous pouvez configurer pratiquement toutes les applications IBM MQ en vue de leur exécution sur un système IBM MQ MQI client en liant l'application sur le IBM MQ MQI client à la bibliothèque MQIC au lieu de la lier à la bibliothèque MQI. Voici les exceptions :

- MQGET avec signal
- Une application nécessitant la coordination de point de synchronisation avec d'autres gestionnaires de ressources doit utiliser un client transactionnel étendu

Si la lecture anticipée est activée pour améliorer les performances de la messagerie non persistante, les options MQGET ne sont pas toutes disponibles. Le tableau présente les options admises et indique si elles peuvent être modifiées d'un appel MQGET à l'autre.

Tableau 15. Options MQGET admises lorsque la lecture anticipée est activée

Valeurs	**Admise lorsque la lecture anticipée est activée, et modifiable d'un appel MQGET à l'autre**	**Admise lorsque la lecture anticipée est activée, mais non modifiable entre les appels MQGET[1]**	**Options MQGET non autorisées lorsque la lecture anticipée est activée[2]**
Valeurs MQGET MD	MsgId[3] CorrelId [3]	Codage CodedCharSetId	
Options MQGET MQGMO	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST[4] MQGMO_BROWSE_NEXT[4] MQGMO_BROWSE_MESSAGE_UNDER_CURSOR [4]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP_BACKOUT MQGMO_MSG_UNDER_CURSOR[4] MQGMO_LOCK MQGMO_UNLOCK

Tableau 15. Options MQGET admises lorsque la lecture anticipée est activée (suite)			
Valeurs	**Admise lorsque la lecture anticipée est activée, et modifiable d'un appel MQGET à l'autre**	**Admise lorsque la lecture anticipée est activée, mais non modifiable entre les appels MQGET[1]**	**Options MQGET non autorisées lorsque la lecture anticipée est activée[2]**
Valeurs MQGMO		MsgHandle	

1. Si ces options sont modifiées d'un appel MQGET à l'autre, un code anomalie MQRC_OPTIONS_CHANGED est renvoyé.
2. Si ces options sont spécifiées lors du premier appel MQGET, la lecture anticipée est désactivée. Si ces options sont spécifiées lors d'un appel MQGET ultérieur, un code anomalie MQRC_OPTIONS_ERROR est renvoyé.
3. Les applications client doivent être informées que, si les valeurs MsgId et CorrelId sont modifiées d'un appel MQGET à l'autre, il se peut que les messages comportant les valeurs précédentes aient déjà été envoyés au client et qu'ils restent dans la mémoire tampon de lecture anticipée du client tant qu'ils ne sont pas consommés (ou automatiquement purgés).
4. Le premier appel MQGET détermine si les messages doivent être explorés ou extraits d'une file d'attente lorsque la lecture anticipée est activée. Si l'application tente d'utiliser une combinaison d'exploration et d'extraction, un code anomalie MQRC_OPTIONS_CHANGED est renvoyé.
5. MQGMO_MSG_UNDER_CURSOR n'est pas possible avec la lecture anticipée. Les messages peuvent être explorés ou extraits lorsque la lecture anticipée est activée mais pas une combinaison des deux.

Une application exécutée sur un IBM MQ MQI client peut se connecter simultanément à plusieurs gestionnaires de files d'attente ou utiliser un nom de gestionnaire de files d'attente comportant un astérisque (*) sur un appel MQCONN ou MQCONNX (voir les exemples dans Connexion des applications IBM MQ MQI client aux gestionnaires de files d'attente).

A quoi correspond un client transactionnel étendu ?

Un client transactionnel étendu IBM MQ peut mettre à jour les ressources gérées par un autre gestionnaire de ressources, sous le contrôle d'un gestionnaire de transactions externe.

Si vous ne maîtrisez pas les concepts de la gestion des transactions, consultez la rubrique «Gestion et prise en charge des transactions», à la page 156.

Notez que le client transactionnel XA est maintenant fourni avec IBM MQ.

Une application client peut participer à une unité d'oeuvre gérée par un gestionnaire de files d'attente auquel elle est connectée. Au sein de l'unité d'oeuvre, l'application client peut placer des messages dans des files d'attente (et d'en extraire) appartenant à ce gestionnaire de files d'attente. L'application client peut ensuite utiliser l'appel **MQCMIT** pour valider l'unité d'oeuvre ou l'appel **MQBACK** pour annuler l'unité d'oeuvre. Cependant, au sein de la même unité d'oeuvre, l'application client ne peut pas mettre à jour les ressources d'un autre gestionnaire de ressources, par exemple les tables d'une base de données Db2. L'utilisation d'un client transactionnel étendu IBM MQ permet de supprimer cette restriction.

Un client transactionnel étendu IBM MQ est un IBM MQ MQI client doté d'une fonction supplémentaire. Cette fonction permet à une application client, au sein de la même unité d'oeuvre :

- de placer des messages dans des files d'attente (et d'en extraire) appartenant au gestionnaire de files d'attente auquel elle est connectée ;
- de mettre à jour les ressources d'un gestionnaire de ressources autre qu'un gestionnaire de files d'attente IBM MQ

Cette unité d'oeuvre doit être gérée par un gestionnaire de transactions externe exécuté sur le même système que l'application client. L'unité d'oeuvre ne peut pas être gérée par le gestionnaire de files d'attente auquel l'application client est connectée. En d'autres termes, le gestionnaire de files d'attente peut faire office uniquement de gestionnaire de ressources, et non de gestionnaire de transactions. Cela signifie également que l'application client peut valider ou annuler l'unité d'oeuvre à l'aide de l'interface de programmation d'application (API) fournie par le gestionnaire de transactions externe. L'application client ne peut donc pas utiliser les appels MQI, **MQBEGIN**, **MQCMIT** et **MQBACK**.

Le gestionnaire de transactions externe communique avec le gestionnaire de files d'attente en tant que gestionnaire de ressources à l'aide du même canal MQI que celui utilisé par l'application client connectée au gestionnaire de files d'attente. Cependant, dans une situation de reprise après incident, lorsqu'aucune application n'est en cours d'exécution, le gestionnaire de transactions peut utiliser un canal MQI dédié pour rétablir les unités d'oeuvre incomplètes auxquelles participait le gestionnaire de files d'attente au moment de l'incident.

Dans cette section, un IBM MQ MQI client qui ne possède pas la fonction transactionnelle étendue s'appelle un client de base IBM MQ. Vous pouvez donc considérer qu'un client transactionnel étendu IBM MQ est constitué d'un client de base IBM MQ auquel s'ajoute la fonction transactionnelle étendue.

Remarque : IBM i Le IBM MQ MQI client sur IBM i ne prend pas en charge la fonction transactionnelle étendue IBM MQ.

Plateformes prises en charge pour les clients transactionnels étendus

Multi

Les clients transactionnels étendus sont disponibles pour toutes les éditions Multiplatforms prenant en charge un client de base. Les clients ne sont pas disponibles pour z/OS.

Une application client qui utilise un client transactionnel étendu peut se connecter à un gestionnaire de files d'attente des produits IBM MQ 9.0suivants ou ultérieurs uniquement:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ pour Linux
- Windows IBM MQ for Windows

z/OS Bien qu'il n'existe aucun client transactionnel étendu qui s'exécute sous z/OS, une application client utilisant un client de ce type peut se connecter à un gestionnaire de files d'attente qui s'exécute sous z/OS.

Pour chaque plateforme, la configuration matérielle et logicielle requise pour le client transactionnel étendu est la même que celle du client de base IBM MQ. Un langage de programmation est pris en charge par un client transactionnel étendu s'il est pris en charge par le client de base IBM MQ MQ et par le gestionnaire de files d'attente que vous utilisez.

Pour plus d'informations sur les gestionnaires de transactions externes pour toutes les plateformes, voir Configuration système requise pour IBM MQ.

Mode de connexion du client au serveur

Un client se connecte à un serveur à l'aide de MQCONN ou de MQCONNX, et communique via un canal.

Une application exécutée dans l'environnement client IBM MQ doit maintenir une connexion active entre les postes client et serveur.

La connexion est établie par une application émettant un appel MQCONN ou MQCONNX. Les clients et les serveurs communiquent via des *canaux MQI* ou, lors de l'utilisation de conversations partagées, chacun partage une instance de canal MQI. Lorsque l'appel aboutit, l'instance de canal MQI ou la conversation reste connectée tant que l'application n'émet pas un appel MQDISC. C'est le cas pour tous les gestionnaires de files d'attente auxquels une application doit se connecter.

Client et gestionnaire de files d'attente sur la même machine

Vous pouvez également exécuter une application dans l'environnement IBM MQ MQI client lorsque votre machine dispose également d'un gestionnaire de files d'attente installé.

Dans cette situation, vous avez le choix de vous connecter aux bibliothèques du gestionnaire de files d'attente ou aux bibliothèques client, mais notez que, si vous vous connectez aux bibliothèques client, vous devez néanmoins définir les connexions de canal. Cela peut être utile lors de la phase de développement d'une application. Vous pouvez tester votre programme sur votre propre machine, sans aucune dépendance, sur d'autres machines et vous assurer qu'il continue de fonctionner lorsque vous le placez dans un environnement IBM MQ MQI client indépendant.

Clients sur des plateformes différentes

Dans cet exemple, la machine serveur communique avec trois IBM MQ MQI clients sur différentes plateformes.

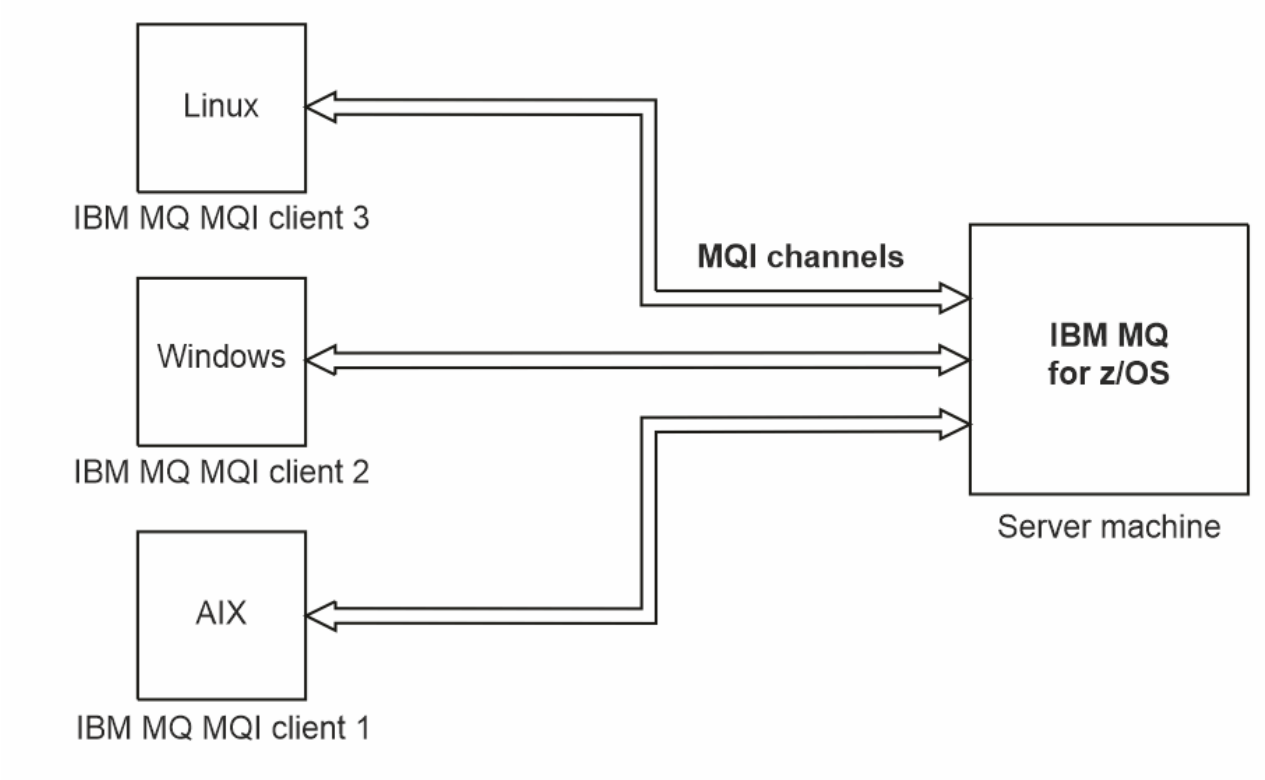


Figure 52. Serveur IBM MQ connecté à des clients sur des plateformes différentes

D'autres environnements plus complexes sont possibles. Par exemple, un client IBM MQ peut se connecter à plusieurs gestionnaires de files d'attente ou à n'importe quel nombre de gestionnaires de files d'attente connectés dans le cadre d'un groupe de partage de files d'attente.

Utilisation des versions différentes des logiciels client et serveur

Si vous utilisez des versions antérieures des produits IBM MQ, assurez-vous que la conversion de code du CCSID de votre client est prise en charge par le serveur.

Un client IBM MQ peut se connecter à toutes les versions prises en charge du gestionnaire de files d'attente. Si vous vous connectez à une version antérieure du gestionnaire de files d'attente, vous ne pouvez pas utiliser les fonctions et structures d'une version antérieure du produit dans votre application IBM MQ sur le client.

Un gestionnaire de files d'attente IBM MQ peut communiquer avec des clients de différentes versions avec lui-même en négociant jusqu'au niveau de protocole le plus élevé mutuellement pris en charge. Cela signifie que les clients plus anciens peuvent être utilisés avec des niveaux de gestionnaire de files d'attente plus récents. Il est recommandé que le client et le serveur soient au niveau des versions de IBM MQ qui sont actuellement prises en charge pour faciliter le diagnostic des problèmes et activer la prise en charge par IBM.

Pour plus d'informations, voir les langages de programmation pris en charge dans Développement d'applications.

Gestion et prise en charge des transactions

Présentation de la gestion des transactions et description de la prise en charge des transactions par IBM MQ.

Un *gestionnaire de ressources* est un sous-système informatique détenant et gérant des ressources accessibles aux applications et mises à jour par ces dernières. Voici des exemples de gestionnaires de ressources :

- Un gestionnaire de files d'attente IBM MQ dont les files d'attente sont les ressources
- Base de données Db2 dont ses tables sont les ressources

Lorsqu'une application met à jour les ressources d'un ou de plusieurs gestionnaires de ressources, il peut y avoir un besoin métier garantissant que certaines mises à jour aboutissent toutes sous la forme d'un groupe ou qu'aucune n'aboutit. La raison de ce type de besoin est que les données métier restent dans un état incohérent si certaines de ces mises à jour ont abouti alors que d'autres n'ont pas abouti.

Les mises à jour des ressources gérées de cette façon sont censées se produire au sein d'une *unité d'oeuvre*, à savoir une *transaction*. Un programme d'application peut regrouper un ensemble de mises à jour dans une unité d'oeuvre.

Pendant une unité d'oeuvre, une application émet des demandes aux gestionnaires de ressources pour mettre à jour leurs ressources. L'unité d'oeuvre prend fin lorsque l'application émet une demande de validation de toutes les mises à jour. Tant que les mises à jour ne sont pas validées, aucune d'elles n'est visible par d'autres applications accédant aux mêmes ressources. Le cas échéant, si l'application décide qu'elle ne peut pas terminer l'unité d'oeuvre pour une raison quelle qu'elle soit, elle peut émettre une demande pour annuler toutes les mises à jour qu'elle a demandées jusqu'à ce stade. Dans ce cas, aucune mise à jour ne devient visible par d'autres applications. Ces mises à jour sont généralement associées de façon logique et doivent toutes aboutir pour que l'intégrité des données soit préservée. Si une mise à jour aboutit alors qu'une autre échoue, l'intégrité des données est perdue.

Lorsqu'une unité d'oeuvre aboutit, elle est censée effectuer une *validation*. Une fois validées, toutes les mises à jour effectuées dans cette unité d'oeuvre deviennent permanentes et irréversibles. Cependant, si l'unité d'oeuvre échoue, toutes les mises à jour sont *annulées*. Ce processus au cours duquel les unités d'oeuvre sont validées ou annulées avec intégrité est appelé *coordination de point de synchronisation*.

Le stade auquel toutes les mises à jour au sein d'une unité d'oeuvre sont validées ou annulées est appelé *point de synchronisation*. Une mise à jour au sein d'une unité d'oeuvre est censée se produire *sous le contrôle d'un point de synchronisation*. Si une application demande une mise à jour qui est *en dehors du contrôle d'un point de synchronisation*, le gestionnaire de ressources valide immédiatement la mise à jour, même si une unité d'oeuvre est en cours, et la mise à jour ne peut pas être ultérieurement.

Le sous-système informatique qui gère les unités d'oeuvre est appelé *gestionnaire de transactions* ou *coordinateur de point*.

Une unité d'oeuvre *locale* est une unité d'oeuvre dans laquelle les seules ressources mises à jour sont celles du gestionnaire de files d'attente IBM MQ. Dans ce cas, la coordination de point de synchronisation est fournie par le gestionnaire de files d'attente proprement dit, à l'aide d'un processus de validation à une seule phase.

Une unité d'oeuvre *globale* est une unité d'oeuvre dans laquelle les ressources appartenant à d'autres gestionnaires de files d'attente, tels que les bases de données compatibles XA, sont également mises à jour. Dans ce cas, une procédure de validation en deux phases doit être utilisée, et l'unité d'oeuvre peut être coordonnée par le gestionnaire de files d'attente proprement dit ou en externe par un autre gestionnaire de transactions compatible XA, tel que IBM TXSeries, ou BEA.

Un gestionnaire de transactions est chargé de garantir que toutes les mises à jour des ressources au sein d'une unité d'oeuvre aboutissent ou qu'aucune d'elles n'abouti. Pour valider ou annuler une unité d'oeuvre, une application émet une demande à un gestionnaire de transactions. CICS et WebSphere

Application Server sont des exemples de gestionnaires de transactions, bien que les deux possèdent également d'autres fonctions.

Certains gestionnaires de ressources fournissent leur propre fonction de gestion de transactions. Par exemple, un gestionnaire de files d'attente IBM MQ peut gérer des unités d'oeuvre impliquant des mises à jour de ses propres ressources et des mises à jour des tables Db2. Le gestionnaire de files d'attente n'a pas besoin d'un gestionnaire de transactions distinct pour effectuer cette fonction, même si un gestionnaire de transactions peut être utilisé s'il s'agit d'un besoin métier. Si un gestionnaire de transactions distinct est utilisé, il est désigné par *gestionnaire de transactions externe*.

Pour qu'un gestionnaire de transactions externe gère une unité d'oeuvre, une interface standard doit exister entre le gestionnaire de transactions et tous les gestionnaires de ressources participant à l'unité d'oeuvre. Cette interface permet au gestionnaire de transactions et à un gestionnaire de ressources de communiquer l'un avec l'autre. L'une de ces interfaces est l'*interface XA*, qui correspond à une interface standard prise en charge par un certain nombre de gestionnaires de transactions et de gestionnaires de ressources. L'interface XA est publiée par The Open Group dans le document *Distributed Transaction Processing: The XA Specification*.

Lorsque plusieurs gestionnaires de ressources participent à une unité d'oeuvre, un gestionnaire de transactions doit utiliser un protocole de *validation à deux phases* pour garantir que toutes les mises à jour au sein de l'unité d'oeuvre aboutissent ou qu'aucune d'elles n'aboutit, même si une erreur système se produit. Lorsqu'une application émet une demande à un gestionnaire de transactions pour valider une unité d'oeuvre, le gestionnaire de transactions effectue les opérations suivantes :

Phase 1 (préparation de la validation)
: Le gestionnaire de transactions demande à chaque gestionnaire de ressources participant à l'unité d'oeuvre de garantir que toutes les informations relatives aux mises à jour prévues de ses ressources sont dans un état récupérable. Un gestionnaire de ressources effectue généralement ce processus en écrivant les informations dans un journal et en garantissant que les informations sont écrites sur disque dur. La phase 1 se termine lorsque le gestionnaire de transactions reçoit de la part de chaque gestionnaire de ressources une notification selon laquelle les informations relatives aux mises à jour prévues de ses ressources sont dans un état récupérable.

Phase 2 (validation)
: Une fois la phase 1 terminée, le gestionnaire de transactions prend la décision irrévocable de valider l'unité d'oeuvre. Il demande à chaque gestionnaire de ressources participant à l'unité d'oeuvre de valider les mises à jour de ses ressources. Lorsqu'un gestionnaire de ressources reçoit cette demande, il doit valider les mises à jour. Il n'a pas la possibilité de les annuler à ce stade. La phase 2 se termine lorsque le gestionnaire de transactions reçoit de la part de chaque gestionnaire de ressources une notification selon laquelle il a validé les mises à jour de ses ressources.

L'interface XA utilise un protocole de validation à deux phases.

Pour plus d'informations, voir Scénarios de support transactionnel.

IBM MQ prend également en charge Microsoft Transaction Server (COM +). L'utilisation de Microsoft Transaction Server (COM +) permet d'obtenir des informations sur la configuration de IBM MQ pour bénéficier du support COM +.

Extension des fonctions du gestionnaire de files d'attente

Vous pouvez étendre les fonctions du gestionnaire de files d'attente à l'aide d'exits utilisateur, d'exits API ou de services optionnels.

Exits utilisateur

Les exits utilisateur vous permettent d'insérer votre propre code dans une fonction du gestionnaire de files d'attente. Les exits utilisateur pris en charge sont les suivants :

Exits de canal
: Ces exits modifient le mode de fonctionnement des canaux. Les exits de canal sont décrits dans Programmes d'exit de canal pour les canaux de messagerie.

Exits de conversion de données
Ces exits créent des fragments de code source pouvant être placés dans des programmes d'application pour convertir des données d'un format à un autre. Les exits de conversion de données sont décrits dans Ecriture d'exits de conversion de données.

Exit de charge de travail du cluster
La fonction de cet exit est définie par le fournisseur de l'exit. Des informations sur les définitions d'appel sont fournies dans MQ_CLUSTER_WORKLOAD_EXIT - Description d'appel.

Exits API

Les exits API permettent d'écrire du code modifiant le comportement des appels API IBM MQ, tels que MQPUT et MQGET, puis d'insérer le code immédiatement avant ou après ces appels. L'insertion est automatique ; le gestionnaire de files d'attente conduit le code d'exit aux points enregistrés. Pour plus d'informations sur les exits API, voir Utilisation et écriture d'exits API.

Services optionnels

Les services optionnels possèdent des interfaces formalisées (API) dotées de plusieurs points d'entrée.

Une implémentation d'un service optionnel est appelée *composant de service*. Vous pouvez utiliser les composants fournis avec IBM MQ, ou écrire votre propre composant pour exécuter les fonctions dont vous avez besoin.

A l'heure actuelle, les services optionnels suivants sont fournis :

Serveur d'autorisation
Le service d'autorisation vous permet de générer votre propre fonction de sécurité.

Le composant de service par défaut implémentant le service est le gestionnaire des droits d'accès aux objets (OAM). Par défaut, le gestionnaire des droits d'accès aux objets (OAM) est actif et vous n'avez pas à intervenir pour le configurer. Vous pouvez utiliser l'interface de service d'autorisation pour créer d'autres composants afin de remplacer ou d'étendre le gestionnaire OAM. Pour plus d'informations sur l'OAM, voir Configuration de la sécurité sur les systèmes AIX, Linux, and Windows.

Service annuaire
Le service annuaire permet aux applications de partager des files d'attente en identifiant les files d'attente éloignées comme si elles étaient des files d'attente locales.

Vous pouvez écrire votre propre composant de service annuaire. Vous pouvez être amené à le faire si vous avez l'intention d'utiliser le service annuaire avec IBM MQ, par exemple. Pour utiliser le service annuaire, vous devez disposer d'un composant écrit par l'utilisateur ou fourni par un fournisseur de logiciel différent. Par défaut, le service annuaire est inactif.

Concepts associés
Exits utilisateur, exits API et services optionnels d'IBM MQ

Interfaces de langage IBM MQ Java

IBM MQ fournit trois interfaces de programme d'application (API) à utiliser dans les applications Java : IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMSet IBM MQ classes for Java.

IBM prend en charge les normes ouvertes et y participe activement.

- Depuis la IBM MQ 8.0, le produit implémente la norme JMS 2.0 , qui introduit une nouvelle API simplifiée ainsi que des fonctions telles que les abonnements partagés.
- Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est également pris en charge.
- En outre, WebSphere Liberty prend en charge JMS 2.0 et Jakarta Messaging 3.0 avec IBM MQ.

Dans IBM MQ , il existe trois API à utiliser dans les applications Java :

JM 3.0 **IBM MQ classes for Jakarta Messaging**
IBM MQ classes for Jakarta Messaging est un fournisseur Jakarta Messaging qui implémente les interfaces Jakarta Messaging pour IBM MQ en tant que système de messagerie. Jakarta Connectors Architecture fournit un moyen standard de connecter des applications s'exécutant dans un environnement Jakarta EE à un système d'information d'entreprise (EIS) tel que IBM MQ ou Db2.

JMS 2.0 **IBM MQ classes for JMS**
IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ en tant que système de messagerie. Java Platform, Enterprise Edition Connector Architecture (JCA) fournit une méthode standard de connexion des applications en cours d'exécution dans un environnement Java EE à un système d'information d'entreprise (EIS) tel qu'IBM MQ ou Db2.

IBM MQ classes for Java
IBM MQ classes for Java vous permet d'utiliser IBM MQ dans un environnement Java. IBM MQ classes for Java permet à une application Java de se connecter à IBM MQ en tant que client IBM MQ ou de se connecter directement à un gestionnaire de files d'attente IBM MQ.

Remarque :

- JMS 2.0 a été remplacé par Jakarta Messaging. IBM MQ classes for JMS continue de prendre en charge la norme JMS 2.0 , mais les futures améliorations apportées à la messagerie Java n'apparaîtront que dans Jakarta Messaging, et donc dans IBM MQ classes for Jakarta Messaging. Les IBM MQ classes for JMS sont uniquement recommandées pour la maintenance et l'extension des applications JMS 2.0 existantes. IBM MQ classes for Jakarta Messaging doit être la technologie préférée pour les nouveaux développements.
- Stabilized IBM MQ classes for Java est fonctionnellement stabilisé au niveau livré dans IBM MQ 8.0. Les applications existantes qui utilisent IBM MQ classes for Java continueront d'être intégralement prises en charge, mais cette API est stabilisée, ce qui signifie qu'aucune nouvelle fonction ne sera ajoutée et que toute demande d'amélioration sera rejetée. "Intégralement prises en charge" signifie que les incidents seront corrigés et que toute modification nécessaire suite à la modification de la configuration système requise pour IBM MQ sera apportée.

JM 3.0 Depuis la IBM MQ 9.3, les IBM MQ classes for Java, IBM MQ classes for JMSet IBM MQ classes for Jakarta Messaging sont générés avec Java 8. Les environnements d'exécution Java à ou au-dessus de ces niveaux doivent être utilisés pour exécuter des applications à l'aide de ces interfaces.

Concepts associés

Accès à IBM MQ à partir de Java -Choix de l'API

Pourquoi utiliser des classes IBM MQ pour Jakarta Messaging?

Pourquoi utiliser IBM MQ classes for JMS ?

Pourquoi utiliser IBM MQ classes for Java ?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont les fournisseurs de messagerie fournis avec IBM MQ. Chacun de ces fournisseurs fournit également deux ensembles d'extensions à l'API de messagerie. Les applications Java Platform, Standard Edition (Java SE) et Java Platform, Enterprise Edition (Java EE) peuvent utiliser ces fournisseurs de messagerie.

JM 3.0 IBM MQ 9.3.0 a introduit la prise en charge de Jakarta Messaging 3.0. JMS 2.0 est toujours entièrement pris en charge.

Les spécifications JMS et Jakarta Messaging définissent un ensemble d'interfaces que les applications peuvent utiliser pour effectuer des opérations de messagerie. Le produit prend en charge la version JMS 2.0 de la norme JMS . Cette implémentation fournit toutes les fonctions de l'API classique, mais requiert moins d'interfaces et elle est plus simple à utiliser. Pour plus d'informations, voir «Modèle JMS et Jakarta Messaging», à la page 163 et la spécification JMS 2.0 sur Java.net. JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging est également pris en charge.

Le package jakarta.jms (Jakarta Messaging 3.0) ou javax.jms (JMS 2.0) spécifie les détails des interfaces de messagerie et un fournisseur de messagerie implémente ces interfaces pour un produit de messagerie spécifique. Exemple :

- IBM MQ classes for JMS est un fournisseur JMS qui implémente les interfaces JMS pour IBM MQ et fournit également les deux ensembles d'extensions à l'API JMS :
  - Extensions IBM MQ JMS
  - Extensions IBM JMS
- Une fabrique de connexions, une file d'attente ou un objet de rubrique créé à l'aide de javax.dimsou de jakarta.jms, d'interfaces ou d'un ensemble d'extensions JMS peut être adressé à l'aide de l'une de ces API, c'est-à-dire qu'il peut être transtypé vers l'une des interfaces. Pour conserver la portabilité des applications au plus haut niveau, utilisez l'API la plus générique qui convient à vos exigences.

Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

Extensions IBM MQ JMS

IBM MQ classes for JMS fournit également des extensions à l'API JMS. IBM MQ classes for JMS contient des extensions implémentées dans des objets MQConnectionFactory, MQQueue et MQTopic. Ces objets ont des propriétés et des méthodes spécifiques à IBM MQ. Les objets peuvent être des objets administrés ou une application peut créer les objets de façon dynamique lors de la phase d'exécution. Ces extensions sont appelées extensions IBM MQ JMS . Notez que, dans cette documentation, les objets créés dynamiquement par une application lors de l'exécution ne sont pas considérés comme des objets gérés.

Extensions IBM JMS

En plus des extensions IBM MQ JMS , IBM MQ classes for JMS fournit un ensemble plus générique d'extensions à l'API JMS ou à Java en tant que langage de programmation utilisé. Ces extensions sont appelées extensions IBM JMS et ont les objectifs généraux suivants:

- Fournir un niveau de cohérence plus élevé entre les fournisseurs IBM JMS .
- Pour faciliter l'écriture d'une application de pont entre deux systèmes de messagerie IBM .
- Pour faciliter le port d'une application d'un fournisseur IBM JMS à un autre.

Le principal objectif de ces extensions concerne la création et la configuration dynamiques des fabriques de connexions et des destinations lors de la phase d'exécution, mais elles offrent aussi une fonction qui n'est pas directement liée à la messagerie, comme l'identification des problèmes.

Tâches associées

Utilisation des classes IBM MQ pour JMS/Jakarta Messaging

Configuration des ressources JMS et Jakarta Messaging

JM 3.0 IBM MQ classes for Jakarta Messaging: présentation

IBM MQ 9.3.0 introduit la prise en charge de Jakarta Messaging. Pour Jakarta Messaging 3.0, le contrôle de la spécification JMS a été déplacé de Oracle vers le processus de communauté Java . Toutefois, Oracle conserve le contrôle du nom "javax", qui est utilisé dans d'autres technologies Java . Par conséquent, bien que Jakarta Messaging 3.0 soit fonctionnellement équivalent à JMS 2.0, il existe des différences de dénomination. Le nom officiel de la version 3.0 est Jakarta Messaging au lieu de Java Message Service, et les noms de package et de constante sont préfixés avec `jakarta` au lieu de `javax`.

Arrière-plan

Depuis de nombreuses années, la plateforme Java se présente sous deux formes: Standard Edition et Enterprise Edition.

Java Platform, Standard Edition (parfois abrégé en Java SE) est le langage de base et les bibliothèques de classes, pouvant être exécutés dans un contexte autonome. La plupart des packages Java dans Java SE ont des noms commençant par "java".

Java Platform, Enterprise Edition (Java EE) l'étend, en ajoutant des fonctionnalités telles que la messagerie, divers beans, la transactionnalité, etc. Certaines de ces technologies peuvent également être utilisées dans un contexte Java SE . La plupart des packages Java dans Java EE ont historiquement des noms commençant par "javax."-il existe un croisement, cependant, certains packages Java SE ont "javax." comme préfixe de leur nom.

Java Message Service (JMS) fait partie de Java Platform, Enterprise Edition. Java EE 7 intègre JMS 2.0.

Jusqu'à Java EE 7, les technologies étaient sous la direction de Oracle.

Les technologies Java EE sont récemment passé de la gestion de Oracle à un processus communautaire supervisé par Eclipse Foundation.

Comme le "javax". Le nom n'a pas pu être déplacé vers le nouveau projet, un nouveau nom a été adopté- tous les packages et les noms de propriété sont maintenant précédés de "jakarta". et Java Platform, Enterprise Edition sera appelé "Jakarta EE" à l'avenir. La numérotation des versions a continué: la version 8 était une version provisoire qui peut être largement ignorée, et Jakarta EE 9 est le point où le "jakarta". Le préfixe prend effet.

La technologie Jakarta EE principale qui s'applique dans le contexte IBM MQ est Jakarta Messaging 3.0 -le successeur de Java Message Service (JMS) 2.0. Par conséquent, Jakarta EE 9 intègre Jakarta Messaging 3.0.

IBM MQ continue de prendre en charge Java EE 7 et JMS 2.0, tout en prenant en charge Jakarta EE 9 et Jakarta Messaging 3.0.

Ce qui est distribué: Java SE

Pour Java Platform, Standard Edition, en plus de IBM MQ classes for JMS (qui prend en charge les opérations JMS 2.0 avec IBM MQ), IBM MQ 9.3.0 et les versions ultérieures fournissent IBM MQ classes for Jakarta Messaging. Ces classes fournissent un fournisseur Jakarta Messaging 3.0 qui s'intègre à IBM MQ, permettant l'utilisation de gestionnaires de files d'attente IBM MQ pour faciliter les opérations Jakarta Messaging .

Ils sont fournis sous la forme d'un fichier JAR standard, `com.ibm.mq.jakarta.client.jar`, dans le sous-répertoire `java/lib` de l'installation IBM MQ .

Pour une utilisation dans des conteneurs OSGi, tels que Apache Felix ou Eclipse Equinox, IBM MQ fournit également une paire de bundles OSGi:

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

où *V.R.M.F* représente la version de IBM MQ, par exemple 9.3.0.0. Ces bundles se trouvent dans le sous-répertoire `java/lib/OSGi` de l'installation IBM MQ .

Ce qui est distribué: Jakarta EE 9

Pour prendre en charge la messagerie IBM MQdans un serveur d'applications compatible Jakarta EE 9 , IBM MQ fournit un adaptateur de ressources compatible Jakarta EE 9: `wmq.jakarta.jmsra.rar`. Vous pouvez le trouver dans le sous-répertoire `java/lib/jca` de l'installation IBM MQ .

IBM MQ continue de fournir un adaptateur de ressources compatible Java EE 7 , `wmq.jmsra.rar`, dans le sous-répertoire `java/lib/jca` de l'installation IBM MQ .

Mode de distribution de ces artefacts

Ces fichiers JAR et le fichier RAR de l'adaptateur de ressources sont fournis avec les artefacts préexistants dans le support d'installation IBM MQ habituel, à la fois le support d'installation spécifique

à la plateforme, tel que les fichiers "`.rpm`", et le support redistribuable, tel que les fichiers JAR du client redistribuable auto-extractibles.

Modifications apportées entre JMS 2.0 et Jakarta Messaging 3.0

Jakarta EE 9 et Jakarta Messaging 3.0 n'introduisent aucune nouvelle fonctionnalité. Tout ce qui change, ce sont les noms. Par exemple, lorsque vous utilisez "javax.jms.Connection" dans JMS 2.0, vous utilisez "jakarta.jms.Connection" dans Jakarta Messaging 3.0.

Au fur et à mesure que Eclipse Foundation fait avancer la plateforme Jakarta EE , elle s'appuie sur cette base et cette convention de dénomination sera utilisée pour les nouvelles fonctionnalités introduites à l'avenir.

Modifications apportées entre IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging

Récapitulatif

IBM MQ classes for JMS, qui fournit une prise en charge pour JMS 2.0, reste disponible et est recommandé principalement pour la maintenance et l'extension des applications existantes. Ils sont entièrement pris en charge.

IBM MQ classes for Jakarta Messaging, qui fournit la prise en charge de Jakarta Messaging 3.0, est recommandé pour le nouveau développement.

Dans IBM MQ 9.3.0, ces deux offres étaient fonctionnellement équivalentes. Seul le nom diffère. Toutefois, les nouvelles fonctionnalités de messagerie sont plus susceptibles d'apparaître dans IBM MQ classes for Jakarta Messaging que dans IBM MQ classes for JMS.

Les deux offres sont interopérables. Les messages générés par IBM MQ classes for JMS peuvent être consommés par IBM MQ classes for Jakarta Messaging, et inversement. Mais les deux offres ne doivent pas coexister dans une seule application.

Nouveaux noms

Tableau 16. Modifications apportées aux noms de package

IBM MQ classes for JMS nom du package	IBM MQ classes for Jakarta Messaging nom du package
com.ibm.mq.jms[.*]	com.ibm.mq.jakarta.jms[.*]
com.ibm.jms	com.ibm.jakarta.jms
com.ibm.msg.client.jms.*	com.ibm.msg.client.jakarta.jms.*
com.ibm.msg.client.wmq.*	com.ibm.msg.client.jakarta.wmq.*

Les packages relatifs aux services communs (trace, journalisation, support de langue nationale, etc.) et aux implémentations JMQI (locales et distantes) sont communs à IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, de sorte qu'aucune modification n'est nécessaire dans ces domaines.

Notez que les noms de propriété ont également été modifiés. Par exemple, la propriété permettant d'activer les extensions IBM MQ dans IBM MQ classes for Jakarta Messaging est **`com.ibm.mq.jakarta.jms.SupportMQExtensions`**.

Les noms de propriété qui sont indépendants de IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, tels que les différentes propriétés **`com.ibm.msg.client.commonservices.trace.*`** , s'appliquent également aux deux offres.

Utilitaires d'administration

Les utilitaires **`crtmqenv`** et **`setmqenv`** acceptent désormais une option permettant de spécifier si le chemin d'accès aux classes doit être configuré pour IBM MQ classes for JMS (`-j 2.0`) ou IBM

MQ classes for Jakarta Messaging (`-j 3.0`), et il existe des variantes IBM MQ classes for Jakarta Messaging des utilitaires **runjms** , appelées **runjms30** et noms similaires.

L'utilitaire **dspmqver** , lorsqu'il est demandé de générer des rapports sur les composants Java , inclut IBM MQ classes for Jakarta Messaging dans sa sortie.

Pour configurer les objets IBM MQ classes for Jakarta Messaging à extraire via JNDI, le nouvel utilitaire **JMS30Admin** est équivalent à l'utilitaire **JMSAdmin** pour IBM MQ classes for JMS.

Notez que les objets sous-jacents proviennent de différents packages. Les définitions JNDI créées par **JMSAdmin** ne peuvent pas être utilisées par IBM MQ classes for Jakarta Messaginget celles créées par **JMS30Admin** ne peuvent pas non plus être utilisées par IBM MQ classes for JMS.

Remarque : Il n'y a pas de prise en charge pour les objets IBM MQ classes for Jakarta Messaging fournis par IBM MQ Explorer; son intégration JNDI est uniquement pour IBM MQ classes for JMS .

Concepts associés

Pourquoi utiliser des classes IBM MQ pour Jakarta Messaging?

Modèle JMS et Jakarta Messaging

Le modèle JMS et Jakarta Messaging définit un ensemble d'interfaces que les applications Java peuvent utiliser pour effectuer des opérations de messagerie. IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging sont des fournisseurs de messagerie qui définissent la façon dont les objets de messagerie Java sont liés aux concepts IBM MQ . Les spécifications JMS et Jakarta Messaging s'attendent à ce que certains objets de messagerie soient des objets gérés.

Depuis la IBM MQ 8.0, le produit prend en charge la version JMS 2.0 de la norme JMS, qui a introduit une API simplifiée, tout en conservant l'API classique, à partir de JMS 1.1.

JMS 3.0 IBM MQ 9.3.0 a introduit la prise en charge de Jakarta Messaging 3.0. JMS 2.0 est toujours entièrement pris en charge. Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

API simplifiée

JMS 2.0 a introduit l'API simplifiée, tout en conservant les interfaces spécifiques au domaine et indépendantes du domaine d' JMS 1.1. L'API simplifiée réduit le nombre d'objets nécessaires à l'envoi et à la réception de messages et se compose des interfaces ci-dessous :

ConnectionFactory
ConnectionFactory est un objet géré qui est utilisé par un client JMS pour créer une connexion. Cette interface est également utilisée dans l'API classique.

JMSContexte
Cet objet combine les objets Connection et Session de l'API classicCd. Les objets JMSContext peuvent être créés à partir d'autres objets JMSContext, auquel cas la connexion sous-jacente est dupliquée.

JMSExpéditeur
Un objet JMSProducer est créé par un objet JMSContext et il utilisé pour envoyer des messages à une file d'attente ou à une rubrique. L'objet JMSProducer entraîne la création des objets nécessaires à l'envoi du message.

JMSConsommateur
Un objet JMSConsumer est créé par un objet JMSContext et il est utilisé pour recevoir des messages provenant d'une rubrique ou d'une file d'attente.

L'API simplifiée a un certain nombre d'effets :

- L'objet JMSContext démarre toujours automatiquement la connexion sous-jacente.
- Les objets JMSProducers et JMSConsumers peuvent maintenant traiter directement le corps des messages, sans devoir détenir l'objet message complet, par le biais de la méthode getBody du message.

- Les propriétés du message peuvent être définies sur l'objet JMSProducer à l'aide du chaînage de méthodes avant d'envoyer un 'corps', c'est-à-dire le contenu des messages. L'objet JMSProducer traite la création de tous les objets nécessaires à l'envoi du message. A l'aide de JMS 2.0, les propriétés sont définies et le message suivant est envoyé :

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 a également introduit des abonnements partagés dans lesquels les messages peuvent être partagés entre plusieurs consommateurs. Tous les abonnements JMS 1.1 sont traités en tant qu'abonnements non partagés.

API classique

Les interfaces JMS principales qui constituent l'API classique sont répertoriées ci-dessous :

Destination
Une destination représente l'endroit où une application envoie des messages, l'endroit d'où elle en reçoit, ou les deux.

ConnectionFactory
Un objet ConnectionFactory encapsule un ensemble de propriétés de configuration pour une connexion. Une application utilise une fabrique de connexions pour créer une connexion.

Connexion
Un objet Connection encapsule la connexion active d'une application sur un serveur de messagerie. Une application utilise une connexion pour créer des sessions.

Session
Une session est un contexte à unité d'exécution unique pour l'envoi et la réception de messages. Une application utilise ensuite une session pour créer des messages, des expéditeurs de message et des consommateurs de message. Une session est transactionnelle ou non transactionnelle.

Message
Un objet Message encapsule un message envoyé ou reçu par une application.

MessageProducer
Une application utilise un expéditeur de message pour envoyer des messages vers une destination.

MessageConsumer
Une application utilise un consommateur de message pour la réception des messages envoyés vers une destination.

La Figure 53, à la page 165 présente ces objets et leurs relations.

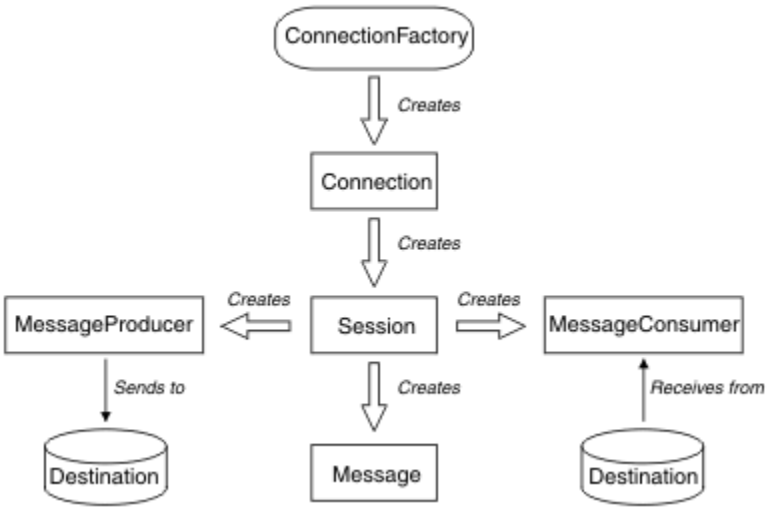


Figure 53. Objets JMS et leurs relations

Le diagramme présente les interfaces principales : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message et Destination. Une application utilise une fabrique de connexions pour créer une connexion et utilise une connexion pour créer des sessions. L'application peut ensuite utiliser une session pour créer des messages, des expéditeurs de message et des consommateurs de message. L'application utilise un expéditeur de message pour envoyer des messages vers une destination et utilise un consommateur de message pour recevoir des messages envoyés vers une destination.

Un objet Destination, ConnectionFactory ou Connection peut être utilisé simultanément par différentes unités d'exécution d'une application comportant plusieurs unités d'exécution, mais un objet Session, MessageProducer ou MessageConsumer ne peut pas être utilisé par des unités d'exécution différentes. La manière la plus simple de s'assurer qu'un objet Session, MessageProducer ou MessageConsumer n'est pas utilisé simultanément consiste à créer un objet Session distinct pour chaque unité d'exécution.

JMS prend en charge deux styles de messagerie :

- Messagerie point-à-point
- Messagerie de type publication/abonnement

Ces deux styles de messagerie sont également appelés *domaines de messagerie* et ils peuvent être combinés dans une application. Dans le domaine point-à-point, une destination est une file d'attente et, dans le domaine de publication/abonnement, une destination est une rubrique.

Avec les versions de JMS antérieures à JMS 1.1, la programmation pour le domaine point-à-point utilise un ensemble d'interfaces et de méthodes et la programmation pour le domaine de publication/abonnement utilise un autre ensemble. Les deux ensembles sont similaires, mais distincts. A partir de JMS 1.1, vous pouvez utiliser un ensemble commun d'interfaces et de méthodes qui prennent en charge les deux domaines de messagerie. Les interfaces communes fournissent une vue indépendante du domaine de chaque domaine de messagerie. Le Tableau 17, à la page 165 affiche la liste des interfaces indépendantes du domaine JMS et leurs interfaces propres au domaine correspondantes.

Tableau 17. Les interfaces indépendantes du domaine JMS et leurs interfaces propres au domaine correspondantes

Interfaces indépendantes du domaine	**Interfaces propres au domaine pour le domaine point-à-point**	**Interfaces propres au domaine pour le domaine de publication/abonnement**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connexion	QueueConnection	TopicConnection

Tableau 17. Les interfaces indépendantes du domaine JMS et leurs interfaces propres au domaine correspondantes (suite)

Interfaces indépendantes du domaine	**Interfaces propres au domaine pour le domaine point-à-point**	**Interfaces propres au domaine pour le domaine de publication/ abonnement**
Destination	File d'attente	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 prend en charge à la fois les interfaces spécifiques au domaine JMS 1.1 et l'API simplifiée de JMS 2.0. IBM MQ classes for JMS 2.0 peut donc être utilisé pour gérer des applications existantes, y compris pour développer de nouvelles fonctions dans des applications existantes.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 prend en charge les versions Jakarta Messaging des mêmes interfaces et est recommandé pour le développement de nouvelles applications.

Dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, les objets JMS sont liés aux concepts IBM MQ de la manière suivante:

- Un objet Connection possède des propriétés issues des propriétés de la fabrique de connexions qui a été utilisée pour créer la connexion. Ces propriétés contrôlent comment une application se connecte à un gestionnaire de files d'attente. Des exemples de ces propriétés sont le nom du gestionnaire de files d'attente et, pour une application qui se connecte au gestionnaire de files d'attente en mode client, le nom d'hôte ou l'adresse IP du système sur lequel s'exécute le gestionnaire de files d'attente.
- Un objet Session encapsule le descripteur de connexion IBM MQ qui définit la portée transactionnelle de la session.
- Un objet MessageProducer et un objet MessageConsumer encapsule chacun un descripteur d'objet IBM MQ.

Lorsque vous utilisez IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging, toutes les règles normales de IBM MQ s'appliquent. Sachez, en particulier, qu'une application peut envoyer un message à une file d'attente éloignée, mais elle peut uniquement recevoir un message d'une file d'attente qui appartient au gestionnaire de files d'attente auquel l'application est connectée.

La spécification JMS s'attend à ce que les objets ConnectionFactory et Destination soient des objets gérés. Un administrateur crée et gère les objets gérés dans un référentiel central et une application JMS extrait ces objets à l'aide de l'interface JNDI (Java Naming and Directory Interface).

Dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging, l'implémentation de l'interface Destination est une superclasse abstraite de Queue et Topic, et donc une instance de Destination est soit un objet Queue, soit un objet Topic. Les interfaces indépendantes du domaine traitent une file d'attente ou une rubrique en tant que destination. Le domaine de messagerie pour un objet MessageProducer ou MessageConsumer varie selon que la destination est une file d'attente ou une rubrique.

Par conséquent, dans IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging , les objets des types suivants peuvent être des objets gérés:

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- File d'attente

- Topic
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

Architecture IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging possèdent une architecture en couches. La couche de code la plus haute est une couche commune que tout fournisseur de messagerie IBM Java peut utiliser.

JM 3.0 IBM MQ 9.3.0 a introduit la prise en charge de Jakarta Messaging 3.0. JMS 2.0 est toujours entièrement pris en charge.

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging possèdent une architecture en couches, comme illustré dans le diagramme Figure 54, à la page 167. La couche de code la plus haute est une couche commune qui peut être utilisée par n'importe quel fournisseur IBM JMS ou Jakarta Messaging. Lorsqu'une application appelle une méthode JMS ou Jakarta Messaging, tout traitement de l'appel qui n'est pas spécifique à un système de messagerie est effectué par la couche commune, qui fournit également une réponse cohérente à l'appel. Les traitements de l'appel spécifiques à un système de messagerie sont délégués à une couche inférieure. Dans le diagramme suivant, le fournisseur de messagerie IBM MQ apparaît dans la couche inférieure avec deux autres fournisseurs de messagerie (fournisseur de messagerie A et fournisseur de messagerie B.)

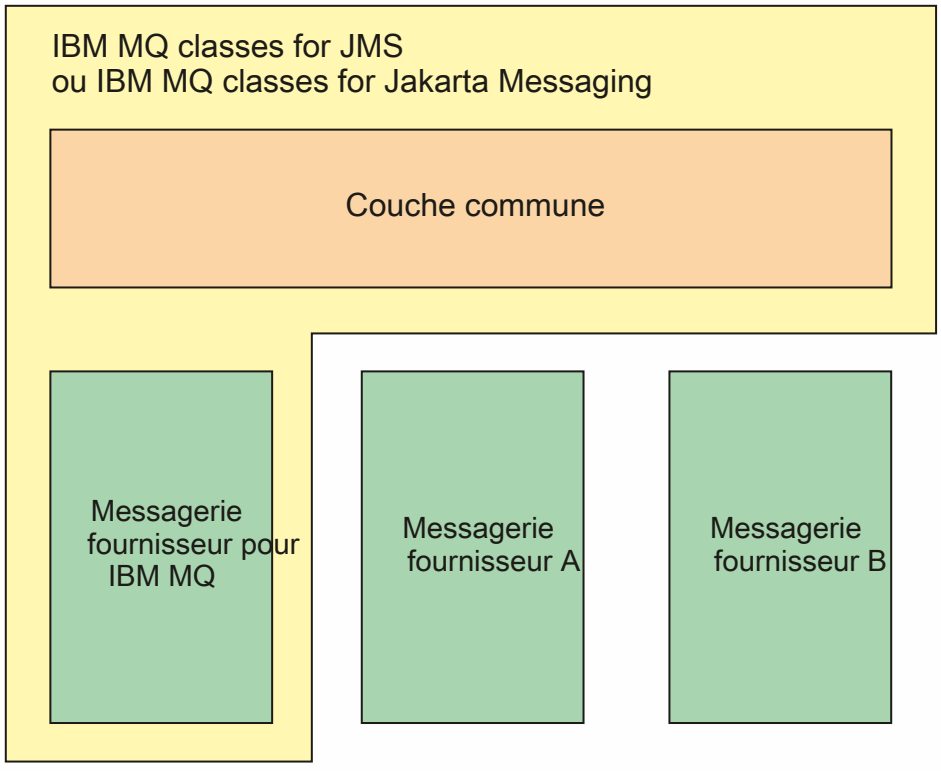


Figure 54. Architecture en couches pour les fournisseurs IBM JMS et Jakarta Messaging

Une architecture en couches répond aux objectifs suivants :

- Pour améliorer la cohérence du comportement des différents fournisseurs IBM JMS et Jakarta Messaging
- Faciliter l'écriture d'une application de passerelle entre deux systèmes de messagerie IBM

- Pour faciliter le portage d'une application d'un fournisseur IBM JMS ou Jakarta Messaging vers un autre

Tâches associées

Utilisation des classes IBM MQ pour JMS/Jakarta Messaging

Prise en charge des objets gérés

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging prennent en charge l'utilisation des objets gérés.

JM 3.0 Depuis IBM MQ 9.3.0, Jakarta Messaging 3.0 est pris en charge pour le développement de nouvelles applications. IBM MQ 9.3.0 et les versions ultérieures continuent de prendre en charge JMS 2.0 pour les applications existantes. L'utilisation de l'API Jakarta Messaging 3.0 et de l'API JMS 2.0 dans la même application n'est pas prise en charge. Pour plus d'informations, voir Utilisation des classes IBM MQ pour JMS/Jakarta Messaging.

Le flux logique dans une application JMS ou IBM MQ classes for Jakarta Messaging commence par les objets ConnectionFactory et Destination. L'application utilise un objet ConnectionFactory pour créer un objet Connection, qui représente la connexion active depuis l'application vers un serveur de messagerie. L'application utilise l'objet Connection pour créer un objet Session, qui est un contexte avec une seule unité d'exécution permettant de générer et de consommer des messages. L'application peut ensuite utiliser l'objet Session et l'objet Destination pour créer un objet MessageProducer, que l'application utilise pour envoyer des messages à la destination indiquée. La destination est soit une file d'attente, soit une rubrique du système de messagerie et est encapsulée par l'objet Destination. L'application peut aussi utiliser l'objet Session et un objet Destination pour créer un objet MessageConsumer, qu'elle utilise pour recevoir des messages envoyés à la destination indiquée.

Les spécifications JMS et Jakarta Messaging s'attendent à ce que les objets ConnectionFactory et Destination soient des objets gérés. Un administrateur crée et gère des objets gérés dans un référentiel central, et une application JMS ou Jakarta Messaging extrait ces objets à l'aide de Java Naming Directory Interface (JNDI). Le référentiel des objets gérés peut aller d'un fichier simple à un annuaire LDAP (Lightweight Directory Access Protocol).

IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging prennent en charge l'utilisation des objets gérés. Une application peut utiliser toutes les fonctions de IBM MQ classes for JMS ou de IBM MQ classes for Jakarta Messaging qui sont exposées via IBM MQ sans qu'aucune information spécifique à IBM MQne soit codée en dur dans l'application elle-même. Cela fournit à l'application une certaine indépendance par rapport à la configuration sous-jacente d'IBM MQ.

Pour obtenir cette indépendance, l'application peut utiliser JNDI pour extraire des fabriques de connexions et des destinations qui sont stockées en tant qu'objets gérés et utiliser uniquement les interfaces définies dans le package `javax.jms` (JMS 2.0) ou `jakarta.jms` (Jakarta Messaging 3.0) pour effectuer des opérations de messagerie.

JMS 2.0 Pour JMS 2.0, un administrateur peut utiliser l'outil d'administration IBM MQ JMS **JMSAdmin**ou IBM MQ Explorerpour créer et gérer des objets gérés dans un référentiel central.

JM 3.0 Pour Jakarta Messaging 3.0, vous ne pouvez pas administrer JNDI à l'aide de IBM MQ Explorer. L'administration JNDI est prise en charge par la variante Jakarta Messaging 3.0 de **JMSAdmin**, qui est **JMS30Admin**.

Un serveur d'applications fournit généralement son propre référentiel pour les objets gérés et ses propres outils pour la création et la gestion des objets. Une application Java EE JM 3.0 ou Jakarta EE peut donc utiliser JNDI pour extraire des objets gérés du référentiel du serveur d'applications ou d'un référentiel central.

Tâches associées

Configuration des ressources JMS et Jakarta Messaging

Types de communication pris en charge sur les plateformes Java EE et Jakarta EE

Sur les plateformes Java EE et Jakarta EE , IBM MQ classes for JMS et IBM MQ classes for Jakarta Messaging prennent en charge deux types de communication entre un composant d'une application et un gestionnaire de files d'attente IBM MQ .

JM 3.0 IBM MQ 9.3.0 a introduit la prise en charge de Jakarta Messaging 3.0. JMS 2.0 est toujours entièrement pris en charge. Etant donné que JMS et Jakarta Messaging partagent beaucoup de choses en commun, d'autres références à JMS dans cette rubrique peuvent être considérées comme faisant référence aux deux. Toutes les différences sont mises en évidence si nécessaire.

Les deux types de communication suivants entre un composant d'une application et un gestionnaire de files d'attente IBM MQ sont pris en charge :

- Communication sortante
- Communication entrante

Communication sortante

A l'aide de l'API JMS ou Jakarta Messaging directement, un composant d'application crée une connexion à un gestionnaire de files d'attente, puis envoie et reçoit des messages.

Par exemple, le composant d'application peut être un client d'application, un servlet, une page JavaServer Page (JSP), un bean enterprise Java (EJB), ou un bean géré par message (MDB). Dans ce type de communication, le conteneur du serveur d'applications fournit uniquement des fonctions de bas niveau pour le support des opérations de messagerie, comme le regroupement de connexions et la gestion des unités d'exécution.

Communication entrante

Dans le cas de la communication entrante, un message parvenant à une destination est livré à un bean géré par message qui traite le message.

Les applications Java EE JM 3.0 et Jakarta EE utilisent des beans gérés par message pour traiter les messages de manière asynchrone. Un bean géré par message agit en tant que programme d'écoute de message JMS et il est implémenté par une méthode onMessage() qui définit comment un message est traité. Un bean géré par message est déployé dans le conteneur d'EJB d'un serveur d'applications. La manière selon laquelle un bean géré par message est configuré varie en fonction du serveur d'applications utilisé, mais les informations de configuration doivent spécifier à quel gestionnaire de files d'attente se connecter, comment se connecter à ce dernier, quelle destination surveiller pour les messages et le comportement transactionnel du bean géré par message. Ces informations sont ensuite utilisées par le conteneur d'EJB. Lorsqu'un message répondant aux critères de sélection du bean géré par message arrive à la destination spécifiée, le conteneur d'EJB utilise IBM MQ classes for JMS ou IBM MQ classes for Jakarta Messaging pour extraire le message du gestionnaire de files d'attente, puis le distribue au bean géré par message en appelant sa méthode onMessage().

Relation avec IBM MQ classes for Java

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging et IBM MQ classes for JMS sont des homologues qui utilisent une interface Java commune à l'interface MQI.

La Figure 55, à la page 170 présente la relation entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging et IBM MQ classes for Java.

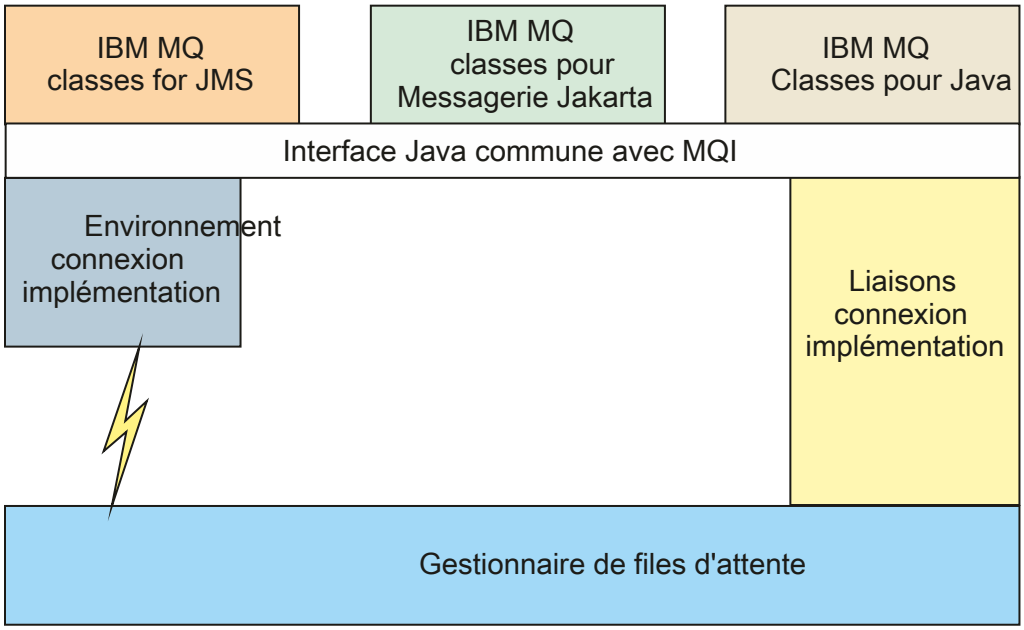


Figure 55. Relation entre IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging et IBM MQ classes for Java

En général, les programmes Java ne doivent utiliser qu'une seule interface pour l'interface avec IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging ou IBM MQ classes for JMS. La combinaison d'interfaces n'est pas prise en charge, à une exception près. Pour assurer la compatibilité avec les versions antérieures à IBM WebSphere MQ 7.0, les classes d'exit de canal qui sont écrites dans Java peuvent encore utiliser les interfaces des IBM MQ classes for Java même si les classes d'exit de canal sont appelées à partir d'IBM MQ classes for JMS. Toutefois, l'utilisation des interfaces IBM MQ classes for Java signifie que vos applications dépendent toujours des éléments suivants:

- JMS 2.0 Le fichier JAR IBM MQ classes for Java , `com.ibm.mq.jar`. Si vous ne souhaitez pas avoir `com.ibm.mq.jar` dans le chemin d'accès aux classes, vous pouvez utiliser à la place l'ensemble d'interfaces du package `com.ibm.mq.exits`.
- JM 3.0 Utilisation du `com.ibm.mq.jakarta.client.jar`lors de l'interaction avec IBM MQ classes for Jakarta Messaging.

Concepts associés

Pourquoi utiliser des classes IBM MQ pour Jakarta Messaging?

Pourquoi utiliser des classes IBM MQ pour JMS?

Pourquoi utiliser des classes IBM MQ pour Java?

Fournisseur de messagerie IBM MQ

Le fournisseur de messagerie IBM MQ possède trois modes de fonctionnement : le mode normal, le mode normal avec restrictions et le mode de migration.

Le fournisseur de messagerie IBM MQ possède trois modes de fonctionnement :

- Le mode normal du fournisseur de messagerie IBM MQ
- Le mode normal avec restrictions du fournisseur de messagerie IBM MQ
- Le mode de migration du fournisseur de messagerie IBM MQ

Le mode de fournisseur de messagerie IBM MQ normal utilise toutes les fonctions d'un gestionnaire de files d'attente IBM MQ pour implémenter JMS. Ce mode est optimisé pour utiliser l'API et la fonctionnalité JMS 2.0 JM 3.0 ou Jakarta Messaging 3.0 .

If :

- Le client spécifie une version de fournisseur de 6 sur un **ConnectionFactory**, le client se comporte d'une manière compatible avec le client fourni avec IBM WebSphere MQ 6.0. Seules les interfaces JMS 1.1 et JMS 2 sont prises en charge, mais certaines fonctionnalités JMS 2, telles que les abonnements partagés, le délai de distribution et l'envoi asynchrone, sont désactivées. Il n'y a pas de partage de connexion.
- Le client spécifie une version de fournisseur de 7 sur un **ConnectionFactory**, les interfaces JMS 1.1 et JMS 2 sont entièrement prises en charge.
- Aucune version de fournisseur n'est spécifiée, une tentative de connexion à la version de fournisseur 7 est effectuée. En cas d'échec, une nouvelle tentative est effectuée avec la version de fournisseur 6.

Si vous voulez vous connecter à IBM Integration Bus à l'aide d'IBM MQ Enterprise Transport, utilisez le mode de migration. Si vous utilisez IBM MQ Real-Time Transport, le mode de migration est automatiquement sélectionné car vous avez sélectionné explicitement des propriétés dans l'objet de fabrique de connexions. La connexion à IBM Integration Bus avec IBM MQ Enterprise Transport suit les règles générales de sélection du mode qui sont décrites dans Configuration de la propriété JMS **PROVIDERVERSION**.

Tâches associées

Configuration des ressources JMS

z/OS IBM MQ for z/OS concepts

Some of the concepts used by IBM MQ for z/OS are unique to the z/OS platform. For example, the logging mechanism, the storage management techniques, unit of recovery disposition, and queue sharing groups are provided only with IBM MQ for z/OS. Use this topic as an introduction to further information about these concepts.

The concepts include an overview of the objects that IBM MQ for z/OSuses, including:

- The queue manager
- The channel initiator
- Shared queues and queue sharing groups
- Intra-group queuing

The following topics also cover various procedures you need, including:

- System definitions on z/OS
- Storage management
- Recovery and restart
- Security concepts in IBM MQ for z/OS

Related concepts

“The queue manager on z/OS” on page 172
Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

“The channel initiator on z/OS” on page 173
The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

“Terms and tasks for managing IBM MQ for z/OS” on page 175
Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

“Shared queues and queue sharing groups” on page 177
You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

“Intra-group queuing” on page 221

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Storage management on z/OS" on page 234
IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

"Logging in IBM MQ for z/OS" on page 238
IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

"Recovery and restart on z/OS" on page 259
Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

"Security concepts in IBM MQ for z/OS" on page 275
Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

"Availability on z/OS" on page 282
IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

"Unit of recovery disposition on z/OS" on page 286
Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Related reference

"System definition on z/OS" on page 249
IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

"Monitoring and statistics on IBM MQ for z/OS" on page 285
IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

z/OS The queue manager on z/OS

Before you can let your application programs use IBM MQ on your z/OS system, you must install the IBM MQ for z/OS product and start a queue manager. The queue manager owns and manages the set of resources that are used by IBM MQ.

The queue manager

A *queue manager* is a program that provides messaging services to applications. Applications that use the Message Queue Interface (MQI) can put messages on queues and get messages from queues. The queue manager ensures that messages are sent to the correct queue or are routed to another queue manager. The queue manager processes both the MQI calls that are issued to it, and the commands that are submitted to it (from whatever source). The queue manager generates the appropriate completion codes for each call or command.

The resources managed by the queue manager include:

- Page sets that hold the IBM MQ object definitions and message data
- Logs that are used to recover messages and objects in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the IBM MQ API
- The IBM MQ channel initiator, which allows communication between IBM MQ on your z/OS system and other systems

The queue manager has a name, and applications can connect to it using this name.

Figure 56 on page 173 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

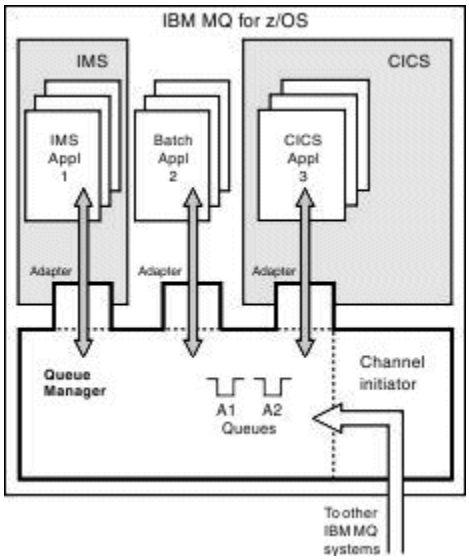


Figure 56. Overview of IBM MQ for z/OS

The queue manager subsystem on z/OS

On z/OS, IBM MQ runs as a z/OS subsystem that is started at IPL time. In the subsystem, the queue manager is started by executing a JCL procedure that specifies the z/OS data sets that contain information about the logs, and that hold object definitions and message data (the page sets). The subsystem and the queue manager have the same name, of up to four characters. All queue managers in your network must have unique names, even if they are on different systems, sysplexes, or platforms.

z/OS The channel initiator on z/OS

The channel initiator provides and manages resources that enable IBM MQ distributed queuing. IBM MQ uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In IBM MQ for z/OS, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as a z/OS address space under the control of the queue manager. There can be only a single channel initiator connected to a queue manager and it is run inside the same z/OS image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 57 on page 174 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX and Windows are placed on the local queue, from where they are retrieved by an application. Reply messages are returned by a similar route.

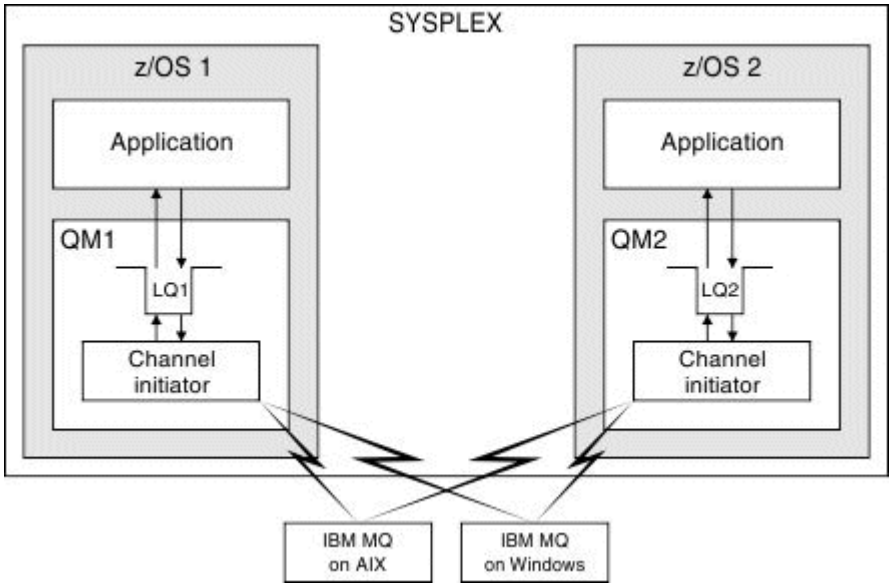


Figure 57. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These processes include:

Listeners
These processes listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

TLS tasks
These are used to perform encryption and decryption and check certificate revocation lists.

z/OS SMF records for the channel initiator

The channel initiator (CHINIT) can produce SMF statistics records and accounting records with information on tasks and channels.

The CHINIT can produce SMF statistics records and accounting records with the following types of information:

- The tasks: dispatcher, adapter, Domain Name Server (DNS), and SSL. These tasks form what is called CHINIT statistics.
- Channels: provides accounting information similar to that available with the DIS CHSTATUS command. This is called channel accounting.

IBM MQ for Multiplatforms provides similar information by writing PCF messages to the SYSTEM.ADMIN.STATISTICS.QUEUE. See Channel statistics message data for further information on how statistics information is recorded on IBM MQ for Multiplatforms.

Statistics data

You can use this information to find out the following information:

- Whether you need more of the CHINIT tasks, such as number of SSL TCBs and how much CPU is used by these tasks.

- The average time for requests on these tasks.
- The longest duration request in the interval, and the time of day this occurred, for DNS and SSL tasks. You can correlate this time of day with problems you may experience with the channel.

Accounting data

You can use this information to monitor channel usage and find out the following information:

- The channels with the highest throughput.
- The rate at which messages were sent, and the rate of sending data in MB/second.
- The achieved batch size. If the achieved batch size is close to the batch size specified for the channel, the channel might be close to its limit for sending messages.

You use the START TRACE and STOP TRACE commands to control the collection of the accounting trace and the statistics trace. You can use the STATCHL and STATACLS options on the channel and queue manager to control whether channels produce SMF data.

z/OS Terms and tasks for managing IBM MQ for z/OS

Use this topic as an introduction to the terminology, and tasks that are specific to IBM MQ for z/OS.

Some of the terms and tasks required for managing IBM MQ for z/OS are specific to the z/OS platform. The following list contains some of these terms and tasks.

- Shared queues
- Page sets and buffer pools
- Logging
- Tailoring the queue manager environment
- Restart and recovery
- Security
- Availability
- Manipulating objects
- Monitoring and statistics
- Application environments

Shared queues

Queues can be *non-shared*, owned by and accessible to only one queue manager, or *shared*, owned by a *queue sharing group*. A queue sharing group consists of a number of queue managers, running within a single z/OS sysplex, that can access the same IBM MQ object definitions and message data concurrently. Within a queue sharing group, the shareable object definitions are stored in a shared Db2 database. The shared queue messages are held inside one or more coupling facility structures (CF structures). If the message data is too large to store directly in the structure (more than 63 KB in size), or if the message is large enough that installation-defined rules select it for offloading, the message control information is still stored in the coupling facility entry, but the message data is offloaded to a shared message data set (SMDS) or to a shared Db2 database. The shared message data sets, the shared Db2 database, and the coupling facility structures are resources that are jointly managed by all of the queue managers in the group.

Pages sets and buffer pools

When a message is put on to a non-shared queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation gets a message from the same queue. If

the message is removed from the queue, space in the page set that holds the data is later freed for reuse. As the number of messages held on a queue increases, so the amount of space used in the page set increases, and as the number of messages on a queue reduces, the space used in the page set reduces.

To reduce the performance cost of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through IBM MQ objects called *buffer pools*.

For more information about page sets and buffer pools, see Storage management.

Logging

Any changes to objects held on page sets, and operations on persistent messages, are recorded as log records. These log records are written to a log data set called the *active log*. The name and size of the active log data set is held in a data set called the *bootstrap data set* (BSDS).

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an *archive log* data set. Information about these actions, including the name of the archive log data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is offloaded to an archive log, and the active log data set is available for reuse.

For more information about the log and bootstrap data sets, see “Logging in IBM MQ for z/OS” on page 238.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager operates are read. In addition, data sets containing IBM MQ commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for IBM MQ to run, and you can tailor these to define or initialize the IBM MQ objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in Db2.

For more information about initialization parameters and system objects, see “System definition on z/OS” on page 249.

Recovery and restart

At any time during the operation of IBM MQ, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set that is the least recently used by a background task within the queue manager.

If the queue manager terminates abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that IBM MQ can recover persistent message data and object changes right up to the point of failure.

If a queue manager that is a member of a queue sharing group encounters a coupling facility failure, the persistent messages on that queue can be recovered only if you have backed up your coupling facility structure.

For more information about recovery and restart, see “Recovery and restart on z/OS” on page 259.

Security

You can use an external security manager, such as Security Server (previously known as RACF) to protect the resources that IBM MQ owns and manages from access by unauthorized users. You can also use Transport Layer Security (TLS) for channel security. TLS is included as part of the IBM MQ product.

For more information about IBM MQ security, see “Security concepts in IBM MQ for z/OS” on page 275.

Availability

There are several features of IBM MQ that are designed to increase system availability in the event of queue manager or communications subsystem failure. For more information about these features, see “Availability on z/OS” on page 282.

Manipulating objects

When the queue manager is running, you can manipulate IBM MQ objects either through a z/OS console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms enable you to define, alter, or delete IBM MQ objects. You can also control and display the status of various IBM MQ and queue manager functions.

For more information about these facilities, see Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS.

You can also manipulate IBM MQ objects using the IBM MQ Explorer, a graphical user interface that provides a visual way of working with queues, queue managers, and other objects.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

For more information about these facilities, see “Monitoring and statistics on IBM MQ for z/OS” on page 285.

Application environments

When the queue manager has started, applications can connect to it and start using the IBM MQ API. These can be CICS, IMS, Batch, or WebSphere Application Server applications. IBM MQ applications can also access applications on CICS and IMS systems that are not aware of IBM MQ, using the CICS and IMS bridges.

For more information about these facilities, see “IBM MQ and other z/OS products” on page 288.

For information about writing IBM MQ applications, see the following documentation:

- Developing applications
- Using C++
- Using IBM MQ classes for Java

z/OS Shared queues and queue sharing groups

You can use shared queues and queue sharing groups, to implement high availability of IBM MQ resources. Shared queues and queue sharing groups are functions unique to IBM MQ for z/OS on the z/OS platform.

This section describes the attributes and benefits, and offers information about how several queue managers can share the same queues and the messages on those queues.

What is a shared queue?

A shared queue is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex.

A queue sharing group

The queue managers that can access the same set of shared queues form a group called a *queue sharing group*.

Any queue manager can access messages

Any queue manager in the queue sharing group can access a shared queue. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue sharing group that does not require channels to be active between queue managers.

IBM MQ supports the offloading of messages to Db2 or a shared message data set (SMDS). The offloading of messages of any size is configurable.

Figure 58 on page 178 shows three queue managers and a coupling facility, forming a queue sharing group. All three queue managers can access the shared queue in the coupling facility.

An application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue sharing group can continue processing the queue if one of the queue managers has a problem.

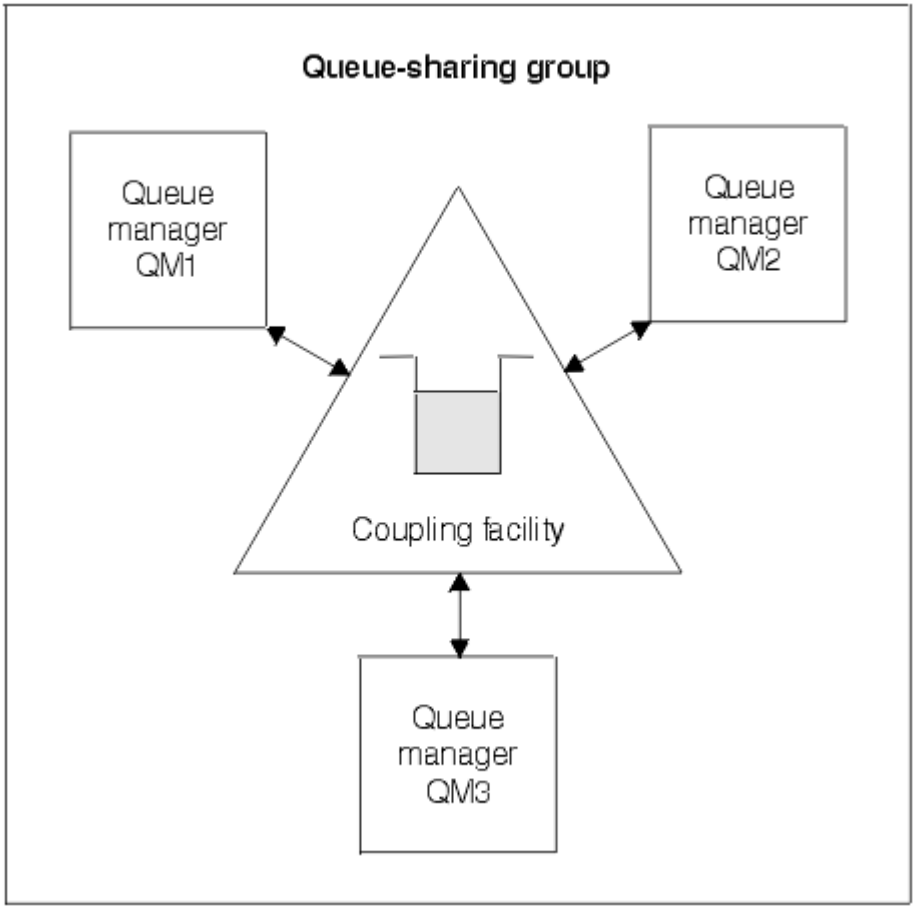


Figure 58. A queue sharing group

Queue definition is shared by all queue managers

Shared queue definitions are stored in the Db2 database table OBJ_B_QUEUE. Because of this, you need to define the queue only once and then it can be accessed by all the queue managers in the queue sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in Page sets).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name exists.

What is a queue sharing group?

A group of queue managers that can access the same shared queues is called a queue sharing group. Each member of the queue sharing group has access to the same set of shared queues.

Queue sharing groups have a name of up to four characters. The name must be unique in your network, and must be different from any queue manager names.

Figure 59 on page 179 illustrates a queue sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue sharing group must also connect to a Db2 system. The Db2 systems must all be in the same Db2 data-sharing group so that the queue managers can access the Db2 shared repository used to hold shared object definitions. These are definitions of any type of IBM MQ object (for example, queues and channels) that are defined only once and then any queue manager in the group can use them. These are called *global* definitions and are described in Private and global definitions.

More than one queue sharing group can reference a particular data-sharing group. You specify the name of the Db2 subsystem and which data-sharing group a queue manager uses in the IBM MQ system parameters at startup.

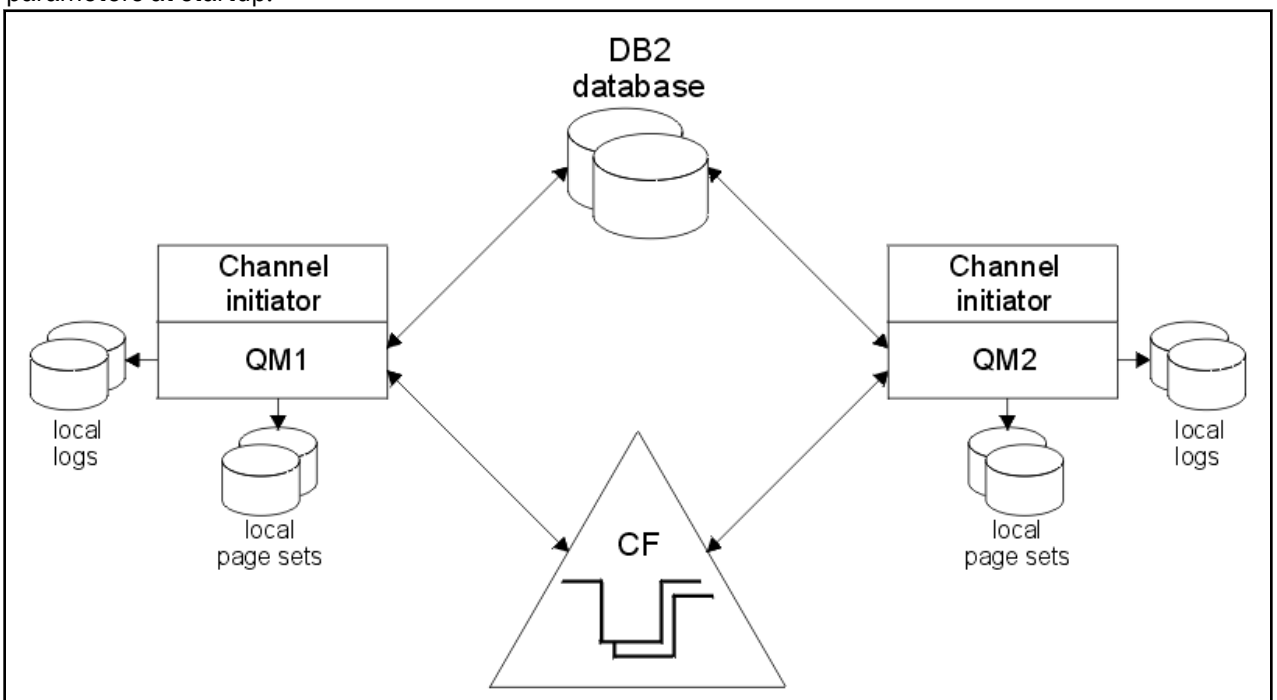


Figure 59. The components of queue managers in a queue sharing group

When a queue manager has joined a queue sharing group, it has access to the shared objects defined for that group, and you can use that queue manager to define new shared objects within the group. If shared queues are defined within the group, you can use this queue manager to put messages to and get messages from those shared queues. Any queue manager in the group can retrieve the messages held on a shared queue.

You can enter an MQSC command once, and have it executed on all queue managers within the queue sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in Directing commands to different queue managers.

When a queue manager runs as a member of a queue sharing group it must be possible to distinguish between IBM MQ objects defined privately to that queue manager and IBM MQ objects defined globally that are available to all queue managers in the queue sharing group. The *queue sharing group disposition* attribute is used for this. This attribute is described in Private and global definitions.

You can define a single set of security profiles that control access to IBM MQ objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to only one queue sharing group, and all queue managers in the group must be in the same sysplex. You specify which queue sharing group the queue manager belongs to in the system parameters at startup.

Related concepts

"Where are shared queue messages held?" on page 180
Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

"Advantages of using shared queues" on page 196
Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

"Distributed queuing and queue sharing groups" on page 215
Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

"Influencing workload distribution with shared queues" on page 219
Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

Related reference

"Where to find more information about shared queues and queue sharing groups" on page 220
Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

z/OS Where are shared queue messages held?

Each message on a shared queue is represented by an entry in a z/OS coupling facility list structure. If the message data is too large to fit in the same entry, it is offloaded either to a shared message data set (SMDS) or to Db2.

If the CF structure has been configured to use System Class Memory (SCM), IBM MQ can use this with no additional configuration.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Shared queue message storage

Messages that are put onto shared queues are not stored on page sets and do not use buffer pools.

The messages in shared queues have entries on list structures in the z/OS coupling facility (CF). Many queue managers in the same sysplex can access those messages using the CF list structure.

The message data for small shared queue messages is normally included in the coupling facility entry. For larger messages, the message data can be stored either in a shared message data set (SMDS), or as one or more binary large objects (BLOBs) in a Db2 table which is shared by a Db2 data sharing group.

Message data exceeding 63 KB is always offloaded to SMDS or Db2. Smaller messages can also optionally be offloaded in the same way to save space in the coupling facility structure. See “Specifying offload options for shared messages” on page 182 for more details.

Messages put on a shared queue are referenced in a coupling facility structure until they are retrieved by an MQGET. Coupling facility operations are used to:

- Search for the next retrievable message
- Lock uncommitted messages on shared queues
- Notify interested queue managers about the arrival of committed messages

MQPUT and MQGET operations on persistent messages are recorded on the log of the queue manager performing that operation. This minimizes the risk of data loss in the event of a coupling facility failure.

The coupling facility

The messages held on shared queues are referenced inside a coupling facility. The coupling facility lies outside any of the z/OS images in the sysplex and is typically configured to run on a different power supply. The coupling facility is therefore resilient to software failures and you can configure it so that it is resilient to hardware failures or power-outages. This means that messages stored in the coupling facility are highly available.

Each coupling facility list structure used by IBM MQ is dedicated to a specific queue sharing group, but a coupling facility can hold structures for more than one queue sharing group. Queue managers in different queue sharing groups cannot share data. Up to 32 queue managers in a queue sharing group can connect to a coupling facility list structure at the same time.

A single coupling facility list structure can contain up to 512 shared queues. The total amount of message data stored in the structure is limited by the structure capacity. However, with **CFLEVEL(5)** you can use the offload parameters to offload data for messages less than 63 KB thereby increasing the number of messages which can be stored in the structure, although each message still requires at least a coupling facility entry plus at least 768 bytes of data, made up of 256 bytes for the entry and 512 bytes for the two elements of header and descriptor.

The size of the list structure is restricted by the following factors:

- It must lie within a single coupling facility.
- It might share the available coupling facility storage with other structures for IBM MQ and other products.

Coupling facility list structures can have storage class memory associated with them. In certain situations this storage class memory can be useful when used with shared queues. See “Use of storage class memory with shared queues” on page 197 for more information.

Planning the CF structure size

If you require guidance on the sizing of your CF structures you can use the MP16: IBM MQ for z/OS Capacity planning and tuning supportpac. You can also use the web-based tool CFSizer, which is provided by IBM to assist with CF sizes.

The CF structure object

The queue manager's use of a coupling facility structure is specified in a CF structure (CFSTRUCT) IBM MQ object.

These structure objects are stored in Db2.

When using z/OS commands or definitions relating to a coupling facility structure, the first four characters of the name of the queue sharing group are required. However, an IBM MQ CFSTRUCT object always exists

within a single queue sharing group, and so its name does not include the first four characters of the name of the queue sharing group. For example, CFSTRUCT(MYDATA) defined in queue sharing group starting with SQ03 would use coupling facility list structure SQ03MYDATA.

CF structures have a CFLEVEL attribute that determines their functional capability:

- 1, 2 - can be used for nonpersistent messages less than 63 KB
- 3 - can be used for persistent and nonpersistent messages less than 63 KB
- 4 - can be used for persistent and nonpersistent messages up to 100 MB
- 5 - can be used for persistent and nonpersistent messages up to 100 MB and selectively offloaded to shared message data sets (SMDS) or Db2.

Note: When using IBM MQ you can encrypt a coupling facility structure. See Encrypting coupling facility structure data for more information.

Backup and recovery of the coupling facility

You can back up coupling facility list structures using the IBM MQ command BACKUP CFSTRUCT. This puts a copy of the persistent messages currently within the CF structure onto the active log data set of the queue manager making the backup, and writes a record of the backup to Db2.

If coupling facility fails, you can use the IBM MQ command RECOVER CFSTRUCT. This uses the backup record from Db2 to locate and restore persistent messages from the backup of the CF structure. Any activity since the last backup is replayed using the logs of all the queue managers in the queue sharing group, and the CF structure is then restored up to the point before the failure.

See the BACKUP CFSTRUCT and RECOVER CFSTRUCT commands for more details.

Related concepts

“Specifying offload options for shared messages” on page 182
You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

“Managing your shared message data set (SMDS) environment” on page 184
If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

z/OS *Specifying offload options for shared messages*

You can choose where the message data for a shared queue message is stored, either in a Db2 table or a shared message data set (SMDS). You can also select which messages are offloaded, based on the size of the message and the current usage of the coupling facility structure (CF).

The message data for shared queues can be offloaded from the coupling facility and stored in either a Db2 table or in an IBM MQ managed data set called a *shared message data set* (SMDS).

For messages larger than the coupling facility entry size of 63 KB, offloading message data to a SMDS can have a significant performance improvement compared with offloading to Db2.

Every shared queue message is still managed using a list entry in a coupling facility structure, but when the message data is offloaded to the SMDS, the coupling facility entry only contains some control information and a list of references to the relevant disk blocks where the message is stored. Using this mechanism means the amount of coupling facility element storage required for each message is only a fraction of the actual size of the message.

Selecting where the shared queue messages are stored

The selection of SMDS or Db2 shared message storage is controlled with the **OFFLOAD(SMDS|DB2)** parameter on the **CFSTRUCT** definition. **OFFLOAD(SMDS)** is the default value.

This parameter also requires the **CFSTRUCT** to use **CFLEVEL(5)** or greater.

The **OFFLOAD** parameter is only valid from **CFLEVEL(5)**. See DEFINE CFSTRUCT for more details.

OFFLOAD(DB2) is supported primarily for migration purposes.

Selecting which shared queue messages are offloaded

Message data is offloaded to SMDS or Db2 based on the size of the message data, and the current usage of the coupling facility structure. There are three rules, and each rule specifies a matching pair of parameters. These parameters are a corresponding coupling facility structure usage threshold percentage (**OFFLDnTH**) and a message size limit (**OFFLDnSZ**).

The current implementation of the three rules is specified using the following pairs of keywords:

- OFFLD1TH and OFFLD1SZ
- OFFLD2TH and OFFLD2SZ
- OFFLD3TH and OFFLD3SZ

Rule pair	Default value	Description
Rule pair 1	OFFLD1TH(70) and OFFLD1SZ(32K)	If the coupling facility structure is more than 70% full offload data for messages exceeding 32 KB
Rule pair 2	OFFLD2TH(80) and OFFLD2SZ(4K)	If the coupling facility structure is more than 80% full offload data for messages exceeding 4 KB
Rule pair 3	OFFLD3TH(90) and OFFLD3SZ(0K)	If the coupling facility structure is more than 90% full offload data for messages exceeding 0 KB (all messages)

If an offload rule has the OFFLD *x* SZ value of 64K this indicates that the rule is not in effect. In this case messages will only be offloaded if another offload rule is in effect, or if the message is greater than 63.75 KB and so, too large to store in the structure.

Each message which is offloaded still requires 0.75 KB of storage in the coupling facility.

The three offload rules which can be specified for each structure are intended to be used as follows.

- Performance
  - When there is plenty of space in the application structure, message data should only be offloaded if it is too large to store in the structure, or if it exceeds some lower message size threshold such that the performance value of storing it in the structure is not worth the amount of structure space that it would need.
  - If a specific message size threshold is required, it is conventionally specified using the first offload rule.
- Capacity
  - When there is very little space in the application structure, the maximum amount of message data should be offloaded so as to make the best use of the remaining space.
  - The third offload rule is conventionally used to indicate that when the structure is nearly full, most messages should be offloaded, so the entries in the application structure will be typically of the minimum size (requiring about 0.75K bytes).
  - The usage threshold parameter should be chosen based on the application structure size and the maximum anticipated backlog. For example, if the maximum anticipated backlog is 1M messages, then the amount of structure storage required for this number of messages is about 0.75G bytes. This means for example that if the structure is about 10G bytes, the usage threshold for offloading all messages must be set to 92% or lower.

  - Structure space is divided into elements and entries, and even though there may be enough space overall, one of these may run out before the other. The system provides AUTOALTER capabilities to adjust the ratio when necessary, but this is not very sensitive, so the amount of space actually available may be somewhat less. It may be better therefore to aim to use not more than 90% of the maximum structure space, so in the previous example, the usage threshold for offloading all messages would be better set around 80%.
- Cushioned transition:
  - As the amount of space left in the coupling facility structure decreases, it would be undesirable to have a large sudden change in the performance characteristics. It is also undesirable for coupling facility management to have a sudden threshold change in the typical ratio of entries to elements being used.
  - The second offload rule is conventionally used to provide some intermediate cushion between the performance and capacity biased offload rules. It can be set to cause a significant increase in offload activity when the space used in the coupling facility structure exceeds an intermediate threshold. This means that the remaining space is used up more slowly, and gives the coupling facility automatic alter processing more time to adapt to the higher usage levels.

If the coupling facility structure cannot be expanded, and there is a need to store at least some predetermined number of messages, the third rule can be modified as necessary to ensure that offloading of data for all messages starts at an appropriate threshold to ensure space is reserved for that predetermined number of messages.

For example, if the coupling facility structure size is 4 GB, and the predetermined number of messages is 1 million, then 1,000,000 * 0.75 KB are needed, which is 768 MB, 18.75% of 4 GB. In this case the threshold for offloading all messages needs to be set around 80% rather than 90%. This gives parameters OFFLD3TH(80) and OFFLD3SZ(0K) . The other offload parameters would also need to be adjusted.

If it is found that offloading very small messages has a significant performance impact, but the relative impact is less for larger messages, then the usage thresholds for the other rules can be reduced to offload larger messages earlier, leaving more space in the structure for small messages before they need to be offloaded.

For example, if messages exceeding 32KB occur frequently but the elapsed time performance for offloading them (as determined from RMF statistics or application performance) is very similar to that for keeping them in the coupling facility, then the threshold for the first rule could be set to 0% to offload all such messages. This gives parameters OFFLD1TH(0) and OFFLD1SZ(32K). Again the other offload parameters would need to be adjusted.

If there are many messages around specific intermediate sizes, such as 16 KB and 6 KB, then it might be useful to change the message size option for the second rule so that the larger ones get offloaded at a fairly low usage threshold, saving a significant amount of space, but the smaller ones still get stored only in the coupling facility.

z/OS *Managing your shared message data set (SMDS) environment*

If you select shared message data sets to offload large messages then you must also be aware of the information that IBM MQ uses to manage these data sets and the commands used to work with this information. Use this topic to understand how to manage shared message data sets.

SMDS objects

The properties and status of each shared message data set are tracked in a shared SMDS object which can be updated through any queue manager in the queue sharing group.

There is one shared message data set for each queue manager that can access each coupling facility application structure. The shared message data set is identified by the owning queue manager name, specified using the SMDS keyword, and by the application structure name, specified using the CFSTRUCT keyword.

Note: When defining SMDS data sets for a structure, you must have one for each queue manager.

The SMDS object is stored in an array (with one entry per queue manager in the group) which forms an extension of the corresponding CFSTRUCT object stored in Db2.

There is no command to DEFINE or DELETE the SMDS object because it is created or deleted as part of the CFSTRUCT object, but there is a command to ALTER it to change settings for an individual owning queue manager.

For further information on SMDS commands, see "SMDS related commands" on page 195

SMDSCONN information

It is possible for a shared message data set to be in a normal state, but for one or more queue managers to be unable to connect to it, for example because of a problem with a security definition or with direct access device connectivity. It is therefore necessary for each queue manager to keep track of connection status, and availability information for each shared message data set, indicating for example whether it can currently connect to it, and if not why not.

The SMDSCONN information represents a queue manager connection to a shared message data set. As for the shared message data set itself, it is identified by the queue manager which owns the shared message data set (as specified on the SMDS keyword for the shared object itself) combined with the CFSTRUCT name.

There is no parameter to identify the connecting queue manager because commands addressed to a specific queue manager can only refer to SMDSCONN information for that same queue manager.

The SMDSCONN information entries are maintained in main storage in the owning queue manager, and are re-created when the queue manager is restarted. However, if a connection from an individual queue manager has been explicitly stopped, this information is also stored as a flag in a connection array in the corresponding CFSTRUCT or SMDS object, so that it persists across a queue manager restart.

Status and availability information

Status information indicates the state of a resource or connection (for example, whether it is not yet being used, is in normal use or is in need of recovery). It is usually described using the STATUS keyword. The possible values depend on the type of object.

Status information is normally updated automatically, for example when an error is detected while using the resource or connection. However, in some cases a command can also be used to update the status, to allow for cases when it is not possible for a queue manager to determine the correct status automatically.

Availability information indicates whether the resource or connection can be used, and is usually primarily determined by the status information. For the resource or connection types used in shared message data set support, three levels of availability are implemented:

Available
: This means that the resource is available to be used normally. This does not necessarily mean that it is in use at present (which can be determined instead from the STATUS value). For a data set, if it requires restart processing, this allows the owning queue manager to open it, but other queue managers must wait until the data set is back in the ACTIVE state.

Unavailable because of error
: This means that the resource has been made unavailable automatically because of an error and is not expected to be available again until some form of repair or recovery processing has been performed. However, attempts to make it available again are permitted without operator intervention. Such an attempt can also be triggered by a command to mark the resource as enabled, or a command which changes the status in such a way as to indicate that recovery processing has been completed.

  The reason that the resource has been made unavailable is normally obvious from the related STATUS value, but in some cases there may be other reasons to make the resource unavailable, in which case a separate REASON value is provided to indicate the reason.

Unavailable because of operator command
This means that access to the resource has been explicitly disabled by a command. It can only be made available by using a command to enable it again.

SMDS availability

For the shared SMDS object, the availability is described by the ACCESS keyword, with the possible values ENABLED, SUSPENDED and DISABLED.

The availability can be updated using a **RESET SMDS** command for the relevant shared object from any queue manager in the group to set ACCESS(ENABLED) or ACCESS(DISABLED).

If the availability was previously ACCESS(SUSPENDED), changing it to ACCESS(ENABLED) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to ACCESS(SUSPENDED).

SMDSCONN availability

For a local SMDSCONN information entry, the availability is described by the AVAIL keyword, with the possible values NORMAL, ERROR or STOPPED. The availability can be updated using a **START SMDSCONN** or **STOP SMDSCONN** command addressed to a specific queue manager to enable or disable its connection.

If the availability was previously AVAIL(ERROR), changing it to AVAIL(NORMAL) will trigger a new attempt to use the shared message data set, but if the previous error is still present, the availability will be reset back to AVAIL(ERROR).

Shared message data set shared status and availability

The availability of each shared message data set is managed within the group using shared status information, which can be displayed using the **DISPLAY CFSTATUS** command with TYPE(SMDS). This displays status information for each queue manager that has activated a data set for each structure. Each data set can be in one of the following states:

NOTFOUND
This means that the corresponding data set has not yet been activated. This status only appears when a specific queue manager is specified, as data sets which have not been activated are skipped when all queue managers are selected.

NEW
The data set is being opened and initialized for the first time, ready to be made active.

ACTIVE
This means that the data set is fully available and should be allocated and opened by all active queue managers for the structure.

FAILED
This means the data set is not available at all (except for recovery processing) and must be closed and deallocated by all queue managers.

INRECOVER
This means that media recovery (using RECOVER CFSTRUCT) is in progress for this data set.

RECOVERED
This indicates that a command has been issued to switch a failed data set back to the active state, but further restart processing is required which is not yet complete, so the data set can only be opened by the owning queue manager for restart processing.

EMPTY
The data set contains no messages. The data set is put into this state if it is closed normally by the owning queue manager, at a time when it does not contain any messages. It can also be put into EMPTY state when the previous data set contents are to be discarded because the application structure has been emptied (using **RECOVER CFSTRUCT** with TYPE PURGE or, for a nonrecoverable structure only, by deleting the previous instance of the structure). The next time the data set is opened by its owning queue manager, the space map is reset to empty, and the status is changed to ACTIVE. As the previous data set contents are no longer required, a data set in this state can be

replaced with a newly allocated data set, for example to change the space allocation or move it to another volume.

The command output includes the date and time at which recovery logging was enabled, if any, and the date and time at which the data set failed, if it is not currently active.

A shared message data set can be put into a FAILED state either by a **RESET SMDS** command or automatically when any of the following types of error are detected:

- The data set cannot be allocated or opened by the owning queue manager.
- Validation of the data set header fails after it has been successfully opened by any queue manager.
- A permanent I/O error occurs when the owning queue manager is reading or writing data.
- A permanent I/O error occurs when another queue manager is reading data from a data set which had successfully completed open processing and validation.

When a data set is in the FAILED or INRECOVER state, it not available for normal use, so if the availability state is ACCESS(ENABLED) it is changed to ACCESS(SUSPENDED).

If a data set has been put into the the FAILED state but no media recovery is required, for example because the data was still valid but the storage device was temporarily offline, then the **RESET SMDS** command can be used to request changing the status directly to the RECOVERED state.

When the data set enters the RECOVERED state, either on completion of recovery processing or as a result of the **RESET SMDS** command, then it is ready to be used again once restart processing has been completed. If it was in the ACCESS(SUSPENDED) state, it is automatically switched back to the ACCESS(ENABLED) state, which allows the owning queue manager to perform restart processing. When restart processing completes, the state is changed to ACTIVE and all other queue managers can then connect to the data set again.

Shared message data set connection status and availability

Each queue manager maintains local status and availability information for its connection to each shared message data set owned by itself and by other queue managers in the group. This information can be displayed using the **DISPLAY SMDSCONN** command.

If it is unable to access a shared message data set in the ACTIVE state which belongs to another queue manager it flags the connection as being unavailable from its own point of view.

If the error definitely indicates a problem with the data set itself, the queue manager also automatically changes the shared status to indicate that the data set is now in a FAILED state. However, if the error could be caused by an environmental problem, such as not being authorized to open the data set, the queue manager issues error messages and treats the data set as being unavailable, but it does not modify the shared data set status. If the environmental error turns out to be a problem with the data set anyway (for example it has been allocated on a device which cannot be accessed by some of the queue managers) then an operator can use the RESET SMDS command specifying STATUS(FAILED) to allow the data set to be recovered or repaired as necessary.

If a connection to a shared message data set could not be established but the data set appears to be valid, a new attempt to use it can be triggered by issuing a **START SMDSCONN** command for the owning queue manager.

If there is an operational need to terminate the connection between a specific queue manager and a data set temporarily, but the data set itself is not damaged, then the data set can be closed and deallocated using the **STOP SMDSCONN** command. If the data set is in use, the queue manager will close it normally (although any requests for data in that data set will be rejected with a return code). If it is the owned data set, the queue manager will save the space map during CLOSE processing, avoiding the need for restart processing.

If a data set needs to be taken out of service temporarily from all queue managers (for example to move it) but is not damaged, then it is best to use **STOP SMDSCONN** for the relevant data set with the option CMDSCOPE(*) to stop the queue managers using it first, as this will avoid the need for restart processing when the data set is brought back into service. In contrast, if the data set is marked as FAILED this tells

queue managers that they must stop using it immediately, which means that the space map will not be saved and will need to be rebuilt by restart processing.

Access to any shared message data sets previously in the ACCESS(SUSPENDED) state will be retried if the queue manager is restarted.

Shared message data set recovery logging

Persistent shared messages are logged for media recovery purposes. This means that the messages can be recovered after any failure of coupling facility structures or shared message data sets, provided that the recovery logs are still intact. Persistent messages can also be re-created from the recovery logs at another site for disaster recovery purposes.

When the message data is written to a shared message data set, each block written to the data set is logged separately followed by the message entry (including the data map) as written to the coupling facility. The recovery process always recovers the coupling facility structure, but it does not need to recover individual shared message data sets except when the data set status is FAILED, or when the status is ACTIVE but the data set header record is no longer valid, indicating that the data set has been re-created. A data set is not selected for recovery if its status is ACTIVE and the data set header is still valid, nor if its status is EMPTY, indicating that no messages were stored in it at the time of the failure.

Shared message data set backups

When BACKUP CFSTRUCT is used to make a backup of the shared messages in an application structure, any data for persistent messages stored in shared message data sets is backed up at the same time, as for persistent shared messages previously stored in DB.

Shared message data set recovery

If a shared message data set is corrupted or lost, then it needs to be put into the FAILED state to stop the queue managers from using it until it has been repaired. This normally happens automatically, but can also be done using the **RESET SMDS** command specifying STATUS(FAILED).

If the shared message data set contained any persistent messages, these can be recovered using the RECOVER CFSTRUCT command. This command first restores any persistent message data for that shared message data set from the most recent BACKUP CFSTRUCT command, then applies all logged changes since that time. If no **BACKUP CFSTRUCT** command has been performed since the time that the data set was first activated, it is reset to empty then all changes since activation are applied.

If the CFSTRUCT contents and all of the shared message data sets are unavailable, for example in a disaster recovery situation, they can all be recovered in a single **RECOVER CFSTRUCT** command.

If a shared message data set is damaged but recovery was not active for the CFSTRUCT, or the log containing the latest BACKUP CFSTRUCT is unavailable or unusable, then the messages offloaded to that data set cannot be recovered. In this case, the **RECOVER CFSTRUCT** command with the parameter TYPE(PURGE) can be used to mark the shared message data set as empty and delete any messages from the structure which had data stored in that data set.

When the **RECOVER CFSTRUCT** command is issued, the shared message data set status is changed from FAILED to INRECOVER. If recovery completes successfully, the status is automatically changed to RECOVERED, otherwise it changes back to FAILED.

When the data set is changed to the RECOVERED state, this tells the owning queue manager that it can now try to open the data set and perform restart processing.

Shared message data set recovery and syncpoints

The shared message data set recovery process reapplies the changes for all complete log records up to the end of the log, regardless of syncpoints.

If changes were made within syncpoint, restart or recovery processing for the CFSTRUCT may result in backing out of uncommitted requests, so some of the recovered changes may not actually be used, but there is no harm in recovering them anyway.

It is also possible that an uncommitted MQPUT message may have been written to the structure but the corresponding data may not have been written to the data set or the log (as I/O completion is only forced at the start of syncpoint processing). This is harmless because restart processing will back out the message entry in the structure, so the fact that it refers to unrecovered data does not matter.

Shared message data set restart processing

If a queue manager connection to a CFSTRUCT terminates normally, the queue manager writes out the free block space map for each shared message data set to a checkpoint area within the data set, just before the data set is closed. The space map can then be read in again at connection restart time, provided that neither the CFSTRUCT nor the shared message data set require any recovery processing before the next restart.

However, if a queue manager terminates abnormally, or the structure or data set require any recovery processing, then additional processing is required to rebuild the space map dynamically when the queue manager connection to the structure is restarted.

Provided that the data set itself did not need to be recovered, queue manager restart simply scans the current contents of the structure to locate references to message data owned by the current queue manager, and marks the relevant data blocks as owned in the space map. Other queue managers can continue to use the structure and read the data owned by the restarting queue manager while the space map is being rebuilt.

Shared message data set restart after recovery

If a shared message data set had to be recovered from a backup, then all nonpersistent messages stored in the data set will have been lost, and if the data set was recovered using TYPE(PURGE) then all messages stored in the data set will have been lost. Until recovery has completed, the data set will be marked as FAILED or INRECOVER so any attempt to read one of the affected messages from another queue manager returns an error code indicating that the data set is temporarily unavailable.

When the data set has been recovered, the status is changed to RECOVERED, which allows the owning queue manager to open it for restart processing, but the data set remains unavailable to other queue managers. Queue manager restart scans the structure to rebuild the space map for any remaining messages. The scan also checks for messages for which the data has been lost, and deletes them from the structure (or if necessary flags them as lost, to be deleted later).

The data set status is automatically changed from RECOVERED to ACTIVE when this restart scan completes, at which point other queue managers can start using it again.

Shared message data set usage information

The DISPLAY USAGE command now also shows information about shared message data set space and buffer pool usage for any currently open shared message data sets. This information is displayed if either the new option TYPE(SMDS) or the existing option TYPE(ALL) is specified.

Shared message data performance and capacity considerations

Monitoring data set usage

The current percentage full of each owned shared message data set can be displayed by the **DISPLAY USAGE** command with the option **TYPE(SMDS)**.

The queue manager will normally automatically expand a shared message data set when it reaches 90% full, provided that the option **DSEXPAND(YES)** is in effect for the SMDS definition. This applies when either the SMDS option is set to **DSEXPAND(YES)** or the SMDS option is set to **DSEXPAND(DEFAULT)** and the CFSTRUCT default option is set to **DSEXPAND(YES)**.

If the expansion attempt fails because no secondary allocation size was specified when the data set was created (giving message IEC070I with reason code 203) the queue manager repeats the expansion request using an override secondary allocation of approximately 20% of the current size.

When a data set is expanded, the new data set extents are formatted as part of the expansion processing, which can take tens of seconds, or even minutes for very large extents. The new space becomes available for use after formatting is complete and the catalog has been updated to show the new high used control interval.

If new messages are being created very rapidly, it is possible for the existing data set to become full before expansion processing completes. In this case, any request which could not allocate space is temporarily suspended until the expansion attempt completes and the new space becomes available for use. If the expansion was successful the request is retried automatically.

If an expansion attempt fails, because of a lack of available space or because the maximum extents have already been reached, a message is issued giving the reason for the failure, then the override option for the affected SMDS is automatically altered to **DSEXPAND(NO)** to prevent further expansion attempts. In this case, there is a risk that the data set may become full, in which case further action may be needed as described in Data set becomes full.

Monitoring application structure usage

The usage level of an application structure can be displayed using the MVS **DISPLAY XCF,STRUCTURE** command specifying the full name of the application structure (including the queue sharing group prefix). The IXC360I response message shows current usage of elements and entries.

When the structure usage exceeds the **FULLTHRESHOLD** value specified in the CFRM policy, the system issues message IXC585E and may perform automatic **ALTER** actions if specified, which may either alter the entry to element ratio or increase the structure size.

Optimising buffer pool sizes

Each buffer in a shared buffer pool is used to read or write a contiguous range of pages for one message of up to the logical block size. If the message spills over into further blocks, each range of pages in a separate block requires a separate buffer.

Buffers containing message data after a write or read operation are retained in storage and reused using a least-recently-used (LRU) cache scheme so that a request to read the same data again shortly afterwards will not need to go to disk. This provides a significant optimization when shared messages are written and then read back soon afterwards by applications running on the same system. If messages owned by another queue manager are browsed for selection purposes then retrieved, this also avoids the need to reread the message from disk.

This means that the number of buffers required for each application structure is one for each concurrent API request which reads or writes large messages for that application structure plus some number of additional buffers which will be used to save recently accessed data in order to optimize subsequent read accesses.

For shared buffer pools, if there are insufficient buffers, API requests will simply wait if a buffer is not immediately available. However, this situation should be avoided as it can cause significantly degraded performance.

The statistics from the **DISPLAY USAGE** command for shared buffer pools show whether there have been any buffer waits within the current statistics interval, and also shows the lowest number of free buffers (or a negative value indicating the maximum number of threads which waited for a buffer at any time), the number of buffers which have saved data, and the percentage of the times that a buffer request has successfully found saved data on the LRU chain ("LRU hits") instead of having to read it ("LRU misses") [1] .

- If there have been any waits, the number of buffers should be increased.
- If there are many unused buffers, the number of buffers may be reduced to make more storage available in the region for other purposes.

[1] `(Hits / (Hits+Misses))* 100`

- If there are many buffers containing saved data but the proportion of reads which were hits against that saved data is very small, the number of buffers may be reduced if the storage could be better used for other purposes. The number of buffers should not however be reduced by more than the lowest number of free buffers, as that could trigger waits, and it should preferably be high enough that the lowest free buffer count is normally well above zero.

Deleting shared message data sets

The DELETE CFSTRUCT command (which is only allowed when all shared queues in the structure are empty and closed) does not delete the shared message data sets themselves, but they can be deleted in the usual way after this command has completed. If the same data set is to be reused as a shared message data set, it must be reformatted first to reset it to the empty state.

Exception situations for shared message data sets

There are a number of exception situations which can occur during normal use, even when no software or hardware error is present.

Data set becomes full

If a data set becomes full but cannot be expanded, or the expansion attempt fails, applications using the corresponding queue manager to write large messages to the corresponding application structure will receive error 2192, `MQRC_STORAGE_MEDIUM_FULL` (also known as `MQRC_PAGESET_FULL`).

A data set could become full because of a failure in the application which is supposed to process the data, causing a large backlog of messages to accumulate. If so, expanding the data set any further will only be a temporary solution, and it is important to get the processing application going again as soon as possible.

If more space can be made available the **ALTER SMDS** command can then be used to set **DSEXPAND(YES)** or **DSEXPAND(DEFAULT)** (assuming that YES has been set or assumed as the **DSEXPAND** default for the CFSTRUCT definition) to trigger a retry. If the reason for the failure was however that maximum extents had been reached, the new expansion attempt will be rejected with a message and **DSEXPAND(NO)** will be set again. In this case, the only way to expand it any further is to reallocate it, which involves making it temporarily unavailable, as described next.

Data set needs to be moved or reallocated

If a data set needs to be moved or expanded but is otherwise in normal use, it can be taken out of use temporarily to allow it to be moved or reallocated. Any API request which attempts to use the data set while it is unavailable will receive the reason code `MQRC_DATA_SET_NOT_AVAILABLE.`

1. Use the **RESET SMDS** command to mark the data set as **ACCESS(DISABLED)**. This will cause it to be closed normally and deallocated by all currently connected queue managers.
2. Move or reallocate the data set as necessary, copying the old contents to the newly allocated data set, for example using the Access Method Services (AMS) **REPRO** command.

   Do not attempt to preformat the new data set before copying the old data into it, as this would result in the copied data being appended to the end of the formatted data set.
3. Use the **RESET SMDS** command to mark the data set as **ACCESS(ENABLED)** again, to bring it back into use.

   If the old contents are smaller than the size of the new data set, the rest of the space will be preformatted automatically when the new data set is opened.

   If the old contents were larger than the size of the new data set then the queue manager has to scan the messages in the coupling facility structure and rebuild the space map to ensure that none of the active data has been lost. If any reference is found to a data block which is outside the new extents, the data set is marked as **STATUS(FAILED)** and must be repaired by replacing the data

set with one of the correct size and either copying the old data set into it again or using **RECOVER CFSTRUCT** to recover any persistent messages.

Coupling facility structure is low on space

If the coupling facility structure is running out of space, causing message IXC585E, it is worth checking whether the offload rules have been set to ensure that the maximum amount of data is being offloaded in this case. If not, the offload rules can be modified using the **ALTER CFSTRUCT** command.

Error situations for shared message data sets

There are a number of problems to be aware of, which can only be caused by errors and not occur in normal operational situations.

Owned data set cannot be opened

If the queue manager which owns a shared message data set cannot allocate it or open it, or the data set attributes are not supported, the queue manager sets an appropriate **SMDSCONN** status value of **ALLOCFAIL** or **OPENFAIL** and sets the **SMDSCONN** availability to **AVAIL(ERROR)**. It also sets the SMDS availability to **ACCESS(SUSPENDED)**. When the error has been corrected, use the **RESET SMDS** command to set **ACCESS(ENABLED)** to trigger a retry, or issue the **START SMDSCONN** command to the owning queue manager.

Read-only data set cannot be opened

If a queue manager cannot allocate or open a shared message data set owned by another queue manager and marked as **STATUS(ACTIVE)**, it assumes that this is probably due to a specific problem with its connection to the data set (represented by the **SMDSCONN** object) rather than a problem with the data set itself.

It marks the **SMDSCONN** as **STATUS(ALLOCFAIL)** or **STATUS(OPENFAIL)** as appropriate and marks the **SMDSCONN** availability as AVAIL(ERROR) to prevent further attempts to use it.

If the problem can been corrected without affecting the status of the data set itself, use the **START SMDSCONN** command to trigger a retry.

If the problem turns out to be a problem with the data set itself, then the **RESET SMDS** command can be used to mark the data set as **STATUS(FAILED)** until it it has been recovered. When the data set has been recovered, the action of changing the status back to **STATUS(ACTIVE)** will cause other queue managers to be notified. If the **SMDSCONN** is marked as **AVAIL(ERROR)**, it will automatically be changed back to **AVAIL(NORMAL)** to trigger a new attempt to open the data set.

Data set header is corrupt

If the data set was successfully opened but the format of the header information is incorrect, the queue manager closes and deallocates the data set and sets the status set to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**. This allows **RECOVER CFSTRUCT** to be used to recover the contents.

If the error arose because the data set contained residual data from another use and had not been subsequently preformatted, then preformat the data set and use the **RESET SMDS** command to change the status to **STATUS(RECOVERED)**.

Otherwise, the data set must be recovered.

Data set is unexpectedly empty

If the queue manager opens a data set which is marked as **STATUS(ACTIVE)** but finds that it is uninitialized or newly preformatted but otherwise valid, the queue manager closes and deallocates the shared message data set then sets the status to **STATUS(FAILED)** and the availability to **ACCESS(SUSPENDED)**.

Data set has permanent I/O errors

If a data set has permanent I/O errors after successful **OPEN** processing, it probably needs recovery. The queue manager will mark the data set as **STATUS(FAILED)** so that all currently connected queue managers will close and deallocate it.

Data set has recoverable I/O errors

If there are hardware problems with the data set, it is possible that this might result in recoverable I/O errors which are not reflected back to the queue manager but which cause significant performance degradation, and also indicate a risk of permanent I/O errors in the near future.

In this case, the data set may be taken off line for recovery by using the **RESET SMDS** command to mark it as **STATUS(FAILED)**. This will cause it to be closed and deallocated by all queue managers, so for example it could be moved to a new volume before being made available again.

When a data set is made unavailable in this way, the space map is not saved so the queue manager connection restart processing will need to scan the coupling facility structure to locate messages in the data set and rebuild the space map before the data set can be made available again. As an alternative, if the shared message data set is still usable, it set can be made unavailable more gently by using the **RESET SMDS** command to mark the data set **ACCESS(DISABLED)** until it is ready to be made available again.

Data set contents are incorrect

The queue manager cannot detect directly that a data set contains incorrect data or is not up to date, for example because a volume including that data set had to be restored from backups. However, it performs integrity checks which make it very unlikely that any such errors could result in incorrect message data being seen by application programs.

For integrity checking purposes, each message block in the data set is prefixed with a copy of the corresponding coupling facility entry ID, including a unique time stamp, which is checked whenever the message block is read, before the message data is passed to the user program. If the message block prefix does not match the entry ID (and the coupling facility entry was not deleted in the mean time) the message block is assumed to be damaged and unusable.

If the damaged message was persistent, the data set is marked as **STATUS(FAILED)** and the structure contents must be recovered using the **RECOVER CFSTRUCT** command. If the damaged message was non-persistent, there is no way to recover it, so a diagnostic message is issued and the corresponding coupling facility message entry is deleted.

If no saved space map is available when the data set is opened, it is rebuilt by scanning the coupling facility structure for references to data in the data set. During this scan, the queue manager performs a number of actions:

1. The queue manager determines the location of the most recent message (if any) currently remaining in the data set.
2. The queue manager then reads that message from the data set to ensure that the block prefix matches the message entry id

These actions ensure that the queue manager detects any case where the data set is down-level, and marks the data set as FAILED. This check does however tolerate the case where the data set was restored from a previous copy and either no new messages had been added since then or all messages added since that copy had been subsequently read and deleted.

To protect against down-level data in the case where the data set was closed normally, the queue manager performs a number of actions:

1. The queue manager saves a copy of the space map time stamp in the SMDS object within Db2 when the data set is closed normally.
2. The queue manager then checks the space map time stamp is the same, when the data set is opened again

If the time stamp does not match, this suggests that a down-level copy of the data set might have been used, so the queue manager ignores the existing space map and rebuilds it, which will succeed only if no message data was actually lost.

Note: These integrity checks do not guarantee to detect a down-level or damaged data set in all theoretically possible cases. For example, they will not detect a case where the start of a message block is valid but the rest of the data has been partly overwritten.

Recovery scenarios for shared message data sets

This section described shared message data set recovery scenarios.

Data set recovery where no data was lost

In some cases, the correct contents of a failed data set can be restored without needing actual recovery. One example is where a data set contains residual data from a previous use and has not been preformatted again, which can be fixed by preformatting it. Another case is when a data set has been moved, but there was an error in the process of copying the data across, which can be fixed by copying the data again correctly.

In such cases, the corrected data set can be made available again by using the **RESET SMDS** command to set **STATUS(RECOVERED)**. If the availability is currently **ACCESS(SUSPENDED)** this will automatically set it back to **ACCESS(ENABLED)**.

When the owning queue manager is notified that the data set has been recovered, it scans the structure contents to reconstruct the space map, then changes the status to **STATUS(ACTIVE)**. The other queue managers can then start reading the data set again.

Data set recovery with TYPE(NORMAL)

If the contents of a data set have been lost, but the application structure was defined with **RECOVER(YES)** and the appropriate recovery logs are available, the **RECOVER CFSTRUCT** command can be used to recover any persistent messages stored in the structure including persistent message data offloaded to shared message data sets. This command restores the current state using information logged by the **BACKUP CFSTRUCT** command plus all logged changes to persistent messages since the backup time.

The **RECOVER CFSTRUCT** command always recovers all persistent messages in the coupling facility structure together with offloaded message data stored in Db2. For offloaded data stored in shared message data sets, each data set is only selected for recovery processing if it is already marked as **STATUS(FAILED)** or if it is found to be unexpectedly empty or otherwise invalid when opened by recovery processing. Any shared message data set which is marked as active and which passes the validation checks does not need to be recovered, as the existing message data is already correct, but the header is updated to indicate that any saved space map will need to be rebuilt after recovery.

Recovery processing is only possible when the structure has been marked as failed, as the complete contents of the structure need to be reconstructed by recovery processing. However, if at least one shared message data set has been marked as failed the **RECOVER CFSTRUCT** command will automatically mark the structure as failed if necessary to allow recovery processing to proceed.

Recovery may be performed from any queue manager in the queue sharing group, provided that it has been given write access to the relevant data sets.

Only persistent messages are backed up and logged, so normal recovery processing will restore all persistent messages, but will cause any non-persistent messages in the structure to be lost.

When recovery has completed, any data set which was selected for recovery is automatically changed to **STATUS(RECOVERED)**, and if the availability was **ACCESS(SUSPENDED)** it is changed to **ACCESS(ENABLED)**. The queue manager rebuilds the space map for each data set by scanning the messages in the coupling facility, then marks the data set as **STATUS(ACTIVE)** so that it can be used again.

Data set recovery with TYPE(PURGE)

For a recoverable structure, if the data set contents have been lost, but recovery is not possible for some reason, for example because recovery logs are not available or recovery would take too long, the **RECOVER CFSTRUCT** command can be used with **TYPE(PURGE)** to get the structure back to a usable state. This resets the structure to the empty state and marks all of the associated data sets as **STATUS(EMPTY)**.

Deleting the application structure

If a non-recoverable application structure is deleted using the MVS **SETXCF FORCE** command, or as a result of structure failure, then the next time the structure is connected, message CSQE028I is issued to say that the structure has been reset and all existing messages have been discarded, and any existing data sets are automatically reset to **STATUS(EMPTY)** as well. This action makes a non-recoverable structure usable again after loss of data either in the structure or in any of the associated data sets.

If a recoverable application structure is deleted, it will be treated in the same way as if the structure had failed.

Data set recovery fails

If **RECOVER CFSTRUCT** cannot complete for some reason, for example because a log data set is no longer available, or because the queue manager terminated while recovery was in progress, then any data set for which recovery was at least started will be marked in the header to show that partial recovery has been attempted, and the data set will be left in the **STATUS(FAILED)** state.

In this case, the options are to repeat the original recovery request or to recover with **TYPE(PURGE)** instead, discarding the existing data.

If an attempt is made to mark the data set as **STATUS(RECOVERED)** without actually recovering it, then the next time it is opened the queue manager will see that the header indicates incomplete recovery and mark it as **STATUS(FAILED)** again.

Off site disaster recovery

For off site disaster recovery, persistent shared messages can be re-created using only the logs and the Db2 shared objects containing the CFSTRUCT definitions and associated SMDS status information.

After setting up the Db2 tables containing the definitions, the application structure and the shared message data sets can be set up as empty. When a queue manager connects to them and finds that they are unexpectedly empty, it will mark them as failed, after which a single **RECOVER CFSTRUCT** command can be used to recover all persistent messages for all affected structures.

z/OS *SMDS related commands*

This topic describes and provides access to the commands relating to shared message data sets.

Display and alter the **CFSTRUCT** options relating to large message offload (**OFFLOAD** and offload rules) and shared message data sets (**DSGROUP, DSBLOCK, DSBUFS, DSEXPAND)**:

- DISPLAY CFSTRUCT
- DEFINE CFSTRUCT
- ALTER CFSTRUCT
- DELETE CFSTRUCT

Display **CFSTRUCT** status relating to large message offload **(OFFLDUSE)**:

- DISPLAY CFSTATUS

Display and alter override data set options (**DSEXPAND** and **DSBUFS**) for individual queue managers:

- DISPLAY SMDS
- ALTER SMDS

Display or modify the status and availability of the data sets within the queue sharing group:

- DISPLAY CFSTATUS TYPE(SMDS)
- RESET SMDS

Display SMDS data set space usage and buffer usage information for a queue manager:

- DISPLAY USAGE TYPE(SMDS)

Display or modify the status and availability of the connections (**SMDSCONN**) to the data sets from an individual queue manager:

- DISPLAY SMDSCONN
- START SMDSCONN
- STOP SMDSCONN

Backup and recover shared messages, including large message data in SMDS when necessary:

- BACKUP CFSTRUCT
- RECOVER CFSTRUCT

z/OS ***Advantages of using shared queues***

Shared queue allows for IBM MQ applications to be scalable, highly available, and allows workload balancing to be implemented.

The advantages of shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new z/OS image with a queue manager (in the queue sharing group) and a copy of the application.
- It is highly available.
- It naturally performs *pull* workload balancing, based on the available processing capacity of each queue manager in the queue sharing group.

Using shared queues for high availability

The following examples illustrate how you can use a shared queue to increase application availability.

Consider an IBM MQ scenario where client applications running in the network want to make requests of server applications running on z/OS. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

IBM MQ manages the transportation of the request message from the client machine to the server's input queue on z/OS and of the response from the server back to the client. By defining the server's input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue sharing group. This means that you can configure a queue manager on each z/OS image in the sysplex and, by connecting them all to the same queue sharing group, any one of them can access messages on the server's input queue.

Messages on the input queue of the server are still available, even if one of the queue managers terminates abnormally or you have to stop it for administrative reasons. You can take an entire z/OS image offline and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each z/OS image in the sysplex to provide higher server application capacity and availability, as shown in Figure 60 on page 197.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result that is sent back to the client as an IBM MQ message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that you can use to configure the return path. For more information about these options, see “Distributed queuing and queue sharing groups” on page 215.

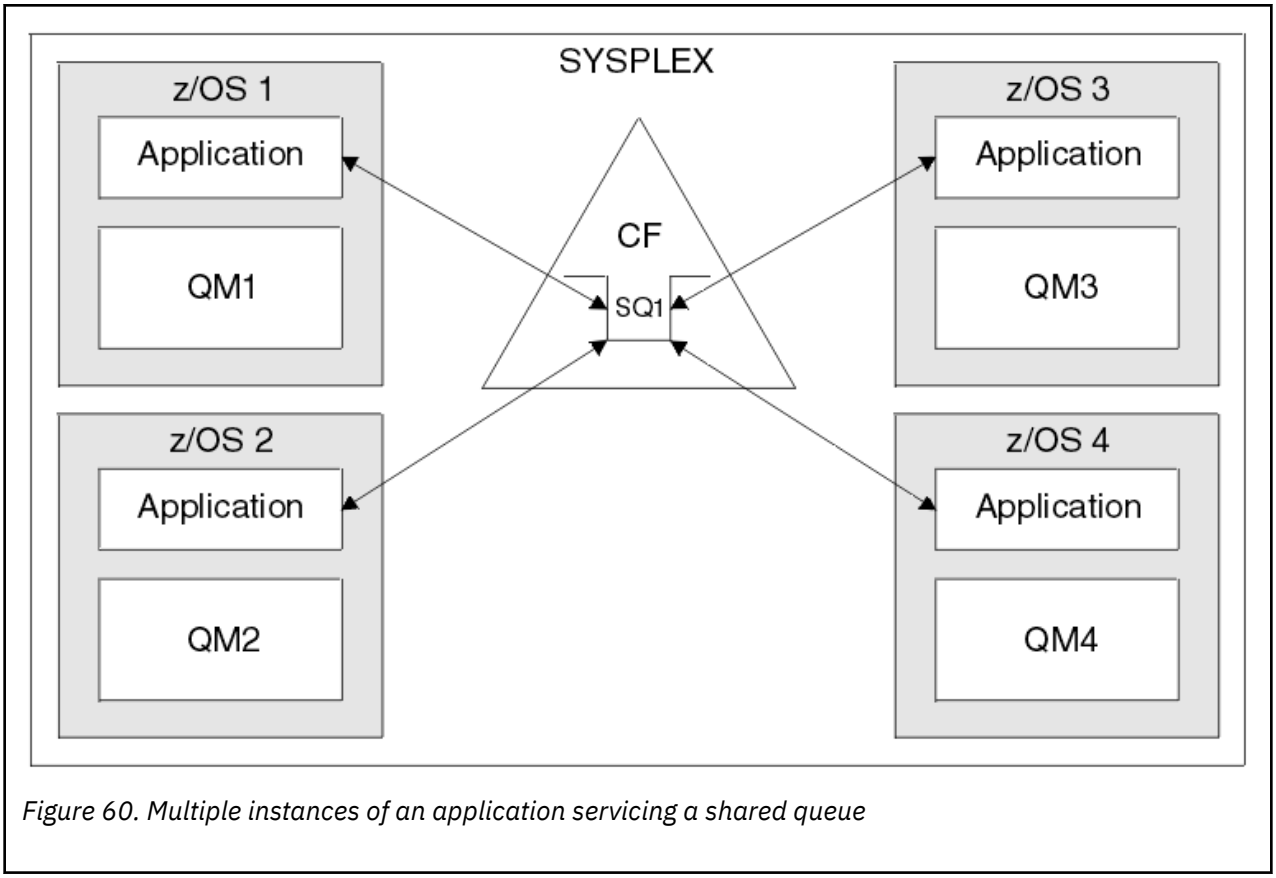


Figure 60. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally and completes units of work for that queue manager that are still pending, where possible. This feature is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in sync point, but has not yet put the response message or committed the unit of work. Another queue manager in the queue sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If IBM MQ cannot resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue sharing group to continue processing that work.

z/OS *Use of storage class memory with shared queues*

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

The z13, zEC12, and zBC12 machines allow the installation of Flash Express cards. These cards contain flash solid-state drives (SSD). After installation, flash storage from the cards can be allocated to one or more LPARs where it is typically known as SCM.

SCM sits between real storage and direct access storage device (DASD) in terms of both I/O latency and cost. Because SCM has no moving parts, it exhibits much lower I/O latencies than DASD.

SCM is also much cheaper than real storage. As a result, a large amount of storage can be installed for a relatively low cost; for example, a pair of Flash Express cards contains 1424 GB of usable storage.

These characteristics mean that SCM is useful when a large amount of data must be taken from real storage in a short period of time, because that data can be written to SCM much quicker than it can be written to DASD. This specific point can be very useful when using coupling facility (CF) list structures containing IBM MQ shared queues.

Why list structures fill up

When a CF structure is defined, it is configured with a SIZE attribute that describes the maximum size of the structure. Because CF structures are always permanently resident in real storage, the sum of the SIZE attributes of the structures that are defined on a CF should be less than the amount of the real storage allocated to the CF.

As a result, there is a constant pressure to keep the SIZE value of any given structure to the minimum value possible, so that more structures can fit into the CF. However, ensuring structures are large enough to achieve their purpose can result in a conflicting pressure, because making a structure too small means that it might fill up, disrupting the applications or subsystems making use of it.

There is a strong need to accurately size a structure based on its expected usage. However, this task is difficult to do because workloads can change over time and accounting for their fluctuations is not easy.

IBM MQ shared queues use CF list structures to store messages. IBM MQ calls CF structures, which contain messages and application structures.

Application structures are referenced using the information stored in IBM MQ CFSTRUCT objects. When a message that is smaller than 63 KB is put on a shared queue, the message is stored entirely in an application structure as a single list entry, and zero or more list elements.

Because IBM MQ shared queues use list structures, the pressures described also affect shared queues. In this case, the maximum number of messages that can be stored on a shared queue is a function of the:

- Size of the messages on the queue
- Maximum size of the structure
- Number of entries and elements available in the structure

Because up to 512 shared queues can use the same structure, and effectively compete for entries and elements, this complicates matters even further

IBM MQ queues are used for the transfer of data between applications, so a common situation is an application putting messages to a queue, when the partner application, which should be getting those messages, is not running.

When this happens the number of messages on the queue increase over time until one or more of the following situations occur:

- The putting application stops putting messages.
- The getting application starts getting messages.
- Existing messages on the queue start expiring, and are removed from the queue.

- The queue reaches its maximum depth in which case an MQRC_Q_FULL reason code is returned to the putting application.
- The structure containing the shared queue reaches its maximum size, or the CF containing the structure runs out of available storage. In either case, an MQRC_STORAGE_MEDIUM_FULL reason code is returned to the putting application.

In the last three cases the queue is full. At this point the putting application has a problem because there is nowhere for its messages to go. The putting application typically solves this problem by using one or more of the following solutions:

- Repeatedly retry putting the message, optionally with a delay between retries.
- Put the messages somewhere else, such as a database or a file. The messages can be accessed later and put to the queue as normal.
- Discard the message if it is nonpersistent.

However, for some classes of applications, for example those with a large volume of incoming messages, or no access to a file system, these solutions are not practical. There is a real need to ensure that queues never, or are extremely unlikely to, fill up in the first place and this is especially pertinent to shared queues.

SMDS and offload rules

The offload rules introduced in IBM WebSphere MQ 7.1 provide a way of reducing the likelihood of an application structure filling up.

Each application structure has three rules associated with it, specified using three pairs of keywords:

- OFFLD1SZ and OFFLD1TH
- OFFLD2SZ and OFFLD2TH
- OFFLD3SZ and OFFLD3TH

Each rule specifies the conditions that must be met for message data to be offloaded to the storage mechanism that is associated with the application structure. Two types of storage mechanisms are currently available:

- Db2
- Group of Virtual Storage Access Method (VSAM) linear data sets, which IBM MQ calls a shared message data set (SMDS).

The following example shows the MQSC command to create an application structure named LIST1, using the DEFINE CFSTRUCT command.

This structure has the default offload rules in place, and uses SMDS as the offload mechanism. This means that when the structure is 70% full (OFFLD1TH), all messages that are 32 KB or larger (OFFLD1SZ) are offloaded to SMDS.

Similarly, when the structure is 80% full (OFFLD2TH) all messages that are 4 KB or larger (OFFLD2SZ) are offloaded. When the structure is 90% full (OFFLD3TH) all messages (OFFLD3SZ) are offloaded.

```
DEFINE CFSTRUCT(LIST1)
CFLEVEL(5)
OFFLOAD(SMDS)
OFFLD1SZ(32K) OFFLD1TH(70)
OFFLD2SZ(4K) OFFLD2TH(80)
OFFLD3SZ(0K) OFFLD3TH(90)
```

An offloaded message is stored in the offload medium, and a pointer to the message is stored in the structure. While the offload rules reduce the chance of the structure filling up, by putting less message data in the structure as it runs out of storage, some data is still written to the structure for each message. That is, the pointer to the offloaded message.

Additionally, the offload rules come with a performance cost. Writing a message to a structure is relatively quick and is largely dominated by the time spent to send the request for the write to the CF. The actual writing to the structure is fast, happening at real storage speeds.

Writing a message to SMDS is much slower because it includes writing to the structure for the message pointer, and writing the message data to SMDS. This second write operation is done at DASD speed and has the potential to add latency. If Db2 is used as the offload mechanism the performance cost is much greater.

z/OS *How storage class memory works with IBM MQ for z/OS*

An overview of the use of storage class memory (SCM) with IBM MQ for z/OS shared queues.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

A coupling facility (CF) that is at CFLEVEL 19, or greater, can have SCM allocated to it. The structures defined in that CF can then be configured to make use of SCM to reduce the chances of the structures filling up (known as a structure full condition). When a structure configured to make use of SCM fills up past a system-determined point, the CF starts moving data from the structure into SCM, which frees space in the structure for new data.

Note: Because SCM itself can fill up, allocating SCM to a structure reduces only the likelihood of a structure full condition, but does not entirely remove the chance of one occurring.

A structure is configured to use SCM by specifying both the **SCMALGORITHM** and **SCMMAXSIZE** keywords in the coupling facility resource manager (CFRM) policy, containing the definition of that structure.

Note that after these keywords are specified, and the CFRM policy is applied, the structure must be rebuilt, or deallocated so that they can take effect.

SCMALGORITHM keyword

Because the input/output speed of SCM is slower than that of real storage, the CF uses an algorithm that is tailored to the expected use of the structure in order to reduce the impact of writing to, or reading from, SCM.

The algorithm is configured by the **SCMALGORITHM** keyword in the CFRM policy for the structure, using the *KEYPRIORITY1* value. Note that you should use the *KEYPRIORITY1* value only with list structures used by IBM MQ shared queues.

The *KEYPRIORITY1* algorithm works by assuming that most applications will get messages from a shared queue in priority order; that is, when an application gets a message, it gets the oldest message with the highest priority.

When a structure starts to fill past the system-defined threshold of 90%, the CF starts asynchronously migrating messages that are least likely to be got next. These are messages with lower priorities that were more recently put on the queue.

This asynchronous migration of messages from the structure into SCM is known as "pre-staging".

Pre-staging reduces the performance cost of using SCM because it reduces the chance of an application being blocked during the occurrence of synchronous input/output to SCM.

In addition to pre-staging, the *KEYPRIORITY1* algorithm also asynchronously brings back messages from SCM and into the structure when sufficient free space is available. For the *KEYPRIORITY1* algorithm, this means that when the structure is less than or equal to 70% full.

The act of bringing messages from SCM into the structure is known as "pre-fetching".

Pre-fetching reduces the likelihood of an application trying to get a message that has been pre-staged to SCM and having to wait while the CF synchronously brings back the message into the structure.

SCMMAXSIZE keyword

The **SCMMAXSIZE** keyword defines the maximum amount of SCM that can be used by a structure. Because SCM is allocated to the structure by the CF when it is required, it is possible to specify a **SCMMAXSIZE** that is greater than the total amount of free SCM available. This is known as "over-committing".

Important: Never over-commit SCM. If you do, the applications that are relying on it will not obtain the behavior that they expect. For example, IBM MQ applications using shared queues might get unexpected MQRC_STORAGE_MEDIUM_FULL reason codes.

The CF uses various data structures to track its use of SCM. These data structures reside in the real storage that is allocated to the CF and, as a result, reduce the amount of real storage that can be used by structures. The storage used by these data structures is known as "augmented space".

When a structure is configured with SCM, a small amount of real storage is allocated from the CF to the structure known as fixed augmented space. This is allocated even if the structure never actually uses any SCM. As data from the structure is stored into SCM, extra dynamic augmented space will be allocated from the spare real storage in the CF.

When the data is removed from SCM, the dynamic augmented space is returned to the CF. Augmented space, either fixed or dynamic, is never taken from the real storage that is allocated to a structure.

In addition to augmented storage, when a structure is configured to use SCM, the amount of control storage used by that structure increases. This means that a list structure configured with SCM can contain fewer entries and elements than a structure of the same size without SCM being configured.

To understand the impact of SCM on new or existing structures, use the CFSizer tool.

A final important point to note is that after data is moved from the structure into SCM, and dynamic augmented space has been used, the structure cannot be altered either manually or automatically.

That is, the amount of storage allocated to the structure cannot be increased or decreased, the entry-to-element ratio that is used by the structure cannot be changed, and so on. To make the structure alterable again, the structure must not have any data stored in SCM and must not be making use of dynamic augmented storage.

z/OS *Why use SCM*

Emergency storage and improved performance are two use cases for using SCM with IBM MQ for z/OS.

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This section introduces the theory behind the two possible scenarios. For further details on how you set up the scenarios, see:

- “Emergency storage - basic configuration” on page 204
- “Improved performance - basic configuration” on page 210

Important: The use of SCM with CF structures is not dependent on any specific version of IBM MQ. However the emergency storage scenario works only with IBM WebSphere MQ 7.1 and later, because it requires SMDS and the offload rules.

Emergency storage

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

Overview

A single shared queue is configured on an application structure. The putting application puts messages onto the shared queue; the getting application gets messages from the shared queue.

During normal running, the queue depth is expected to be close to zero, but a business requirement indicates that the system must be able to tolerate a two-hour outage of the getting application. This means that the shared queue must be able to contain two hours of messages from the putting application.

Currently, this process is achieved by using the default offload rules, and SMDS, so that the size of the structure is minimized, while reducing the performance cost that is associated with offloading.

The rate of messages being sent to the shared queue is expected to double in the short to medium term. Although the requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure.

Because the CF, which contains the application structure, resides on a zEC12 machine, the possibility exists to associate sufficient SCM with the structure to store enough messages so that a two-hour outage can be tolerated.

Consider what happens over a period of time:

1. Initially, the system is in its steady state. Both the putting and getting application are running normally and the queue depth is close to, or at, zero. The result is that the application structure is largely empty.
2. At a certain time, the getting application suffers an unexpected failure and stops. The putting application continues to put messages to the queue and the application structure starts to fill up.
3. After the structure reaches 70% full, the conditions of the first offload rule are met and all messages with a size greater than or equal to 32 KB are offloaded to SMDS.

   See "SMDS and offload rules" on page 199 for an overview of the offload rules.
4. As messages continue to be put to the shared queue, the structure continues to fill (either because of the message data being stored in the structure, or as a result of the pointers to the offloaded messages being stored in the structure).

   When the structure reaches 80% full, the second offload rule starts to apply and messages that are 4 KB or greater are offloaded to SMDS.
5. When the structure is past 90% full, all messages are offloaded to SMDS and only the message pointers are being placed in the structure.

   About this time, the pre-staging algorithm starts to run, and begins moving data from the structure into SCM. Assuming all messages on the queue are the same priority, the newest messages are pre-staged.

   Because all messages are now being offloaded to SMDS, the data being moved into SCM is not actual message data, but instead the pointers to the messages on SMDS.

   As a result, the number of messages that can be stored on the combination of the structure, and the SCM and SMDS associated with the structure, is very large.

   Performance: During this stage of the outage, the putting application can suffer a degree of performance degradation because of having to write to SMDS. In this case, the use of SCM should not be a limiting factor on the putting application in terms of performance. SCM provides extra space to prevent the structure filling up.
6. Eventually the getting application is available again and the outage is over.

   However SCM is still being used by the structure. The getting application starts reading messages off the queue, getting the oldest, highest priority messages first.

   Because these messages were written before the structure started to fill up, they come out entirely from the real storage portion of the structure.
7. As the structure starts to empty, it goes below the threshold at which pre-staging is active, and so pre-staging stops.
8. The structure usage reduces below the point at which the offload rules take effect, so messages are no longer offloaded to SMDS unless they are more than 63 KB.

At about this time, the pre-fetch algorithm starts moving data from SCM in to the structure. Because the getting application gets messages from the queue in the order expected by the SCM algorithms, messages are brought in before the getting application needs them.

The result is that the getting application never needs to wait for messages to be brought in synchronously from SCM.

9. As the getting application continues to move down the queue, it starts retrieving messages that were offloaded to SMDS.
10. Finally, the system is in a steady state again. No messages are stored in SCM or SMDS, and the queue depth is close to zero.

Improved performance

This scenario describes using SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

Description

For this scenario, a putting and getting application communicate through a shared queue which is stored in application structure.

The putting application tends to run in bursts, when it puts a large number of messages in a short amount of time. Then, in an extended period of time, it produces no messages at all.

The getting application sequentially processes each message, and performs complex processing on each one. As a result, most of the time the queue depth is zero, except for when the putting application starts to run, where the queue depth starts increasing as messages are being put faster than they are being got.

The queue depth increases until the putting application stops, and the getting application has enough time to process all the messages on the queue.

Notes:

1. In this scenario, the key factor is performance. The messages being sent to the queue are always less than 63 KB and so never need to be offloaded to SMDS.
2. The application structure has been sized so that it is large enough to contain all of the messages that will be placed on it by the putting application in a single "burst."
3. The offload rules must all be disabled so that, even when the structure starts to fill, the messages are not offloaded to SMDS. This is because the performance costs that are associated with writing messages to, and reading messages from, SMDS are deemed unacceptable.

Over time, the number of messages that the putting application send in a burst must increase by several orders of magnitude. Because the getting application must process each message sequentially, the number of messages on the queue increases to the point where the structure fills up.

At this point, the putting application receives a reason code (MQRC_STORAGE_MEDIUM_FULL) when putting a message, and the put operation fails. The putting application can only briefly tolerate periods when it is unable to put messages to the queue. If the period is too long, the application ends.

Assuming that you do not have the time, or skills available, to rewrite either the putting application or getting application, this problem has three possible solutions:

1. Increase the size of the application structure.
2. Add offload rules to the application structure so that messages are offloaded to SMDS as the queue starts to fill up.
3. Associate SCM with the structure.

The first solution is quick to implement, but not enough real storage is available on the CF.

The second solution might also be quick to implement, but the performance impact of offloading to SMDS is considered too significant to use this option.

The third solution, associating SCM with the structure, provides an acceptable balance of cost and performance.

Associating SCM with a structure results in a higher use of real storage in the CF because of the augmented storage that get operations used. However, the actual amount of real storage will be less than the amount used in the first option.

Another consideration is the cost of SCM. However this cost is much cheaper than real storage. These factors combine to make the third option cheaper than the first option.

Although the third option, potentially, might not perform as well as the first option, the pre-fetch and pre-staging algorithms used by the CF can combine to make the differences in performance acceptable, or in some cases negligible.

Certainly the performance can be much better than using SMDS to offload messages.

Consider what happens over a period of time:

1. Initially, the getting application is active and waiting for messages to be delivered to the shared queue. The putting application is not active and the shared queue is empty.
2. At a certain time, the putting application becomes active, and starts putting a large number of messages to the shared queue. The getting application starts getting the messages, but the queue depth rapidly starts to increase because the getting application is slower than the putting application.

   As a result, the application structure starts to fill up.
3. As the time increases, the putting application is still active. The application structure fills up to approximately 90%.

   This is when the SCM pre-staging algorithm starts to move messages from the structure into SCM, freeing space in the structure.

   Because the getting application gets the oldest, highest priority messages from the queue first, it is always getting messages from the structure and does not need to wait for messages to be brought synchronously from SCM in to the structure.
4. The putting application is still active and putting messages to the shared queue. However, the application never receives an MQRC_STORAGE_MEDIUM_FULL reason code, because enough space exists in SCM to store all the messages that do not fit in the structure.
5. Eventually, the putting application stops because it has no more messages to put.

   The pre-staging algorithm stops because the structure falls below 90% in use, and the getting application continues processing the messages in the queue.
6. As the getting application starts to free space in the structure, the pre-fetch algorithm starts to bring messages back from SCM in to the structure.

   Because the getting application processes messages in the order expected by the pre-fetch algorithm, the getting application never becomes blocked waiting for message data to be brought synchronously from SCM in to the structure.
7. Finally, the getting application processes all the messages on the shared queue, and waits until the next message is available. The structure and SCM are empty of messages.

z/OS *Emergency storage - basic configuration*

How you set up a basic scenario for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

SMDS and message offloading can be used in conjunction with SCM to reduce the likelihood of an MQRC_STORAGE_MEDIUM_FULL reason code being returned to an IBM MQ application during an extended outage.

For example, your enterprise has an application that puts messages onto the queue and an application that gets messages from the queue. During normal running, you expect the queue depth to be close to zero, but a business requirement indicates that the system be able to tolerate a two-hour outage of the application that gets the messages.

This means that the shared queue being used must be able to contain two hours of messages from the putting application. Currently you achieve this by using the default offload rules, and SMDS.

You expect the rate of messages being sent to the shared queue to double in the short to medium term. Although your requirement that the system be able to tolerate a two-hour outage still exists, not enough real storage is available in the CF to double the size of the structure. Because the CF containing the application structure resides on a zEC12 machine, you have the capability of associating sufficient SCM with the structure to store enough messages, so that a two-hour outage can be tolerated

This initial scenario uses a:

- Queue sharing group, `IBM1`, that contains a single queue manager, `CSQ3`. In addition to the administration structure, the queue sharing group defined a single application structure, `SCEN1`.
- Coupling facility (CF) `CF01`, in which the `SCEN1` application structure is stored as the `IBM1SCEN1` structure. This structure has a maximum size of 1 GB.
- Single shared queue, `SCEN1.Q` that the application structure uses.

This configuration is illustrated in Figure 61 on page 205.

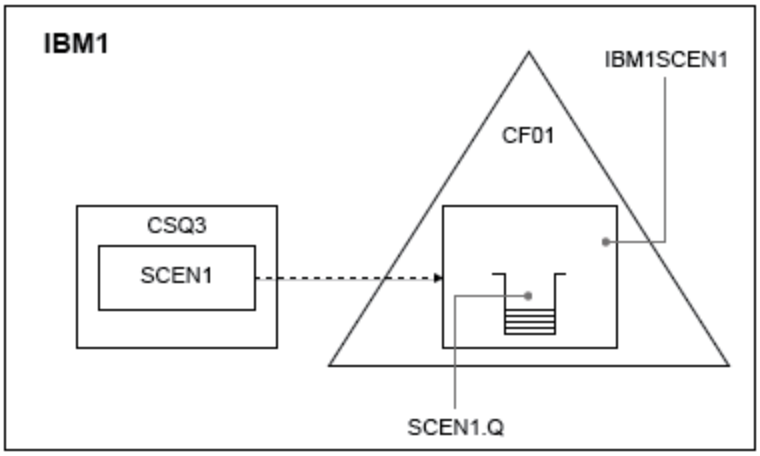


Figure 61. Basic configuration

Furthermore, assume that queue manager `CSQ3` is already the only member of queue sharing group `IBM1`.

You must add the definition for structure `IBM1SCEN1` to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, `CF01`, by specifying `PREFLIST(CF01)`.

Attention: To allow for high availability in your production system, you should include at least two CFs in the `PREFLIST` for any structures that are used by IBM MQ.

Procedure

1. Refresh the CFRM policy by using the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN1
```

Sample CFRM policy for structure IBM1SCEN1:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

2. Verify that the structure has been created correctly, by using the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN1 to create an IBM MQ CFSTRUCT object:

```
DEFINE CFSTRUCT(SCEN1)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 1')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   .
   b. Validate the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN1.Q shared queue, to use the SCEN1 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN1.Q) QSGDISP(SHARED) CFSTRUCT(SCEN1) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN1.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

Check in the output from the command, that the STATUS line shows ALLOCATED.

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

Add SMDS and SCM to the initial structure

Related concepts

“Use of storage class memory with shared queues” on page 197

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SMDS and SCM to the initial structure*

How you add SMDS and SCM for emergency storage on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in "Emergency storage - basic configuration" on page 204. The scenario describes the addition of shared message data sets (SMDS), and then of SCM to the initial structure.

This final configuration is illustrated in Figure 62 on page 207.

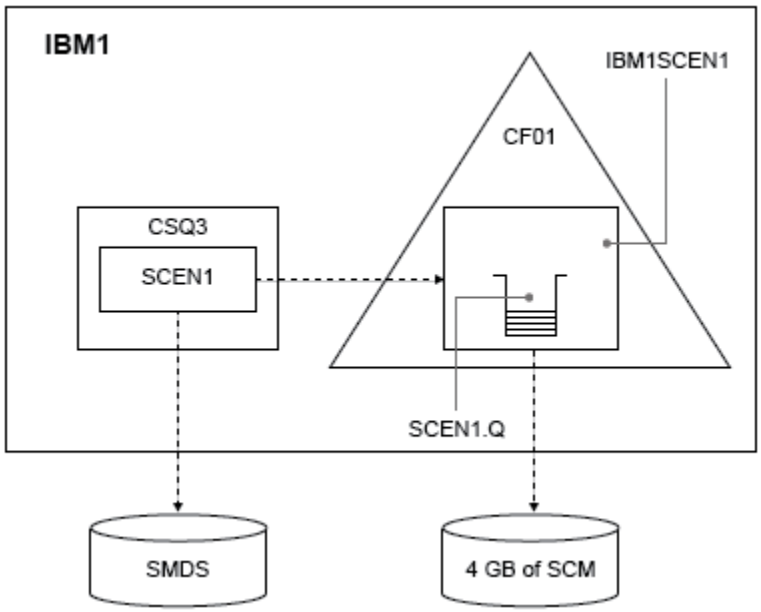


Figure 62. Configuration adding SMDS and SCM for emergency storage

Procedure

1. Create the SMDS data set that the SCEN1 application structure uses, by editing the **CSQ4SMDS** sample JCL, as shown:

```
//CSQ4SMDS JOB NOTIFY=&SYSUID
//*
//* Allocate SMDS
//*
//DEFINE EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER           -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS) -
MEGABYTES(5000 3000)     -
LINEAR            -
SHAREOPTIONS(2 3) )      -
DATA                -
(NAME(CSQSMDS.SCEN1.CSQ3.SMDS.DATA) )
/*
//*
//* Format the SMDS
//*
//FORM   EXEC PGM=CSQJUFMT,COND=(0,NE),REGION=0M
//STEPLIB DD DSN=MQ800.SCSQANLE,DISP=SHR
//     DD DSN=MQ800.SCSQAUTH,DISP=SHR
```

```
//SYSUT1   DD DISP=OLD,DSN=CSQSMDS.SCEN1.CSQ3.SMDS
//SYSPRINT DD SYSOUT=*
```

2. Issue the ALTER CFSTRUCT command to change the SCEN1 application structure, to use SMDS for offloading, implementing the default offload rules:

```
ALTER CFSTRUCT(SCEN1) OFFLOAD(SMDS) OFFLD1SZ(32K) OFFLD2SZ(4K) OFFLD3SZ(0K)
DSGROUP('CSQSMDS.SCEN1.*.SMDS') DSBLOCK(1M)
```

   Note the following:

   - Because SCEN1.Q is the only shared queue on the SCEN1 application structure, the **DSBLOCK** value has been set to 1M, the largest value possible. This should be the most efficient setting for our scenario.
   - Because the messages sent by the putting application are 30 KB, offloading to SMDS does not start until the second offload rule is met, when the structure is 80% full.

3. Run your test application again.

   Note the increased storage of messages on the queue.

4. Add 4 GB of SCM to structure IBM1SCEN1 by carrying out the following procedure:

   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.

   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN1)
SIZE(1024M)
INITSIZE(512M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

5. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=polname
```

6. Rebuild the IBM1SCEN1 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN1
```

Results

You have successfully added SCM to your configuration.

What to do next

Optimize the performance of your system. See “Optimizing storage class memory usage” on page 209 for more information.

z/OS *Optimizing storage class memory usage*

How you improve your use of storage class memory (SCM).

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

Run the following command:

```
D XCF,STR,STRNAME=IBM1SCEN1
```

As the structure was already full with message data, because of the previous tests, part of the rebuild involved pre-staging some of the messages from the structure into SCM. This process was initiated by using the previous command.

The output from this command produces, for example:

```
ACTIVE STRUCTURE
----------------
ALLOCATION TIME: 06/17/2014 09:28:50
CFNAME : CF01
COUPLING FACILITY: 002827.IBM.02.00000000B8D7
PARTITION: 3B CPCID: 00
STORAGE CONFIGURATION ALLOCATED MAXIMUM %
ACTUAL SIZE: 1024 M 1024 M 100
AUGMENTED SPACE: 3 M 142 M 2
STORAGE-CLASS MEMORY: 88 M 4096 M 2
ENTRIES: 120120 1089536 11
ELEMENTS: 240240 15664556 1
SPACE USAGE IN-USE TOTAL %
ENTRIES: 84921 219439 38
ELEMENTS: 2707678 3149050 85
EMCS: 2 282044 0
LOCKS: 1024
SCMHIGHTHRESHOLD : 90
SCMLOWTHRESHOLD : 70
ACTUAL SUBNOTIFYDELAY: 5000
PHYSICAL VERSION: CD5186A0 2BD8B85C
LOGICAL VERSION: CD515C50 CE2ED258
SYSTEM-MANAGED PROCESS LEVEL: 9
XCF GRPNAME : IXCLO053
DISPOSITION : KEEP
ACCESS TIME : NOLIMIT
MAX CONNECTIONS: 32
# CONNECTIONS : 1
CONNECTION NAME ID VERSION  SYSNAME JOBNAME  ASID STATE
--------------- -- -------  ------- -------  ---- ----------------
CSQEIBM1CSQ301  01 00010059 SC61    CSQ3MSTR 0091 ACTIVE
```

Note the following from the output of the command:

- That STORAGE_CLASS MEMORY provides confirmation that a **MAXIMUM** of 4096 MB of SCM has been added to the structure.
- The ALLOCATED figure for the amount of STORAGE-CLASS MEMORY used for pre-staging. There is now free space in the structure where there was none before SCM was added.
- The amount of AUGMENTED SPACE used to track SCM usage.
- The point at which the pre-staging algorithm starts to move data from the structure into SCM is when the structure is 90% full. This is indicated by the non-configurable **SCMHIGHTHRESHOLD** property.
- The point below which the prefetching algorithm starts to move data from SCM into the structure is when the structure is 70% full. This is indicated by the non-configurable **SCMLOWTHRESHOLD** property.

You can now test various ways to optimize the use of SCM. Note the following:

- After SCM is used to store messages, you cannot alter the structure until you have removed all the data from SCM.

  In this case, that means that the entry-to-element ratio is frozen at the value that was in place when SCM was first used. You must carefully ensure that the structure is in the state you want, before the pre-staging algorithm starts moving data into SCM.

- Is the current structure size correct before using SCM?

  For example, have you increased **INITSIZE** from 512 MB to a SIZE of 1 GB?

  If you do not do this, it is possible that although you enabled your structure for auto-alteration, the pre-staging algorithm will start to move data into SCM before the alteration has a chance to start. As a result, the structure is frozen using 512 MB of real storage.
- Is the entry-to-element ratio correct before using SCM?

  The goal of this scenario is to increase the number of offloaded message pointers that can be stored in the structure and SCM as a whole, as well as keeping as many messages entirely in structure storage as possible. Accessing these messages is faster than accessing messages on SMDS.

  Therefore, you need to have a structure that starts with an entry-to-element ratio that is good for storing messages, and then transitions to a ratio that is good for storing message pointers before the prestage algorithm first starts. This transition can be achieved, in part, by making use of the IBM MQ offload rules.

Change the offload rules by issuing the following command:

```
ALTER CFSTRUCT(SCEN1) OFFLD1SZ(0K)
```

You might have to carry out several runs to optimize the entry-to-element ratios.

The following table shows possible improvements in the number of messages put on the queue during the different phases of the emergency storage scenario.

Table 18. Comparison of results for the emergency storage scenario

Test description	Number of messages	Time to fill queue (seconds)
Basic configuration	27,850	3.2
SMDS with default offload rules	205,000	158
SCM with default offload rules	828,610	469
SCM with adjusted offload rules	1,135,775	679

The last row in the table shows that adjusting the offload rules had the required effect.

You need to examine your system to see if you can improve on these figures in any way. For example, you might run out of available SMDS storage. If you can allocate more SMDS storage you should be able to increase the number of messages on the queue quite significantly.

z/OS *Improved performance - basic configuration*

How you set up a basic scenario for improved performance using shared queues on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This scenario describes the use of SCM to increase the number of messages that can be stored on a shared queue without incurring the performance cost of using SMDS.

This initial scenario is very similar to that used for emergency storage and uses a:

- Queue sharing group, IBM1, that contains a single queue manager, CSQ3. In addition to the administration structure, the queue sharing group defined a single application structure, SCEN2.

- Coupling facility (CF) CF01, in which the SCEN2 application structure is stored as the IBM1SCEN2 structure. This structure has a maximum size of 2 GB.
- Single shared queue, SCEN2.Q, which is configured to use the application structure.

This configuration is illustrated in Figure 63 on page 211.

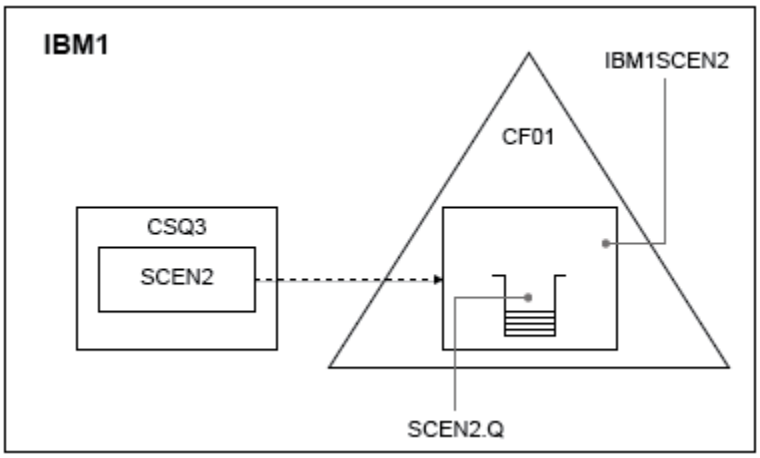


Figure 63. Basic configuration

Furthermore, assume that queue manager CSQ3 is already the only member of queue sharing group IBM1.

You must add the definition for structure IBM1SCEN2 to the coupling facility resource manager (CFRM) policy. For simplicity, the structure is defined so that it can be created in only a single coupling facility, CF01, by specifying PREFLIST(CF01).

Sample CFRM policy for structure IBM1SCEN2:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
```

Both the **INITSIZE** and **SIZE** keywords have the value 2048M so that the structure cannot resize.

Procedure

1. Refresh the CFRM policy by using the following command:

   ```
   SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
   ```

2. Verify that the structure has been created correctly, by using the following command:

   ```
   D XCF,STR,STRNAME=IBM1SCEN2
   ```

   Issuing the preceding command gives the following output:

   ```
   RESPONSE=SC61
   IXC360I 07.58.51 DISPLAY XCF 581
   STRNAME: IBM1SCEN2
   STATUS: NOT ALLOCATED
   POLICY INFORMATION:
   POLICY SIZE : 2048 M
   POLICY INITSIZE: 2048 M
   ```

```
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY

EVENT MANAGEMENT: MESSAGE-BASED   MANAGER SYSTEM NAME: SC53
MANAGEMENT LEVEL : 01050107
```

At this point, your structure has not been allocated, shown by the STATUS line, to the queue sharing group.

3. Configure IBM MQ to make use of the structure defined in the CFRM policy.
   a. Use the DEFINE CFSTRUCT command, with the structure name of SCEN2 to create an IBM MQ CFSTRUCT object.

```
DEFINE CFSTRUCT(SCEN2)
CFCONLOS(TOLERATE)
CFLEVEL(5)
DESCR('Structure for SCM scenario 2')
RECOVER(NO)
RECAUTO(YES)
OFFLOAD(DB2)
OFFLD1SZ(64K) OFFLD1TH(70)
OFFLD2SZ(64K) OFFLD2TH(80)
OFFLD3SZ(64K) OFFLD3TH(90)
```

   b. Check the structure, using the DISPLAY CFSTRUCT command.
   c. Define the SCEN2.Q shared queue, to use the SCEN2 structure, using the following MQSC command:

```
DEFINE QLOCAL(SCEN2.Q) QSGDISP(SHARED) CFSTRUCT(SCEN2) MAXDEPTH(999999999)
```

4. Use IBM MQ Explorer to put a single message to the queue SCEN2.Q and take the message off again.
5. Issue the following command to check that the structure is now allocated:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output from the command, a portion of which is shown, and ensure that the STATUS line shows ALLOCATED.

```
RESPONSE=SC61
IXC360I 08.31.27 DISPLAY XCF 703
STRNAME: IBM1SCEN2
STATUS: ALLOCATED
EVENT MANAGEMENT: MESSAGE-BASED
TYPE: SERIALIZED LIST
POLICY INFORMATION:
POLICY SIZE : 2048 M
POLICY INITSIZE: 2048 M
POLICY MINSIZE : 1536 M
FULLTHRESHOLD : 85
ALLOWAUTOALT : YES
REBUILD PERCENT: N/A
DUPLEX : DISABLED
ALLOWREALLOCATE: YES
PREFERENCE LIST: CF01
ENFORCEORDER : NO
EXCLUSION LIST IS EMPTY
```

Additionally, note the values of the fields in the SPACE USAGE section:

- ENTRIES
- ELEMENTS

- EMCS
- LOCKS

An example of the values follows:

```
SPACE USAGE   IN-USE      TOTAL    %
ENTRIES:      344686     345242   99
ELEMENTS:    6548455    6548467   99
EMCS:              2     780318    0
LOCKS:          1024
```

Results

You have created the basic configuration. You can now obtain an idea of the baseline performance of your configuration using whatever method you select.

What to do next

You should test the basic scenario. As an example, you can use the following three applications, starting the applications in the order shown and, running them concurrently.

1. Use a PCF application to request the current depth (**CURDEPTH**) value for SCEN2.Q every five seconds. The output can be used to plot the depth of the queue over time.
2. A single threaded getting application repeatedly gets messages from SCEN2.Q, using a get with an infinite wait. To simulate processing of the messages that were removed, the getting application pauses for four milliseconds for every ten messages that it removed.
3. A single threaded putting application puts a total of one million 4 KB non-persistent messages to SCEN2.Q. This application does not pause between putting each message so messages are put on SCEN2.Q faster than the getting application can get them.

   As a result, when the putting application is running, the depth of SCEN2.Q increases.

   When structure IBM1SCEN2 is filled, and the putting application receives a MQRC_STORAGE_MEDIUM_FULL reason code, the putting application sleeps for five seconds before attempting to put the next message to the queue.

You can plot the results of the CURDEPTH application over a period of time. You obtain some form of saw-tooth wave output as the putting application pauses to allow the queue to partially empty.

Go to “Adding SCM to the initial structure” on page 213.

Related concepts

“Use of storage class memory with shared queues” on page 197

The use of storage class memory (SCM) can be advantageous when used with IBM MQ for z/OS shared queues.

z/OS *Adding SCM to the initial structure*

How you add SCM for improved performance on IBM MQ.

About this task

Important: IBM z16 is planned to be the last generation of IBM Z® to support the use of Virtual Flash Memory (also known as Storage Class Memory, or SCM) for Coupling Facility images. For more information see: IBM Z and IBM LinuxONE 4Q 2023 Statements of Direction.

As an alternative, you should either use larger structures or offload messages to SMDS.

This part of the task uses the basic configuration described in “Improved performance - basic configuration” on page 210. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 214.

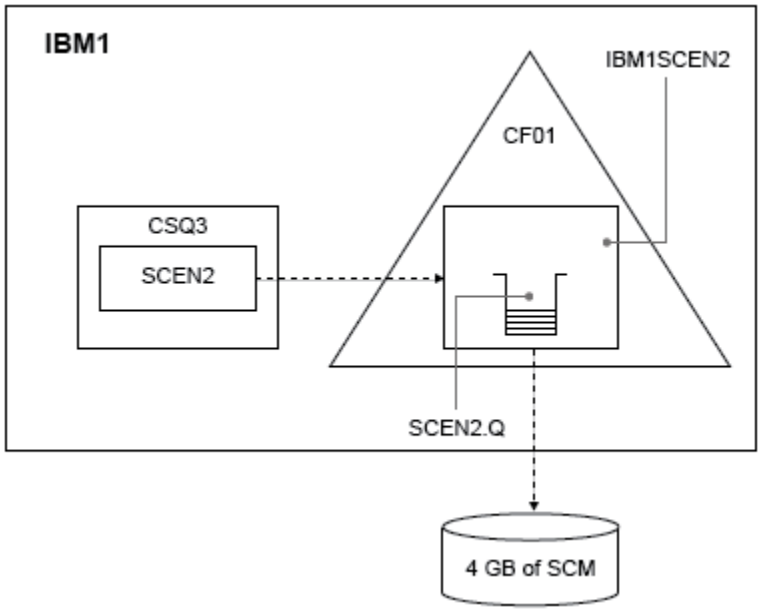


Figure 64. Configuration adding SCM for improved performance

Procedure

1. Add 4 GB of SCM to structure IBM1SCEN2 by carrying out the following procedure:
   a) Check how much SCM is installed, and allocated to CF01, by issuing the following command:

```
D CF,CFNAME=CF01
```

   b) Check the STORAGE-CLASS MEMORY figures in the STORAGE CONFIGURATION section of the displayed output to see the available storage.
   c) Update the CFRM policy with the SCMMAXSIZE and SCMALGORITHM keywords as shown:

```
STRUCTURE
NAME(IBM1SCEN2)
SIZE(2048M)
INITSIZE(2048M)
ALLOWAUTOALT(YES)
FULLTHRESHOLD(85)
PREFLIST(CF01)
ALLOWREALLOCATE(YES)
DUPLEX(DISABLED)
ENFORCEORDER(NO)
SCMMAXSIZE(4G)
SCMALGORITHM(KEYPRIORITY1)
```

2. Activate the CFRM policy by issuing the following command:

```
SETXCF START,POLICY,TYPE=CFRM,POLNAME=IBM1SCEN2
```

3. Rebuild the IBM1SCEN2 structure.

   You must carry out this procedure because the structure was allocated when you made the previous changes.

   Issue the following command to rebuild the structure:

```
SETXCF START,REBUILD,STRNM=IBM1SCEN2
```

4. Issue the following command to confirm the new configuration of the structure:

```
D XCF,STR,STRNAME=IBM1SCEN2
```

Review the output of the command, a portion of which follows:

```
SPACE USAGE     IN-USE       TOTAL      %
ENTRIES:          33         342684     0
ELEMENTS:         48        6503697     0
EMCS:              2         575600     0
LOCKS:                         1024
```

Results

Calculate the change in the use of real storage by the increase in control storage required to use SCM.

- Before SCM is added to the structure, the structure has these totals as shown in "Improved performance - basic configuration" on page 210:
  - 345,242 entries
  - 6,548,467 elements
  - 780,318 EMCS
- After SCM is added to the structure, the structure has these totals:
  - 342,684 entries
  - 6,503,697 elements
  - 575,600 EMCS

Using these figures, after the SCM was added, the structure is reduced in size by:

- 2558 entries
- 44,770 elements
- 204,718 EMCS

The amount of structure storage that is used to manage SCM, is as follows for a 2 GB structure with 4 GB of SCM allocated:

```
(2558 + 44,770 + 204,718) * 256 = 61.5 MB
```

Note that adding more SCM is likely to achieve only a marginal reduction of the size of the structure, because the amount of control storage used to track SCM increases, both as the structure size, and the amount of allocated SCM increases.

What to do next

Repeat the tests described in the final section of "Improved performance - basic configuration" on page 210.

You can plot the results of the revised application over a period of time. Comparing the plot to the one obtained previously, you now obtain an output without a saw-tooth wave, as the putting application no longer has to wait for the queue to partially empty.

For more information, refer to MP16: WebSphere MQ for z/OS - Capacity planning & tuning.

z/OS Distributed queuing and queue sharing groups

Distributed queuing and queue sharing groups are two techniques that you can use to increase the availability of your application systems. Use this topic to find further information about these techniques.

To complement the high availability of messages on shared queues, the distributed queuing component of IBM MQ has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue sharing group.

Figure 65 on page 216 illustrates distributed queuing and queue sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

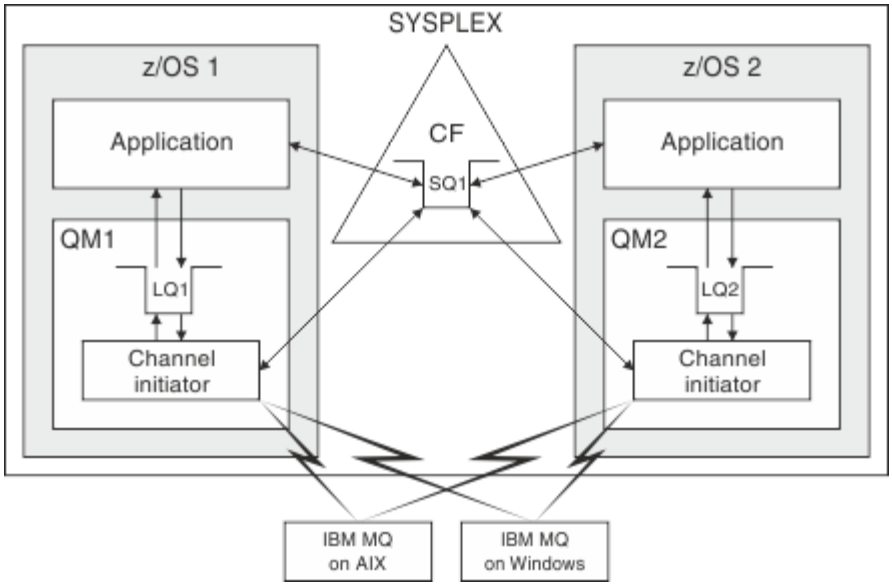


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 216
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 221
This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

"Clusters and queue sharing groups" on page 218
Use this topic to understand how you can use queue sharing groups with clusters.

z/OS *Shared channels*

Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. The network products make a *generic port* available for inbound network connection requests, and the inbound request can be satisfied by connecting to one of the eligible servers.

These networking products include:

- VTAM generic resources
- SYSPLEX Distributor

The channel initiator takes advantage of these products to use the capabilities of shared queues

There are two types of shared channels, *shared inbound channel*, and the *shared outbound channel*.

- Shared inbound channels
- Shared outbound channels

For further information about channels see

- Shared channel summary
- Shared channel status

Shared inbound channels

Each channel initiator in the queue sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network by one of the supporting technologies (VTAM, TCP/IP). Inbound network attach requests to the generic port are dispatched by the network technology, to any one of the listeners in the queue sharing group (QSG) that are listening on the generic port.

You can start a channel on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. You can define a channel definition to be private to a queue manager or stored on the shared repository and so available anywhere (a global definition). This means that you can make a channel definition available on any channel initiator in the queue sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue sharing group and not with an individual queue manager. For example, consider a remote queue manager starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed is still correct because the synchronization is with the queue sharing group.

You can use an inbound channel started through the generic port to put messages to any queue. The remote queue manager does not know whether the target queue is shared or not. If the target queue is a shared queue, the remote queue manager connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue.

If the target queue is a private queue, the messages are put to the private queue owned by which ever queue manager the current instance of the channel is connected to. In this environment, known as *replicated local queues*, each queue manager must have the same set of private queues defined.

Configuring SVRCONN channels for a queue sharing group

The optimal configuration for SVRCONN channels in a queue sharing group is to set up private listeners in each CHINIT which use a different port number from the point to point channels. These listener ports are then used as the 'back-end' resources for a new workload distribution mechanism such as Sysplex Distributor using Virtual IP addresses (VIPA). The external VIPA address is then used as the target address for the CLNTCONN definitions in the network. The SVRCONN channel can be defined with QSGDISP(GROUP) so the same definition is available to all queue managers in the QSG. This configuration avoids using a shared listener, and therefore reduces the performance effect of the queue sharing group maintaining shared channel state, which is not needed for client/server channels.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing for shared outbound channels

An outbound shared channel is eligible for starting on any channel initiator within the queue sharing group, if you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by IBM MQ is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a Db2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from private channels in the following ways:

Private channel
Tied to a single channel initiator.

- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.

- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.

You specify whether a channel is private or shared when you start the channel by using CHLDISP options with the START CHANNEL command. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, IBM MQ performs workload balancing and starts the channel on the most appropriate channel initiator within the queue sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue sharing group maintain a shared channel-status table in Db2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue sharing group.

z/OS *Clusters and queue sharing groups*

Use this topic to understand how you can use queue sharing groups with clusters.

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue sharing group (the shared queue is not advertised as being hosted by the queue sharing group). Clients can start sessions with any members of the queue sharing group to put messages to the same shared queue.

Figure 66 on page 219 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

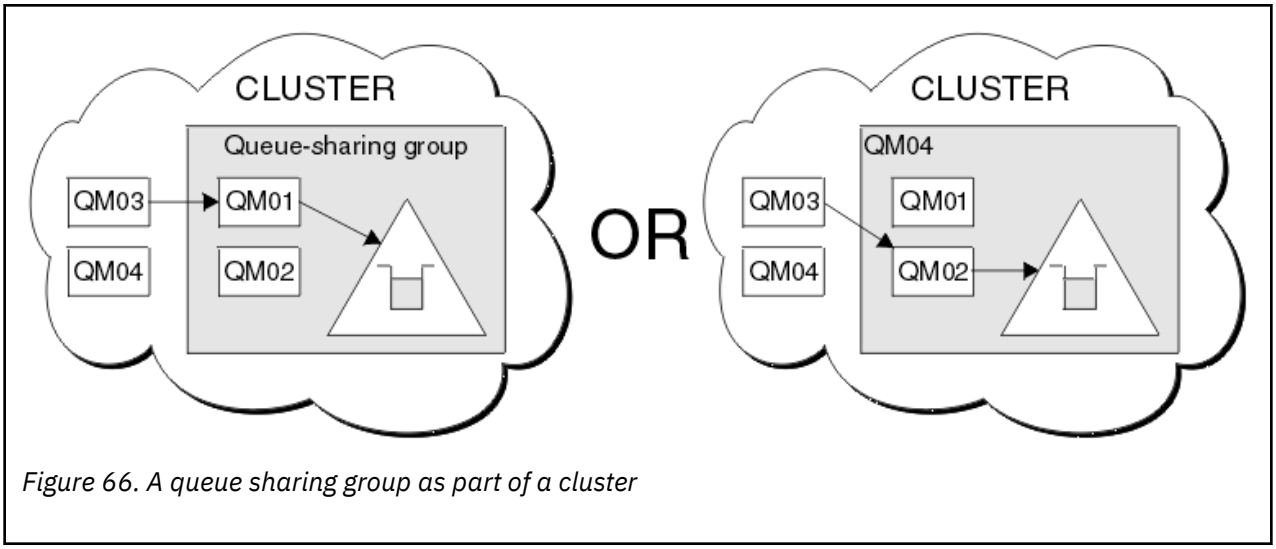


Figure 66. A queue sharing group as part of a cluster

z/OS Influencing workload distribution with shared queues

Use this topic to understand the factors that affect workload distribution with shared queues in a queue sharing group.

IBM MQ does not provide workload balancing for shared queues. However, workload distribution in a queue sharing group (QSG) can be influenced in *a pull based fashion*. The choice of which queue manager services a queue (receives a message written to a shared queue) is affected by the available processing capacity of each queue manager in the queue sharing group and the workload management goals defined across the sysplex.

However, it is important to appreciate, the queue manager that performs the MQPUT of a message can also have a large influence in deciding which queue manager gets the message.

A local queue manager is more likely to perform the MQGET

For an application performing an MQPUT, the local queue manager is said to be the queue manager to which the application is connected.

Exactly which queue manager services an MQPUT of a message by performing an MQGET on behalf of a getting application is influenced by the following considerations.

When a message is put to an empty shared queue, the local queue manager is typically posted before any of the other queue manager in the queue sharing group is notified. If the local queue manager is in a position to process the message, it receives a list transition notification from the coupling facility (CF) before any other queue manager in the QSG. (A list transition notification is a notification that the shared queue has changed state from empty to non-empty.)

The possible scenarios, in this case, are as follows:

1. MQPUT of nonpersistent message out of sync point and *fast put to waiting getter*.

   If there is an application with an *MQGET with wait* on the local queue manager for the queue, then the MQPUT of the message is passed directly to the getting application's buffer and not written to the queue. This is true for shared and non-shared queues. This feature is often called *fast put to a waiting getter* mechanism. In the case of shared queues, no other queue manager in the QSG is notified as there is no transition from empty to non-empty of the queue. This means, for example, that provided this queue manager can service all the puts from this application and assuming that no other applications are putting messages to the queue, then no other queue manager in the queue sharing group assists in draining this queue. If however there is no MQGET with wait on the local queue manager, and a message is put to the shared queue then the CF will notify other queue managers in the queue sharing group according to its rules for notifications of list transitions.
2. MQPUT of a persistent or in-syncpoint message.

In this case, if there is an application with an *MQGET with wait* on the local queue manager, then the message is put to the shared queue and the CF notifies other queue managers in the queue sharing group according to its rules for notifications of list transitions. However, the local queue manager does not wait for a transition notification from the CF but honors any local *MQGET with wait* first and usually performs the get of this message on behalf of the application before any other queue manager in the queue sharing group can respond to a CF notification. This is dependent on how busy the local queue manager is. Otherwise, any queue manager notified by the CF due to the arrival of the message on the empty queue will try to service the get first. The first queue manager to respond processes the new message.

3. Finally, if the queue is not drained of messages, where the CF has sent a notification of a state change from empty to non-empty for the queue, all connected queue managers will have an opportunity to assist in the processing of the queue. In this event, the workload is said to be *pull based*.

This design allows for the improved performance over a purely pull based workload distribution. The aim is to take advantage of the high availability offered by queues held in the CF while allowing the queue manager, where possible, to perform the MQGET without needing to reference the CF and so to process the message workload as efficiently as possible.

Alternative approaches can be adopted where emphasis on balance of the workload is more important than the previously described performance enhancements. For example, ensuring that none of the getting applications are connected to the same queue manager that the putting application is connected to. Using this design all messages are put to the queue and all queue managers in the QSG are notified when the queue moves from empty to non-empty, in accordance with the CF algorithm for handling such transitions. In addition, the *fast put to waiting getter* mechanism is not applicable.

z/OS Where to find more information about shared queues and queue sharing groups

Use the table in this topic to find more information about how IBM MQ for z/OS uses shared queues and queue sharing groups.

Table 19. Where to find more information about shared queues and queue sharing groups

Topic	Where to look
Queue sharing group recovery	“Recovery and restart on z/OS” on page 259
Queue sharing group security	“Security concepts in IBM MQ for z/OS” on page 275
Private and global object definitions Directing commands to different queue managers	Sources from which you can issue commands on z/OS
Planning your coupling facility environment	Defining coupling facility resources
Planning your SMDS environment	Planning your shared message data set (SMDS) environment
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	“Shared queues and queue sharing groups” on page 177
Utility programs Migrating queues	IBM MQ utilities on z/OS reference

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Console messages	Messages for IBM MQ for z/OS
MQSC commands	MQSC commands
IBM MQ clusters	Configuring a queue manager cluster
IBM MQ distributed queuing Channel names	Introduction to distributed queue management
Writing applications	Overview of application design
MQCONNX call	MQCONNX

z/OS Intra-group queuing

This section describes intra-group queuing, an IBM MQ for z/OS function unique to the z/OS platform. This function is only available to queue managers defined to a queue sharing group.

For information about queue sharing groups, see “Shared queues and queue sharing groups” on page 177.

Intra-group queuing concepts

You can perform fast message transfer between queue managers in a queue sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE, which is a shared transmission queue. Each queue manager in the queue sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing (IGQ) is enabled and the target queue manager is within the queue sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

You enable intra-group queuing through a queue manager attribute. Intra-group queuing moves nonpersistent messages outside sync point, and persistent messages within sync point. If it finds a problem delivering messages to the target queue, intra-group queuing tries to put them to the dead-letter queue. If the dead-letter queue is full or undefined, nonpersistent messages are discarded, but persistent messages are backed out and returned to the SYSTEM.QSG.TRANSMIT.QUEUE, and the IGQ agent tries to deliver the messages until it is successful.

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue sharing group can use intra-group queuing to *hop* the message to the correct destination.

There might be times when you want the local queue manager to put a message directly to the target queue if the target queue is a shared queue, rather than the message first being transferred to the target queue manager. You can use the queue manager attribute SQQMNAME to control this. If you set the value of SQQMNAME to USE, the MQOPEN command is performed on the queue manager specified by the **ObjectQMgrName**.

However, if the target queue is a shared queue and you set the value of SQQMNAME to IGNORE, and the **ObjectQMgrName** is that of another queue manager in the queue sharing group, the shared queue is opened on the local queue manager. If the local queue manager cannot open the target queue, or put a

message to the queue, the message is transferred to the specified **ObjectQMgrName** through either IGQ or an IBM MQ channel.

Intra-group queuing can be used to deliver, more efficiently, small messages to queues residing on remote queue managers within a queue sharing group. Intra-group queuing also supports large messages, the largest being 100 MB *minus* the length of the transmission queue header.

Note: If you use this feature, users must have the same access to the queues on each queue manager in the queue sharing group.

The following diagram shows a typical example of intra-group queuing.

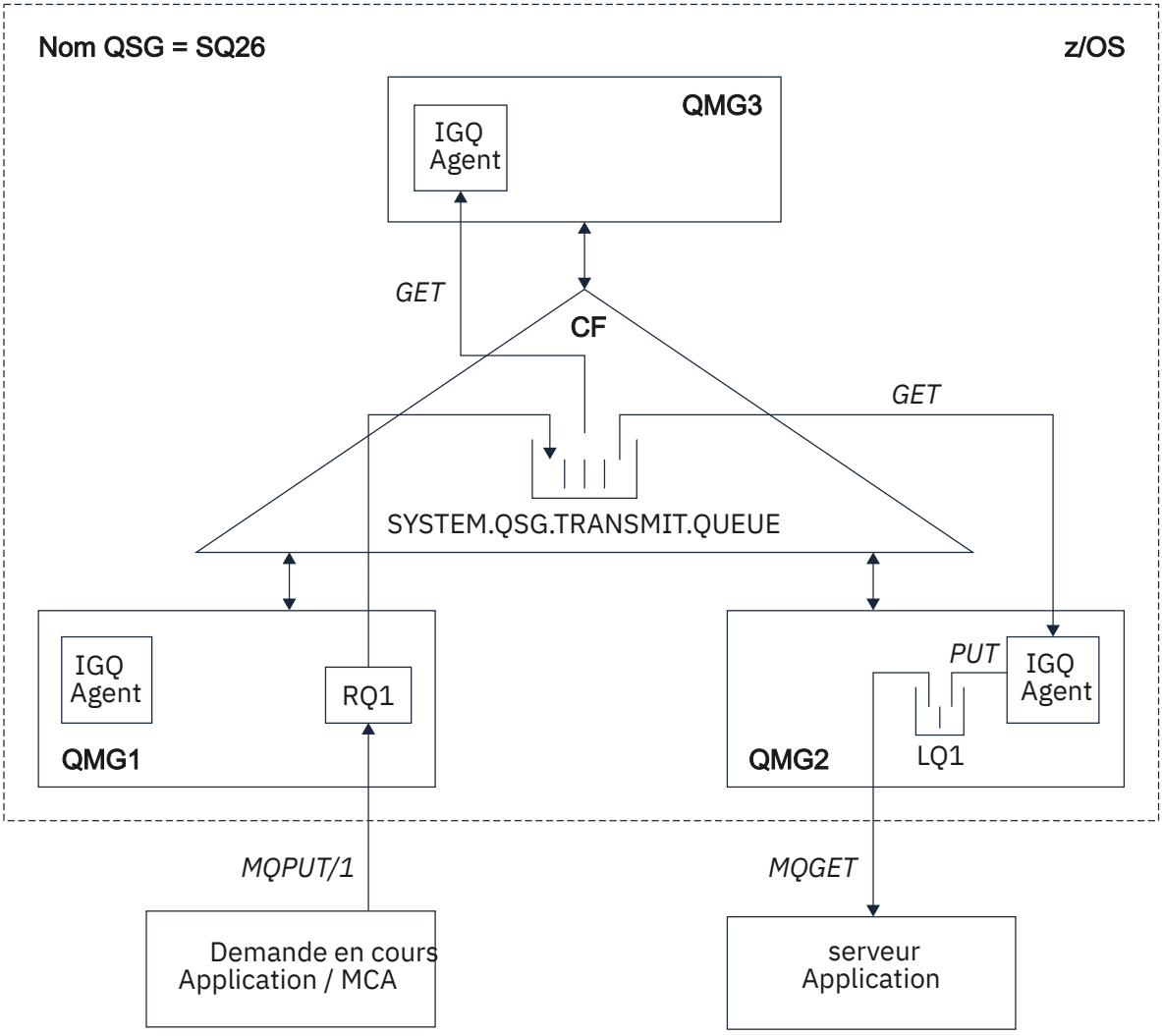


Figure 67. An example of intra-group queuing

The diagram shows:

- IGQ agents running on three queue managers (QMG1, QMG2, and QMG3) that are defined to a queue sharing group called SQ26.
- Shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE that is defined in the coupling facility (CF).
- A remote queue definition that is defined in queue manager QMG1.
- A local queue that is defined in queue manager QMG2.
- A requesting application (this application could be a Message Channel Agent (MCA)) that is connected to queue manager QMG1.

- A server application that is connected to queue manager QMG2.
- A request message being placed on to the SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing and the intra-group queuing agent

An IGQ agent is started during queue manager initialization. When applications open and put messages to remote queues, the local queue manager determines whether intra-group queuing is used for message transfer. If intra-group queuing is to be used, the local queue manager places the message on to the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent on the target remote queue manager retrieves the message and places it on to the destination queue.

Intra-group queuing terminology

Explanations of the terminology: intra-group queuing, shared transmission queue for use by intra-group queuing, and intra-group queuing agent.

Intra-group queuing

Intra-group queuing can effect potentially fast and less expensive message transfer between queue managers in a queue sharing group, without the need to define channels.

Shared transmission queue for use by intra-group queuing

Each queue sharing group has a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE for use by intra-group queuing. If intra-group queuing is enabled, SYSTEM.QSG.TRANSMIT.QUEUE appears in the name resolution path when opening remote queues. When applications (including Message Channel Agents (MCAs)) put messages to a remote queue, the local queue manager determines the eligibility of messages for fast transfer and places them on SYSTEM.QSG.TRANSMIT.QUEUE.

Intra-group queuing agent

The IGQ agent is the task, started at queue manager initialization, that waits for suitable messages to arrive on the SYSTEM.QSG.TRANSMIT.QUEUE. The IGQ agent retrieves suitable messages from this queue and delivers them to the destination queues.

The IGQ agent for each queue manager is always started because intra-group queuing is used by the queue manager itself for its own internal processing.

z/OS Benefits of intra-group queuing

The benefits of intra-group queuing are: reduced system definitions, reduced system administration, improved performance, supports migration, and delivery of messages when multi-hopping between queue managers in a queue sharing group.

The benefits of intra-group queuing are:

Reduced system definitions
Intra-group queuing removes the need to define channels between queue managers in a queue sharing group.

Reduced system administration
Because there are no channels defined between queue managers in a queue sharing group, there is no requirement for channel administration.

Improved performance
Because there is only one IGQ agent needed for the delivery of a message to a target queue (instead of two intermediate sender and receiver agents), the delivery of messages using intra-group queuing can be less expensive than the delivery of messages using channels. In intra-group queuing there is only a receiving component, because the need for the sending component has been removed. This saving is because the message is available to the IGQ agent at the destination queue manager for

delivery to the destination queue once the put operation at the local queue manager has completed and, in the case of messages put in sync point scope, committed.

Supports migration

Applications external to a queue sharing group can deliver messages to a queue residing on any queue manager in the queue sharing group, while being connected only to a particular queue manager in the queue sharing group. This is because messages arriving on a receiver channel, destined for a queue on a remote queue manager, can be transparently sent to the destination queue using intra-group queuing. This facility allows applications to be deployed among the queue sharing group without the need to change any systems that are external to the queue sharing group.

A typical configuration is illustrated by the following diagram, in which:

- A requesting application connected to queue manager QMG1 needs to send a message to a local queue on queue manager QMG3.
- Queue manager QMG1 is connected only to queue manager QMG2.
- Queue managers QMG2 and QMG3, which were previously connected using channels, are now members of queue sharing group SQ26.

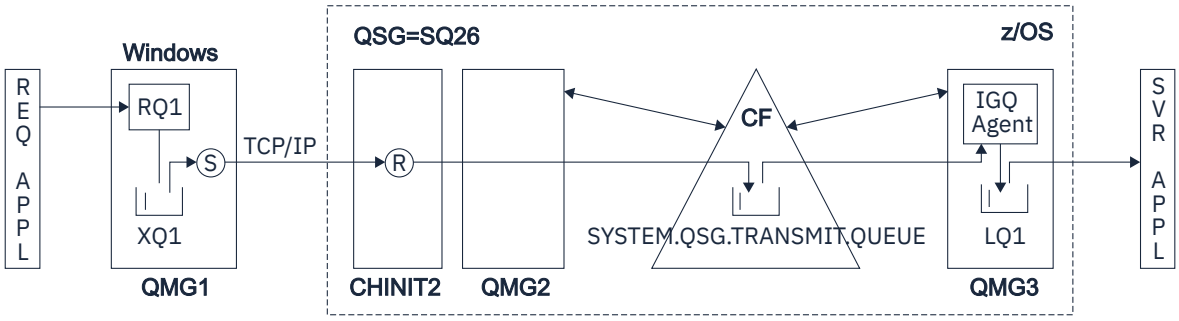


Figure 68. An example of migration support

The flow of operations is as follows:

1. The requesting application puts a message, destined for local queue LQ1 at remote queue manager QMG3, on to remote queue definition RQ1.
2. Queue manager QMG1, running on a Windows NT workstation, places the message on to the transmission queue XQ1.
3. Sender MCA (S) on QM1 transmits the message, using TCP/IP, to the receiver MCA (R) on channel initiator CHINIT2.
4. Receiver MCA (R) on channel initiator CHINIT2 places the message on to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
5. IGQ agent on queue manager QMG3 retrieves the message from the SYSTEM.QSG.TRANSMIT.QUEUE and places it on to the target local queue LQ1.
6. The server application retrieves the message from the target local queue and processes it.

Delivery of messages when multi-hopping between queue managers in a queue sharing group

The previous diagram in Supports migration also illustrates the delivery of messages when multi-hopping between queue managers in a queue sharing group. Messages arriving on a queue manager within the queue sharing group, but destined for a queue on another queue manager in the queue sharing group, can be easily transmitted to the destination queue on the destination queue manager, using intra-group queuing.

z/OS Limitations of intra-group queuing

The limitations of intra-group queuing are: messages eligible for transfer using intra-group queuing, number of intra-group queuing agents per queue manager, and starting and stopping the intra-group queuing agent.

This topic describes the limitations of intra-group queuing.

Messages eligible for transfer using intra-group queuing
Because intra-group queuing uses a shared transmission queue that is defined in the coupling facility (CF), intra-group queuing is limited to delivering messages of the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Number of intra-group queuing agents per queue manager
Only one IGQ agent is started per queue manager in a queue sharing group.

Starting and stopping the intra-group queuing agent
The IGQ agent is started during queue manager initialization and terminated during queue manager shut-down. It is designed to be a long running, self recovering (in the event of abnormal termination), task. If there is an error with the definition of the SYSTEM.QSG.TRANSMIT.QUEUE (for example, if this queue is Get inhibited) the IGQ agent keeps retrying. If the IGQ agent encounters an error that results in normal termination of the agent while the queue manager is still active, it can be restarted by issuing an ALTER QMGR IGQ(ENABLED) command. This command avoids the need to recycle the queue manager.

Setting the queue manager attribute IGQ to ENABLED or DISABLED
If the queue manager attribute IGQ is set to ENABLED or DISABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See Getting started with intra-group queueing for more information.

z/OS Getting started with intra-group queuing

You can enable, disable, and use intra-group queuing as described in this topic.

Enabling intra-group queuing
To enable intra-group queuing on your queue managers, you need to do the following:

- Define a shared transmission queue called SYSTEM.QSG.TRANSMIT.QUEUE. The definition of this queue can be found in thlqual.SCSQPROC(CSQ4INSS), the CSQINP2 sample for SYSTEM objects for queue sharing groups. This queue must be defined with the correct attributes, as stated in thlqual.SCSQPROC(CSQ4INSS), for intra-group queuing to work properly.
- Because the IGQ agent is always started at queue manager initialization, intra-group queuing is always available for inbound message processing. The IGQ agent processes any messages that are placed on the SYSTEM.QSG.TRANSMIT.QUEUE. However, to enable intra-group queuing for outbound processing, the queue manager attribute IGQ must be set to ENABLED.

  Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 233 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 225 for further information.

Disabling intra-group queuing

To disable intra-group queuing for outbound message transfer, set the queue manager attribute IGQ to DISABLED. If intra-group queuing is disabled for a particular queue manager, the IGQ agent on that queue manager can still process inbound messages that have been placed on the SYSTEM.QSG.TRANSMIT.QUEUE by a queue manager that does have intra-group queuing enabled for outbound transfer.

Important: If the queue manager attribute IGQ is set to ENABLED, existing object handles may be invalidated with reason code MQRC_OBJECT_CHANGED. See “Specific properties of intra-group queuing” on page 233 for more information. As described in the 'Programmer response' section for this reason code, applications need to be coded to handle this situation (see 2041 (07F9) (RC2041): MQRC_OBJECT_CHANGED for more details).

Additionally, as IGQ is designed as a long running and self-recovering task, which starts during initialization and terminates with shutdown, see “Limitations of intra-group queuing” on page 225 for further information.

Using intra-group queuing

Once intra-group queuing is enabled, it is available for use and a queue manager uses it whenever possible. That is, when an application puts a message to a remote queue definition, to a fully qualified remote queue, or to a cluster queue, the queue manager determines if the message is eligible to be delivered using intra-group queuing and if it is, places the message on to SYSTEM.QSG.TRANSMIT.QUEUE. There is no need to change user applications, or to application queues, because for eligible messages the queue manager uses the SYSTEM.QSG.TRANSMIT.QUEUE, in preference to any other transmission queue.

z/OS Intra-group queuing configurations

In addition to the typical intra-group queuing configuration, other configurations are possible.

“Intra-group queuing concepts” on page 221 describes the typical configuration.

Related concepts

“Distributed queuing with intra-group queuing (multiple delivery paths)” on page 226
For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

“Clustering with intra-group queuing (multiple delivery paths)” on page 228
It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

“Clustering, intra-group queuing and distributed queuing” on page 230
It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

z/OS *Distributed queuing with intra-group queuing (multiple delivery paths)*

For applications that process short messages it might be feasible to configure intra-group queuing only for delivering messages between queue managers in a queue sharing group.

The choice of intra-group queuing over channel communications can be controlled by the CFSTRUCT type level. (3 instead of 4 or 5). The maximum message length as set on the SYSTEM.QSQ.TRANSMIT.QUEUE.

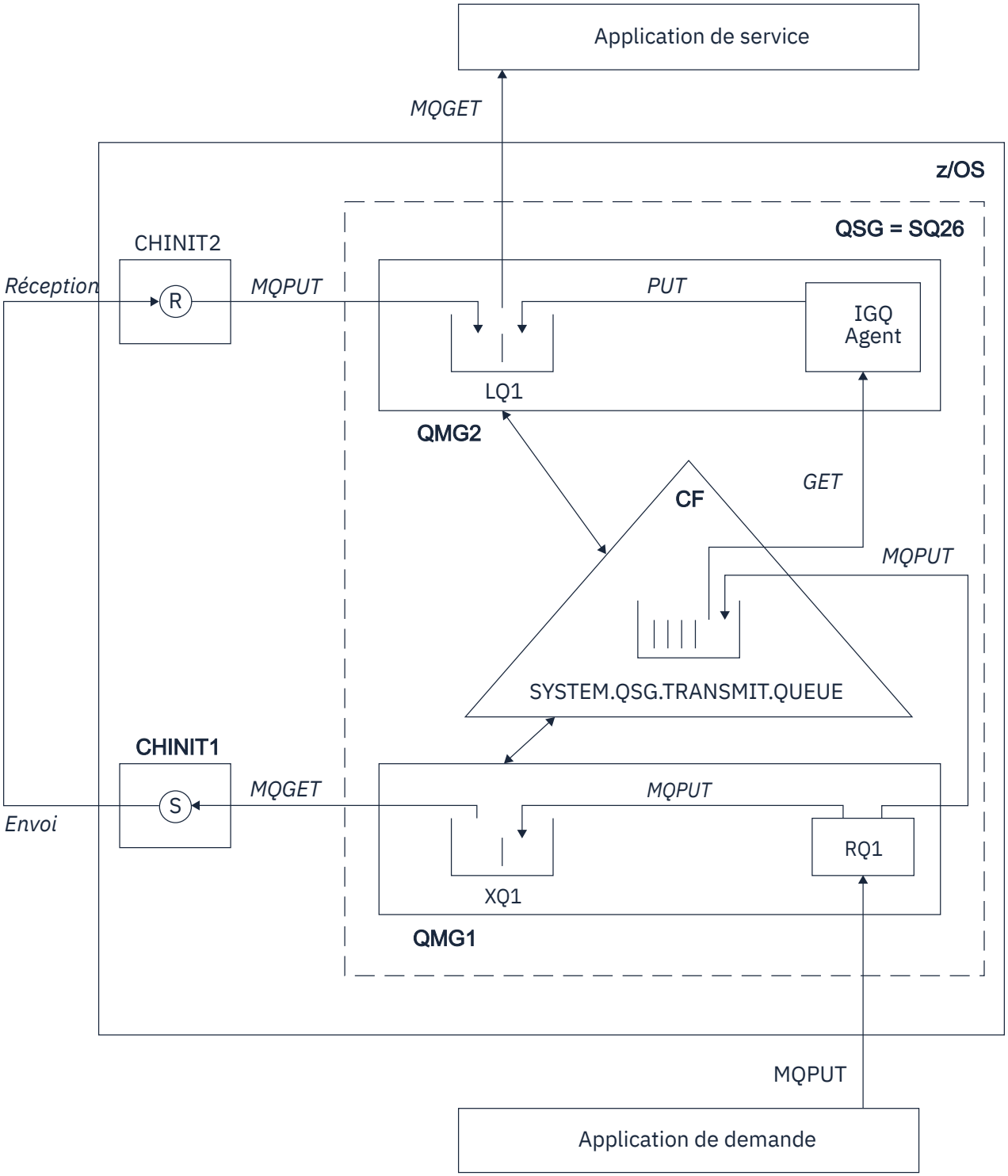


Figure 69. An example configuration

Open/Put processing

1. It is important to note that when the requesting application opens remote queue RQ1, name resolution occurs for both the non-shared transmission queue XQ1 and the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
2. When the requesting application puts a message on to the remote queue, based on whether intra-group queuing is enabled for outbound transfer on the queue manager and on the message characteristics, the message is put to transmission queue XQ1, or to transmission queue

SYSTEM.QSG.TRANSMIT.QUEUE. The queue manager places all large messages on to transmission queue XQ1, and all small messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.

3. If transmission queue XQ1 is full, or is not available, put requests for large messages fail synchronously with a suitable return and reason code. However, put requests for small messages continue to succeed and are placed on transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
4. If transmission queue SYSTEM.QSG.TRANSMIT.QUEUE is full, or cannot be put to, put requests for small messages fail synchronously with a suitable return and reason code. However, put requests for large messages continue to succeed and are placed on transmission queue XQ1. In this case, no attempt is made to put the small messages on to a transmission queue.

Flow for large messages

1. The requesting application puts large messages to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue XQ1.
3. Sender MCA (S) on queue manager QMG1 retrieves the messages from transmission queue XQ1 and sends them to queue manager QMG2.
4. Receiver MCA (R) on queue manager QMG2 receives the messages and places them on to destination queue LQ1.
5. The serving application retrieves and then processes the messages from queue LQ1.

Flow for small messages

1. The requesting application puts small messages on to remote queue RQ1.
2. Queue manager QMG1 puts the messages on to transmission queue SYSTEM.QSG.TRANSMIT.QUEUE.
3. IGQ on queue manager QMG2 retrieves the messages and places them on to the destination queue LQ1.
4. The serving application retrieves the messages from queue LQ1.

Points to note

1. The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
2. A potentially faster message delivery mechanism can be achieved for small messages.
3. Multiple paths are available for message delivery (that is, the normal channel route and the intra-group queuing route).
4. The intra-group queuing route, being potentially faster, is selected in preference to the normal channel route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence (though this delivery is also possible if messages are delivered using only the normal channel route).
5. When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to transmission queue XQ1.

z/OS *Clustering with intra-group queuing (multiple delivery paths)*

It is possible to configure queue managers so that they are in a cluster as well as in a queue sharing group.

When messages are sent to a cluster queue and the local and remote destination queue managers are in the same queue sharing group, intra-group queuing is used for the delivery of small messages (using the SYSTEM.QSG.TRANSMIT.QUEUE), and the delivery of large messages if intra-group queuing supports the size of the message. Also, the SYSTEM.CLUSTER.TRANSMIT.QUEUE is used for the delivery of messages to any queue manager that is in the cluster, but outside the queue sharing group. The following diagram illustrates this configuration (the channel initiators are not shown).

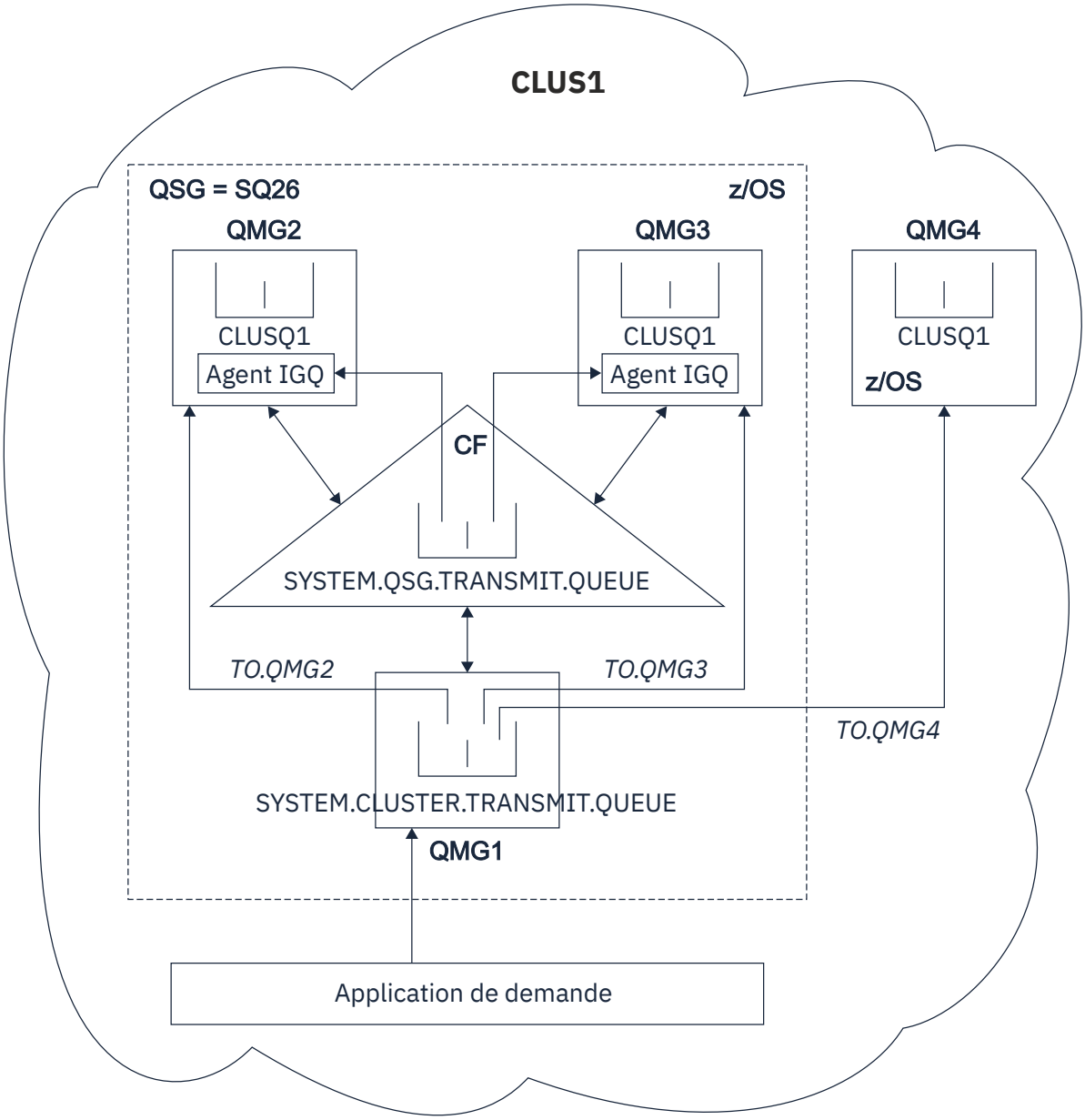


Figure 70. An example of clustering with intra-group queuing

The diagram shows:

- Four z/OS queue managers QMG1, QMG2, QMG3, and QMG4 configured in a cluster CLUS1.
- Queue managers QMG1, QMG2, and QMG3 configured in a queue sharing group SQ26.
- IGQ agents running on queue managers QMG2 and QMG3.
- The local SYSTEM.CLUSTER.TRANSMIT.QUEUE defined in QMG1.

  Note: For clarity, the SYSTEM.CLUSTER.TRANSMIT.QUEUE on the other queue managers not shown.
- The shared SYSTEM.QSG.TRANSMIT.QUEUE defined in the CF, which is in an IBM MQ structure configured with the CFLEVEL(3) RECOVER(YES) attribute.
- Cluster channels TO.QMG2 (connecting QMG1 to QMG2), TO.QMG3 (connecting QMG1 to QMG3), and TO.QMG4 (connecting QMG1 to QMG4).
- Cluster queue CLUSQ1 being hosted on queue managers QMG2, QMG3, and QMG4.

Assume that the requesting application opens the cluster queue with the MQOO_BIND_NOT_FIXED option, so that the target queue manager for the cluster queue is selected at put time.

If the selected target queue manager is QMG2:

- All large messages put by the requesting application are:
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1, because SYSTEM.QSG.TRANSMIT.QUEUE is in a CFLEVEL(3) structure; therefore supports messages only up to 63 KB in size.
  - Transferred to cluster queue CLUSQ1 on QMG2 using cluster channel TO.QMG2
- All small messages put by the requesting application are
  - Put to the shared transmission queue SYSTEM.QSG.TRANSMIT.QUEUE. This queue is in a structure configured with the RECOVER(YES) attribute, so is used for both persistent, and non-persistent, small messages.
  - Retrieved by the IGQ agent on QMG2
  - Put to the cluster queue CLUSQ1 on QMG2

If the selected target queue manager is QMG4:

- Because QMG4 is not a member of queue sharing group SQ26, all messages put by the requesting application are
  - Put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE on QMG1
  - Transferred to cluster queue CLUSQ1 on QMG4 using cluster channel TO.QMG4

Points to note

- The requesting application does not need to be aware of the underlying mechanism used for the delivery of messages.
- A potentially faster delivery mechanism is achieved for the transfer of small non-persistent messages between queue managers in a queue sharing group (even if the same queue managers are in a cluster).
- Multiple paths are available for message delivery (that is, both the cluster route and the intra-group queuing route).
- The intra-group queuing route, being potentially faster, is selected in preference to the cluster route. Depending on the message characteristics, message delivery might be divided across the two paths. Hence, messages might be delivered out of sequence. It is important to note that this delivery is possible without regard to the MQOO_BIND_* option specified by the application. Intra-group queuing distributes messages in the same way as clustering does, depending on whether the MQOO_BIND_NOT_FIXED, MQOO_BIND_ON_OPEN, MQOO_BIND_ON_GROUP, or MQOO_BIND_AS_Q_DEF is specified on open.
- When a route has been selected, and messages have been placed on to the transmission queues, only the selected route is used for message delivery. Any unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE are not diverted to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

z/OS *Clustering, intra-group queuing and distributed queuing*

It is possible to configure a queue manager that is a member of a cluster as well as a queue sharing group and is connected to a distributed queue manager using a sender/receiver channel pair.

This configuration is a combination of distributed queuing with intra-group queuing and clustering with intra-group queuing.

Intra-group queuing is described in “Distributed queuing with intra-group queuing (multiple delivery paths)” on page 226.

Clustering with intra-group queuing is described in “Clustering with intra-group queuing (multiple delivery paths)” on page 228.

z/OS Intra-group queuing messages

This section describes the messages put to the SYSTEM.QSG.TRANSMIT.QUEUE.

Message structure

Like all other messages that are put to transmission queues, messages that are put to the SYSTEM.QSG.TRANSMIT.QUEUE are prefixed with the transmission queue header (MQXQH).

Message persistence

In IBM WebSphere MQ 5.3 and above, shared queues support both persistent and non-persistent messages.

If the queue manager terminates while the IGQ agent is processing non-persistent messages, or if the IGQ agent terminates abnormally while in the middle of processing messages, non-persistent messages being processed might be lost. Applications must make arrangements for the recovery of non-persistent messages if their recovery is required.

If a put request for a non-persistent message, issued by the IGQ agent, fails unexpectedly, the message being processed is lost.

Delivery of messages

The IGQ agent retrieves and delivers all nonpersistent messages outside of sync point scope, and all persistent messages within sync point scope. In this case, the IGQ agent acts as the sync point coordinator. The IGQ agent therefore processes nonpersistent messages like the way fast, nonpersistent messages are processed on a message channel. See Fast, nonpersistent messages.

Batching of messages

The IGQ agent uses a fixed batch size of 50 messages. Any persistent messages retrieved within a batch are committed at intervals of 50 messages. The agent commits a batch consisting of persistent messages when there are no more messages available for retrieval on the SYSTEM.QSG.TRANSMIT.QUEUE.

Message size

The maximum size of message that can be put to the SYSTEM.QSG.TRANSMIT.QUEUE is the maximum supported message length for shared queues minus the length of a transmission queue header (MQXQH).

Default message persistence and default message priority

If the SYSTEM.QSG.TRANSMIT.QUEUE is in the queue name resolution path established at open time, then for messages that are put with default persistence and default priority (or with default persistence or default priority), the normal rules are applied in the selection of the queue that has default priority and persistence values that are used. (See the IBM MQ messages section for more information about the rules of queue selection).

Related concepts

“Undelivered/unprocessed messages” on page 231
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 232
This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

z/OS *Undelivered/unprocessed messages*

This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

If an IGQ agent cannot deliver a message to the destination queue, the IGQ agent:

- Honors the MQRO_DISCARD_MSG report option (if the Report options field of the MQMD for the undelivered message indicates that it must) and discards the undelivered message.

- Attempts to place the undelivered message on to the dead letter queue for the destination queue manager, if the message has not already been discarded. The IGQ agent prefixes the message with a dead letter queue header (MQDLH).

If a dead letter queue is not defined, or if an undelivered message cannot be put to the dead letter queue, and if the undelivered message is:

- persistent, the IGQ agent backs out the current batch of persistent messages that it is processing, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 233.
- non-persistent, the IGQ agent discards the message and continues to process the next message.

If a queue manager in a queue sharing group is terminated before its associated IGQ agent has had time to process all its messages, the unprocessed messages remain on the SYSTEM.QSG.TRANSMIT.QUEUE until the queue manager is next started. The IGQ agent then retrieves and delivers the messages to the destination queues.

If the coupling facility fails before all the messages on the SYSTEM.QSG.TRANSMIT.QUEUE have been processed, any unprocessed non-persistent messages are lost.

IBM recommends that applications do not put messages directly to transmission queues. If an application does put messages directly to the SYSTEM.QSG.TRANSMIT.QUEUE, the IGQ agent might not be able to process these messages and they remain on the SYSTEM.QSG.TRANSMIT.QUEUE. Users then have to use their own methods to deal with these unprocessed messages.

z/OS *Report messages - Intra Group Queuing*

This topic describes the report messages: Confirmation of arrival, confirmation of delivery, expiry report, and exception report.

Confirmation of arrival (COA)/confirmation of delivery (COD) report messages
COA and COD messages are generated by the queue manager, when intra-group queuing is used.

Expiry report messages
Expiry report messages are generated by the queue manager.

Exception report messages

Depending on the MQRO_EXCEPTION_* report option specified in the *Report options* field of the message descriptor for the undelivered message, the IGQ agent generates the required exception report and places it on the specified reply-to queue. Intra-group queuing can be used to deliver the exception report to the destination reply-to queue.

The persistence of the report message is the same as the persistence of the undelivered message. If the IGQ agent fails to resolve the name of the destination reply-to queue, or if it fails to put the reply message to a transmission queue (for subsequent transfer to the destination reply-to queue) it attempts to put the exception report to the dead letter queue of the queue manager on which the report message is generated. If it is not possible, then if the undelivered message is:

- persistent, the IGQ agent discards the exception report, backs out the current batch of messages, and enters a state of retry. For more information, see "Specific properties of intra-group queuing" on page 233.
- non-persistent, the IGQ agent discards the exception report and continues processing the next message on the SYSTEM.QSG.TRANSMIT.QUEUE.

z/OS Security for intra-group queuing

This topic describes the security arrangements for intra-group queuing.

Queue manager attributes IGQAUT (IGQ authority) and IGQUSER (IGQ agent user ID) can be set to control the level of security checking that is performed when the IGQ agent opens destination queues.

Intra-group queuing authority (IGQAUT)

The IGQAUT attribute can be set to indicate the type of security checks to be performed, and hence to determine the userids to be used by the IGQ agent when it establishes the authority to put messages on to the destination queue.

The IGQAUT attribute is analogous to the PUTAUT attribute that is available on channel definitions.

Intra-group queuing user identifier (IGQUSER)

The IGQUSER attribute can be used to nominate a user ID to be used by the IGQ agent when it establishes the authority to put messages on to a destination queue.

The IGQUSER attribute is analogous to the MCAUSER attribute that is available on channel definitions.

z/OS Specific properties of intra-group queuing

This section describes the specific properties of intra-group queuing including Invalidation of object handles, self recovery and retry capability of the intra-group queuing agent, and the intra-group queuing agent and serialization.

Invalidation of object handles (MQRC_OBJECT_CHANGED)

If the attributes of an object are found to have changed after the object is opened, the queue manager invalidates the object handle with MQRC_OBJECT_CHANGED on its next use.

Intra-group queuing introduces the following rules for object handle invalidation:

- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was ENABLED at open time, but intra-group queuing is found to be DISABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was not included in the name resolution path during open processing because intra-group queuing was DISABLED at open time, but intra-group queuing is found to be ENABLED at put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.
- If the SYSTEM.QSG.TRANSMIT.QUEUE was included in the name resolution path during open processing because intra-group queuing was enabled at open time, but the SYSTEM.QSG.TRANSMIT.QUEUE definition is found to have changed by put time, then the queue manager invalidates the object handle and fails the put request with MQRC_OBJECT_CHANGED.

Self recovery of the intra-group queuing agent

If the IGQ agent terminates abnormally, message CSQM067E is issued and the IGQ agent starts again.

Retry capability of the intra-group queuing agent

If the IGQ agent encounters a problem accessing the SYSTEM.QSG.TRANSMIT.QUEUE (because it is not defined, for example, or is defined with incorrect attributes, or is inhibited for Gets, or for some other reason), the IGQ agent goes into the state of retry.

The IGQ agent observes short and long retry counts and intervals. The values for these counts and intervals, which cannot be changed, are as follows:

Table 20. Short and long retry counts and intervals values

Constant	**Value**
Short retry count	10
Short retry interval	60 seconds = 1 min
Long retry count	999,999,999

Table 20. Short and long retry counts and intervals values (continued)	
Constant	**Value**
Long retry interval	1200 seconds = 20 min

The intra-group queuing agent and serialization

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail.

If there is a failure of a queue manager in a queue sharing group while the IGQ agent is dealing with uncommitted messages on a shared queue or queues, the IGQ agent ends, and shared queue peer recovery takes place for the failing queue manager. Because shared queue peer recovery is an asynchronous activity, it leaves the possibility for the failing queue manager, and also the IGQ agent for that queue manager, to restart before shared queue peer recovery is complete. Which in turn leaves the possibility for any committed messages to be processed ahead of and out of sequence with the messages still being recovered. To ensure that messages are not processed out of sequence, the IGQ agent serializes access to shared queues by issuing the MQCONNX API call.

An attempt, by the IGQ agent to serialize access to shared queues while peer recovery is still in progress might fail. An error message is issued and the IGQ agent is put into retry state. When queue manager peer recovery is complete, for example at the time of the next retry, the IGQ agent can start.

z/OS Storage management on z/OS

IBM MQ for z/OS requires permanent and temporary data structures and uses page sets and memory buffers to store this data. These topics give more details on how IBM MQ uses these page sets and buffers.

Related concepts

"Page sets for IBM MQ for z/OS" on page 234
Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

"Storage classes for IBM MQ for z/OS" on page 235
A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

"Buffers and buffer pools for IBM MQ for z/OS" on page 237
IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

Related reference

"Where to find more information about storage management for IBM MQ for z/OS" on page 238
Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

z/OS Page sets for IBM MQ for z/OS

Use this topic to understand how IBM MQ for z/OS uses pages sets to store messages.

A *page set* is a VSAM linear data set that has been specially formatted to be used by IBM MQ. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on Db2, and the messages on shared queues. These are not stored on the queue manager page sets. For more information about shared queues, see "Shared queues and queue sharing groups" on page 177, and for more information about global definitions, see Private and global definitions.

IBM MQ page sets can be up to 64 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

IBM MQ uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager. For normal operation of IBM MQ it is essential that page set zero does not become full, so do not use it to store messages.

To improve the performance of your system, you should also separate short-lived messages from long-lived messages by placing them on different page sets.

You must format page sets, and IBM MQ provides a FORMAT utility for this; see Formatting page sets (FORMAT). Page sets must also be defined to the IBM MQ subsystem.

IBM MQ for z/OS can be configured to expand a page set dynamically if it becomes full. IBM MQ continues to expand the page set if required until 123 logical extents exist, if there is sufficient disk storage space available. The extents can span volumes if the linear data set is defined in this way, however, IBM MQ cannot expand the page sets beyond 64 GB.

You cannot use page sets from one IBM MQ queue manager on a different IBM MQ queue manager, or change the queue manager name. If you want to transfer the data from one queue manager to another, you must unload all the objects and messages from the first queue manager and reload them onto another.

It is not possible to use page sets greater than 4 GB in a queue manager running a release earlier than V6. During the migration period, when it is likely that you might need to fall back to a previous release of code:

- Do not change page set 0 to be greater than 4 GB.
- Other page sets greater than 4 GB will be left offline when restarting a queue manager with a previous release.

For further information about migrating existing page sets capable of expanding beyond 4 GB, see Defining a page set to be larger than 4 GB.

It is possible for an administrator to dynamically add page sets to a running queue manager, or remove page sets from a running queue manager (except for page set zero). The DEFINE PSID command can run after the queue manager restart has completed, only if the command contains the DSN keyword.

z/OS Storage classes for IBM MQ for z/OS

A storage class is an IBM MQ for z/OS concept that allows the queue manager to map queues to page sets. You can use storage classes to control which data sets are used by which queues.

Introducing storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. You can also use storage classes to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in “IBM MQ and IMS” on page 290).

Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you specify the storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 71 on page 236, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

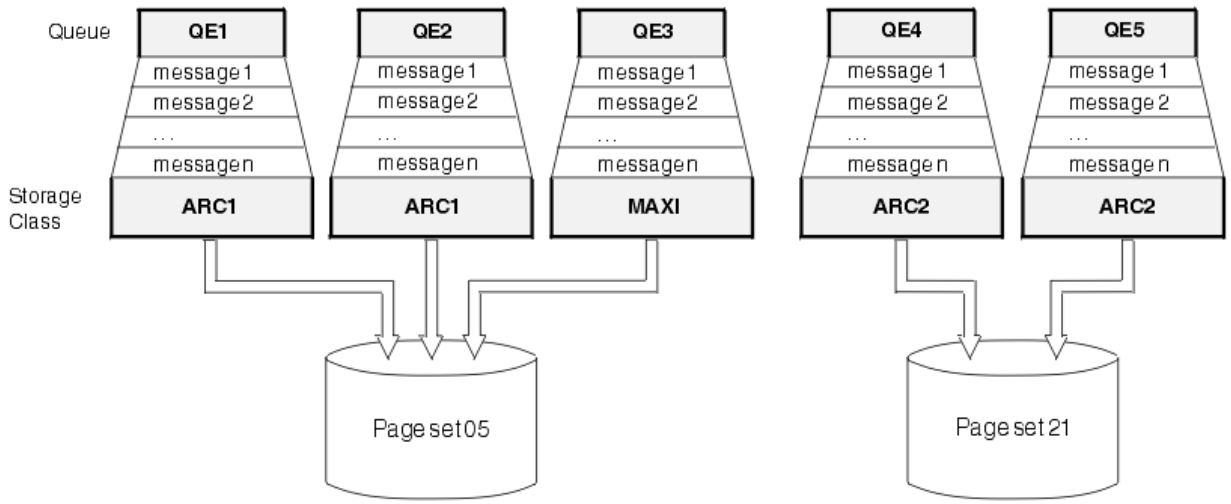


Figure 71. Mapping queues to page sets through storage classes

If you define a queue without specifying a storage class, IBM MQ uses a default storage class.

If a message is put on a queue that names a nonexistent storage class, the application receives an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

You can change a storage class only when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

z/OS Buffers and buffer pools for IBM MQ for z/OS

IBM MQ for z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type separation outlined in Figure 72 on page 237, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the **DEFINE BUFFPOOL** command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Figure 72 on page 237 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

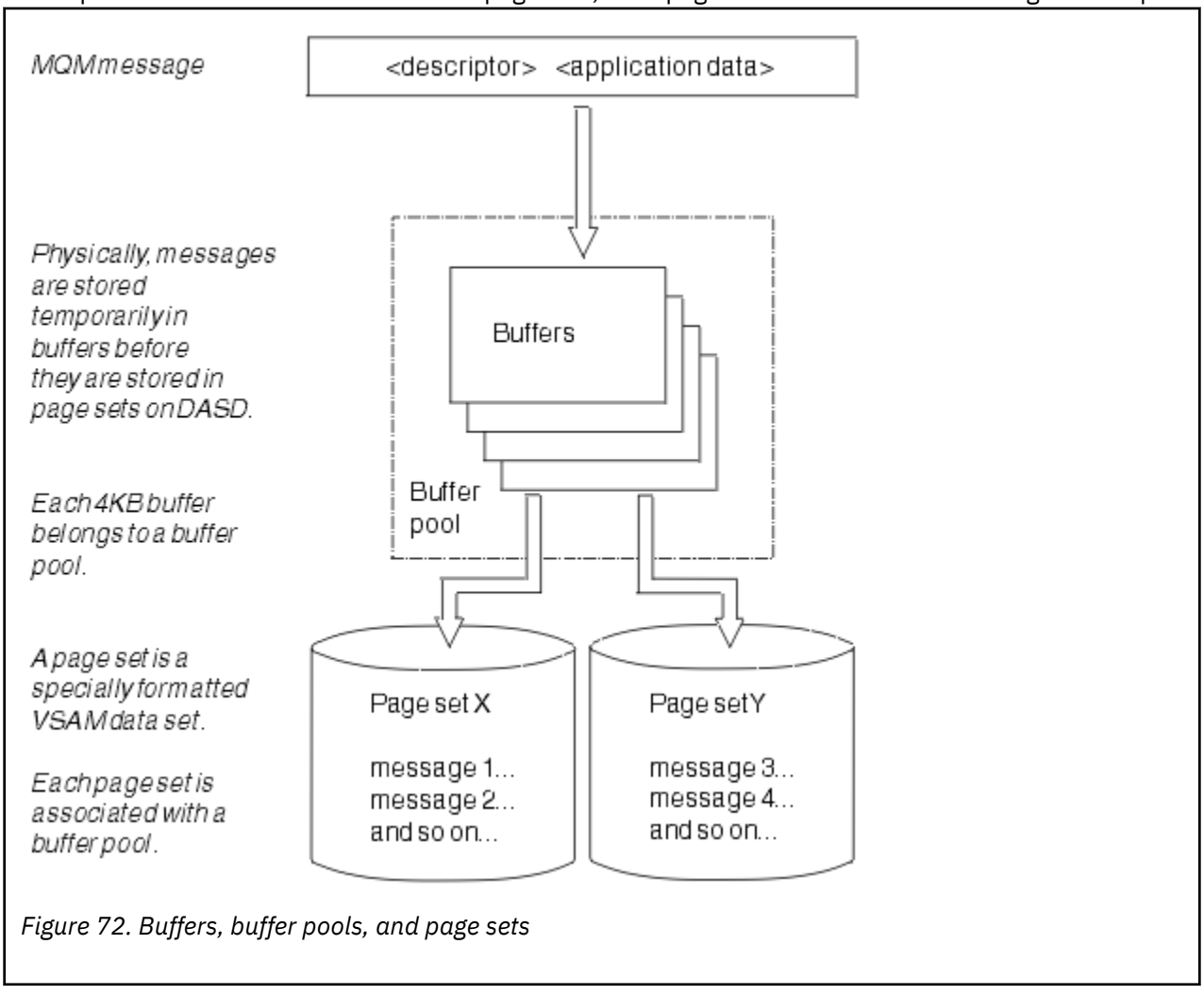


Figure 72. Buffers, buffer pools, and page sets

You can dynamically issue commands to modify buffer pool size, and location, using the **ALTER BUFFPOOL** command. Page sets can be dynamically added by using the **DEFINE PSID** command, or deleted by using the **DELETE PSID** command.

If a buffer pool is too small, IBM MQ issues message CSQP020E. You can then dynamically add more buffers to the affected buffer pool (note that you may have to remove buffers from other buffer pools to do this).

You specify the number of buffers in a pool with the **DEFINE BUFFPOOL** command, and you can dynamically resize buffer pools with the **ALTER BUFFPOOL** command. You determine the current number of buffers in a pool dynamically by displaying a page set that uses the buffer pool, using the **DISPLAY USAGE** command.

For performance reasons, do not put messages and object definitions in the same buffer pool. Use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, keep short-lived messages and long-lived messages in different buffer pools and therefore on different page sets, and in different queues.

The **DEFINE BUFFPOOL** command cannot be used after restart to create a new buffer pool. Instead, if a **DEFINE PSID** command uses the DSN keyword, it can explicitly identify a buffer pool that is not currently defined. That new buffer pool will then be created.

z/OS Where to find more information about storage management for IBM MQ for z/OS

Use this topic as a reference to find further information about storage management for IBM MQ for z/OS.

You can find more information about the topics in this section from the following sources:

Table 21. Where to find more information about storage management

Topic	Where to look
How much storage you need	Planning your storage and performance requirements on z/OS
How large to make your page sets and buffer pools	Plan your page sets and buffer pools
Managing page sets	Managing page sets
MQSC commands	The MQSC commands

z/OS Logging in IBM MQ for z/OS

IBM MQ maintains *logs* of data changes and significant events as they occur. These logs can be used to recover data to a previous state if required.

The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

The log does not contain information for statistics, traces, or performance evaluation. For further details about the statistical and monitoring information that IBM MQ collects, see Monitoring and statistics.

For more information about logging, see the following topics:

- “Log files in IBM MQ for z/OS” on page 239
- “How the log is structured” on page 242
- “How the IBM MQ for z/OS logs are written” on page 243
- “Larger log Relative Byte Address” on page 246
- “The bootstrap data set” on page 247

Related tasks

Planning your logging environment

Setting logs using the system parameter module

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS Log files in IBM MQ for z/OS

Log files contain information needed for transaction recovery. Active log files can be archived so that you can keep log data for a long period.

What is a log file

IBM MQ records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- IBM MQ objects, such as queues
- The IBM MQ queue manager

The active log comprises a collection of data sets (up to 310) which are used cyclically.

You can enable log archiving so that when an active log fills a copy is made in an archive data set. Using archiving allows you to keep log data for an extended period. If you do not use archiving, the logs wrap and earlier data is overwritten. To recover a page set, or recover data in a CF structure, you need log data from when the backup of the page set or structure was taken. An archive log can be created on disk or on tape.

Archiving

Because the active log has a fixed size, IBM MQ copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct-access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, IBM MQ uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. You can catalog each data set using the z/OS integrated catalog facility (ICF).

Archiving is an essential component of IBM MQ recovery. If a unit of recovery is a long-running one, log records within that unit of recovery might be found in the archive log. In this case, recovery requires data from the archive log. However, if archiving is disabled, the active log with new log records wraps, overwriting earlier log records. This means that IBM MQ might not be able to back out the unit of recovery and messages might be lost. The queue manager then terminates abnormally.

Therefore, in a production environment, **never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in Using CSQ6LOGP.

To help prevent problems with unplanned long-running units of work, IBM MQ issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload processing.

Dual logging

In dual logging, each log record is written to two different active log data sets to minimize the likelihood of data loss problems during restart.

You can configure IBM MQ to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. Each active log data set is a single-extent VSAM linear data set (LDS). With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart.

Log shunting

Log shunting causes the log records for some units of work to be written further down the log. This reduces the amount of log data that must be read at queue manager restart, or backout, for long running or long term in-doubt units of work.

When a unit of work is considered to be long, a representation of each log record is written further down the log. This technique is known as *shunting*. When the whole of the unit of work has been processed, the unit of work is in a *shunted* state. Any backout or restart activity relating to the shunted unit of work can use the shunted log records instead of using the original unit of work log records.

Detecting a long-running unit of work is a function of the checkpoint process. At checkpoint time, each active unit of work is checked to establish whether it needs to be shunted. If the unit of work has been through two prior checkpoints since it was created, or since it was last shunted, the unit of work is suitable to be shunted. This means that a single unit of work might be shunted more than once. This is known as a *multi-shunted* unit of work.

A unit of work is shunted every three checkpoints. However the checkpoint is performed asynchronously to the log-switch (or the writing of the log record which caused LOGLOAD to be exceeded).

There is only a single checkpoint taking place at a time, so there might be multiple log-switches before a checkpoint completes.

This means that if there are not enough active logs, or if they are too small, then shunting of a large unit of work might not complete before all the logs are filled.

Message CSQR027I results if shunting is unable to complete.

If log archiving is turned off, ABEND 5C6 with reason 00D1032A occurs if there is an attempt to back out the unit of work for which shunting failed. To avoid this problem you should use OFFLOAD=YES.

Log shunting is always active, and runs whether log archiving is enabled or not.

Note: Although all log records for a unit of work are shunted, the entire content of each record is not shunted, only the part that is necessary for backout. This means that the amount of log data written is kept to a minimum, and that shunted records cannot be used if a page set failure occurs. A long running unit of work is one that has been running for more than three queue manager checkpoints.

For more information about log shunting, see Managing the logs.

Log compression

You can configure IBM MQ for z/OS to compress and decompress log records as they are written and read from the log data set.

Log compression can be used to reduce the amount of data written to the log for persistent messages on private queues. The amount of compression that is achieved depends on the type of data contained within messages. For example, Run Length Encoding (RLE) works by compacting repeated instances of bytes which can give good results efficiently for structured or record oriented data.

Attention: Persistent messages that are being put to a shared queue are not subject to log compression.

You can use fields within the Log manager section of the System Management Facility 115 (SMF) records to monitor how much data compression is achieved. For more information about SMF, see Using the System Management Facility and Accounting and statistics messages.

Log compression increases the processor utilization of the system. You should only consider using compression if throughput of your queue manager is constrained by the IO bandwidth writing to the

log data sets or you are constrained by the disk storage needed to hold log data sets. If you are using shared queues then IO bandwidth constraints can be relieved by adding additional queue managers to the queue sharing group and distributing the workload across more queue managers.

The log compression option can be enabled and disabled as required without the need to stop and restart the queue manager. The queue manager can read any compressed log records regardless of the current log compression setting.

The queue manager supports 3 settings for log compression.

NONE
No log data compression is used. This is the default value.

RLE
Log data compression is performed using run-length encoding (RLE).

ANY
Enable the queue manager to select the compression algorithm that gives the greatest degree of log record compression. This option results in RLE compression.

You can control the compression of log records using one of the following:

- The SET and DISPLAY LOG commands in MQSC; see SET LOG and DISPLAY LOG
- The Set Log and Inquire Log functions in the PCF interface; see Set log and Inquire log
- The CSQ6LOGP macro in the system parameter module; see Using CSQ6LOGP

In addition the Log Print utility CSQ1LOGP has support for expanding any compressed log records.

Log data

The log can contain up to 18 million million million ($1.8*10^{19}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The RBA is referenced by a 6-byte or 8-byte field giving a total addressable range of 2^{48} bytes, or 2^{64} bytes, depending on whether 6-byte or 8-byte log RBAs are in use.

However, when IBM MQ detects that the used range is beyond F00000000000 (if 6-byte RBAs are in use) or FFFF800000000000 (if 8-byte log RBAs are in use), messages CSQI045, CSQI046, CSQI047, and CSQJ032 are issued, warning you to reset the log RBA.

If the RBA value reaches FFF800000000 (if 6-byte log RBAs are in use) or FFFFFFC000000000 (if 8-byte log RBAs are in use) the queue manager terminates with reason code 00D10257.

Once the warning messages about the used log range are being issued, you should plan a queue manager outage during which the queue manager can be converted to use 8-byte log RBAs, or the log can be reset. The procedure to reset the log is documented in Resetting the queue manager's log.

If your queue manager is using 6-byte log RBAs, consider converting the queue manager to use 8-byte log RBAs rather than resetting the queue manager's log, following the procedure documented in Implementing the larger log Relative Byte Address.

The log consists of *log records*, each of which is a set of log data treated as a single unit. A log record is identified either by the RBA of the first byte of its header, or by its log record sequence number (LRSN). The RBA or LRSN uniquely identifies a record that starts at a particular point in the log.

Whether you use the RBA or LRSN to identify log points depends on whether you are using queue sharing groups. In a queue-sharing environment, you cannot use the relative byte address to uniquely identify a log point, because multiple queue managers can update the same queue at the same time, and each has its own log. To solve this, the log record sequence number is derived from a timestamp value, and does not necessarily represent the physical displacement of the log record within the log.

Each log record has a header that gives its type, the IBM MQ subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are four types of log record, described under the following headings:

- Unit of recovery log records
- Checkpoint records
- Page set control records
- CF structure backup records

Unit of recovery log records

Most of the log records describe changes to IBM MQ queues. All such changes are made within units of recovery.

IBM MQ uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that the queue manager has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be reapplied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, IBM MQ takes periodic checkpoints during normal operation. These occur as follows:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in Using CSQ6SYSP.
- At the end of a successful restart.
- At normal termination.
- Whenever IBM MQ switches to the next active log data set in the cycle.

At the time a checkpoint is taken, IBM MQ issues the DISPLAY CONN command (described in DISPLAY CONN) internally so that a list of connections currently in doubt is written to the z/OS console log.

Page set control records

These records register the page sets and buffer pools known to the IBM MQ queue manager at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

Certain dynamic changes to page sets and buffer pools are also written as page set control records, so that the changes can be recovered and automatically reinstated at the next queue manager restart.

CF structure backup records

These records hold data read from a coupling facility list structure in response to a BACKUP CFSTRUCT command. In the unlikely event of a coupling facility structure failure, these records are used, together with unit of recovery records, by the RECOVER CFSTRUCT command to perform media recovery of the coupling facility structure to the point of failure.

Related tasks

Implementing the larger log Relative Byte Address

z/OS How the log is structured

Use this topic to understand the terminology used to describe log records.

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information logged at a particular time forms a *logical* record, with a length that varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term *log record* refers to the *logical* record, regardless of how many *physical* records are needed to store it.

z/OS How the IBM MQ for z/OS logs are written

Use this topic to understand how IBM MQ processes log file records.

IBM MQ writes each log record to a DASD data set called the *active log*. When the active log is full, IBM MQ copies its contents to a DASD or tape data set called the *archive log*. This process is called *offloading*.

Figure 73 on page 244 illustrates the process of logging. Log records typically go through the following cycle:

1. IBM MQ notes changes to data and significant events in recovery log records.
2. IBM MQ processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM Controls Intervals (CI). Each log record is identified by a relative byte address in the range zero through 2^{64} -1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically offloaded to a new archive log data set.

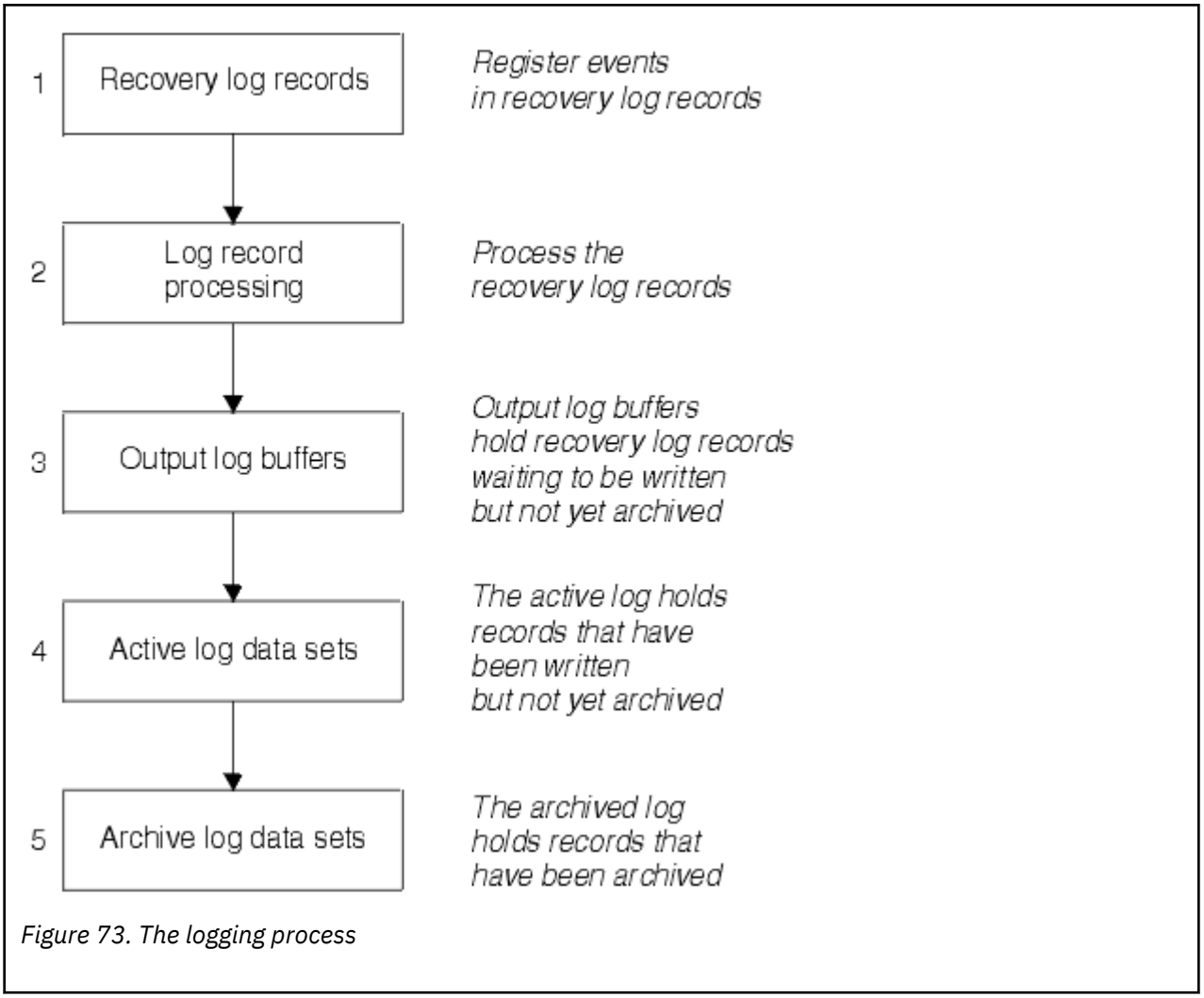


Figure 73. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point, or when an IBM MQ BACKUP CFSTRUCT command is issued.

When the queue manager is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by the queue manager and remain allocated exclusively to IBM MQ until the queue manager terminates.

Dynamically adding log data sets

It is possible to dynamically define new active log data sets while the queue manager is running. This feature alleviates the problem of a queue manager hang when archiving is not able to offload active logs due to a transient problem. See the DEFINE LOG command for more information.

Note: To redefine or remove active logs you must terminate and restart the queue manager.

IBM MQ and Storage Management Subsystem

IBM MQ parameters enable you to specify Storage Management Subsystem (MVS™/DFP SMS) storage classes when allocating IBM MQ archive log data sets dynamically. IBM MQ initiates the archiving of log data sets, but you can use SMS to perform allocation of the archive data set.

Related reference

“When the IBM MQ for z/OS archive log is written” on page 245
Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

z/OS *When the IBM MQ for z/OS archive log is written*

Use this topic to understand the process of copying active logs to archive logs, and when the process occurs.

The process of copying active logs to archive logs is called *offloading*. The relation of offloading to other logging events is shown schematically in Figure 74 on page 245.

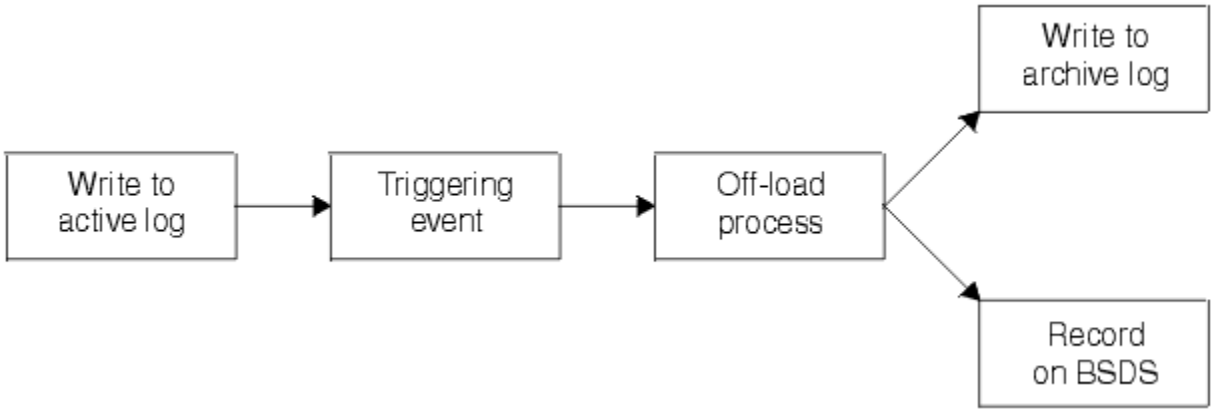


Figure 74. The offloading process

Triggering the offloading process

The offload process of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSC ARCHIVE LOG command.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that was not written becomes the first record of the new data set. Offloading is triggered for the truncated data set as for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 5% full and at 5% increments thereafter, stating the percentage of the log's capacity that is in use. If all the active logs become full, IBM MQ stops processing, until offloading occurs, and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The offload process

When all the active logs become full, IBM MQ runs the offloading process and halts processing until the offloading process has been completed. If the offload processing fails when the active logs are full, IBM MQ abends.

When an active log is ready to be offloaded, a request is sent to the z/OS console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (for further information, see Using CSQ6ARVP) determines whether the request is received. If you are using tape for offloading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the offload process. It does not affect IBM MQ performance unless the operator delays the response for so long that IBM MQ runs out of active logs.

The operator can respond by canceling the offload process. In this case, if the allocation is for the first copy of dual archive data sets, the offload process is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while offloading

A request to stop the queue manager does not take effect until offload processing has finished. If IBM MQ fails while offloading is in progress, offloading begins again when the queue manager is restarted.

Messages during offload processing

Offloaded messages are sent to the z/OS console by IBM MQ and the offloading process. You can use these messages to find the RBA ranges in the various log data sets.

z/OS Larger log Relative Byte Address

This function improves the availability of the queue manager by increasing the period of time before you have to reset the log.

Recovery data is written to the log so that persistent messages are available when the queue manager is restarted. The term log Relative Byte Address (log RBA) is used to refer to the location of data as an offset from the beginning of the log.

Before IBM MQ 8.0, the 6 byte log RBA could address up to 256 terabytes of data. Before this quantity of log records has been written, you have to reset the queue manager's log by following the procedure documented in Resetting the queue manager's log.

Resetting the logs of queue managers is not a quick process, and can require an extended outage, due to the need to reset the page sets as part of the process. For a high use queue manager this operation might typically be done once a year.

From IBM MQ 8.0, the log RBA can be 8 bytes long and the queue manager can now address over 64,000 times as much data (16 exabytes) before the log RBA needs to be reset. The impact of using the larger log RBA is that the size of the log data written increases by a few bytes.

When is this function enabled?

Queue managers created at IBM MQ 9.3.0 or later already have this function enabled.

If the current log RBA is approaching the end of the log RBA range, consider converting the queue manager to use an 8 byte log RBA rather than resetting the queue manager's log. Converting a queue manager to use 8 byte log RBAs requires a shorter outage than resetting the log, and significantly increases the period of time before you have to reset the log.

Message CSQJ034I, issued during queue manager initialization, indicates the end of the log RBA range for the queue manager as configured, and can be used to determine whether 6 byte or 8 byte log RBAs are in use.

How is this function enabled?

8 byte log RBA is enabled by starting the queue manager with a version 2 format BSDS. In summary, this is achieved by:

1. Ensuring that all queue managers in the queue sharing group meet the requirements for enabling 8 byte log RBA
2. Shutting down the queue manager cleanly
3. Running the BSDS conversion utility to create a copy of the BSDS in version 2 format.
4. Restarting the queue manager with the converted BSDS.

Once a queue manager has been converted to use 8 byte log RBAs, it cannot go back to using 6 byte log RBA.

See Implementing the larger log Relative Byte Address for the detailed procedure on how to enable 8 byte log RBAs.

Related tasks

Planning to increase the maximum addressable log range

Related reference

The BSDS conversion utility (CSQJUCNV)

z/OS The bootstrap data set

The bootstrap data set is required by IBM MQ as a mechanism to reference log data sets, and log records. This information is required during normal processing, and restart recovery.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by IBM MQ. It contains the following:

- An inventory of all active and archived log data sets known to IBM MQ. IBM MQ uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  IBM MQ stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each portion of the log held in that data set.
- A *wrap-around* inventory of all recent IBM MQ activity. This is needed if you have to restart the queue manager.

The BSDS is required if the queue manager has an error and you have to restart it. IBM MQ **must** have a BSDS. To minimize the likelihood of problems during a restart, you can configure IBM MQ with dual BSDSs, each recording the same information. Using dual BSDSs is known as running in *dual mode*. If possible, place the copies on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Use dual BSDSs rather than dual write to DASD.

The BSDS is set up when IBM MQ is customized and you can manage the inventory using the change log inventory utility (CSQJU003). For more information about this utility, see Administration de IBM MQ for z/OS. It is referenced by a DD statement in the queue manager startup procedure.

Normally, IBM MQ keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual-mode operation, this is described in the Administration de IBM MQ for z/OS.

The active logs are first registered in the BSDS when IBM MQ is installed. You cannot replace the active logs without terminating and restarting the queue manager.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (IBM MQ does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

You can use the following MQSC command to determine the extent of the log, and the name of the active or archive log data set holding the earliest log RBA, required for various types of media or queue manager recovery:

```
DISPLAY USAGE TYPE(DATASET)
```

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

The BSDS version

The format of the BSDS varies according to its version. Increasing the version of the BSDS allows new features to be used. The following BSDS versions are supported by IBM MQ:

Version 1
Supported by all releases of IBM MQ. A version 1 BSDS supports 6-byte log RBA values.

Version 2
Supported by IBM MQ 8.0 and later. A version 2 BSDS enables 8-byte log RBA values, and up to 310 data sets in each active log copy.

Enabled by default for queue managers created at IBM MQ 9.3.0 or later.

Version 3
Supported by IBM MQ 8.0 and later. The BSDS is automatically converted to version 3, from version 2, when more than 31 data sets are added to either active log copy.

You can determine the version of a BSDS by running the print log map utility (CSQJU004). To convert a BSDS from Version 1 to Version 2, run the BSDS conversion utility (CSQJUCNV).

See “Larger log Relative Byte Address” on page 246 for more information on 6-byte and 8-byte log RBAs.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229. *A* 0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229. *B* 0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the offloading to the new archive log data set continues without the BSDS copy.

z/OS System definition on z/OS

IBM MQ for z/OS uses many default object definitions, and provides sample JCL to create those default objects. Use this topic to understand these default objects, and the sample JCL.

Setting system parameters

In IBM MQ for z/OS, a system parameter module controls the logging, archiving, tracing, and connection environments that IBM MQ uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

Default parameter modules are supplied with IBM MQ for z/OS. If these do not contain the values that you want to use, you can create your own parameter modules using the sample supplied with IBM MQ. The sample is `thlqual.SCSQPROC(CSQ4ZPRM)`.

You can alter some system parameters while a queue manager is running. See the SET SYSTEM, SET LOG, and SET ARCHIVE commands in The MQSC commands.

For more information about defining , see the following topics:

- “Defining system objects for IBM MQ for z/OS” on page 249
- “Tuning your queue manager on IBM MQ for z/OS” on page 254
- “Sample definitions supplied with IBM MQ for z/OS” on page 255

Related concepts
Customize the sample initialization input data sets
Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS
Related tasks
Administering z/OS
Configuring clusters
Monitoring IBM MQ

z/OS Defining system objects for IBM MQ for z/OS

IBM MQ for z/OS requires additional predefined objects for publish/subscribe applications, cluster, and channel control and other system administration functions.

The system objects required by IBM MQ for z/OS can be divided into the following categories:

- Publish/subscribe objects
- System default objects
- System command objects
- System administration objects
- Channels queues
- Cluster queues

- Queue sharing group queues
- Storage classes
- Defining the system object dead-letter queue
- Default transmission queue
- Internal queues
- “Channel authentication queue” on page 253

Publish/subscribe objects

There are several system objects that you need to define before you can use publish/subscribe applications with IBM MQ for z/OS. Sample definitions are supplied with IBM MQ to help you define these objects. These samples are described in CSQ4INSG.

To use publish/subscribe you need to define the following objects:

- A local queue called SYSTEM.RETAINED.PUB.QUEUE, which is used to hold a copy of each retained publication in the queue manager. Each full topic name could have up to one retained publication stored on this queue. If your applications will make use of retained publications on many different topics, or if your retained publication messages are large messages, the requirements for storage for this queue should be carefully planned, including assigning it to its own page set if the storage requirements for it are large. To improve performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.SUBSCRIBER.QUEUE, which is used to hold a persistent copy of the durable subscriptions in the queue manager. To improve performance, you should define this queue with an index type of CORRELID (as shown in the supplied sample queue definition).
- A local queue called SYSTEM.DURABLE.MODEL.QUEUE, which is used as a model for managed durable subscriptions.
- A local queue called SYSTEM.NDURABLE.MODEL.QUEUE, which is used as a model for managed non-durable subscriptions.
- A namelist called SYSTEM.QPUBSUB.QUEUE.NAMELIST, which contains a list of queue names monitored by the queued publish/subscribe interface.
- A namelist called SYSTEM.QPUBSUB.SUBPOINT.NAMELIST, which contains a list of topic objects used by the queued publish/subscribe interface to match topic objects to subscription points.
- A topic called SYSTEM.BASE.TOPIC, which is used as a base topic for resolving attributes.
- A topic called SYSTEM.BROKER.DEFAULT.STREAM, which is the default stream used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.DEFAULT.SUBPOINT, which is the default RFH2 subscription point used by the queued publish/subscribe interface.
- A topic called SYSTEM.BROKER.ADMIN.STREAM, which is the admin stream used by the queued publish/subscribe interface.
- A subscription called SYSTEM.DEFAULT.SUB, which is a default subscription object used to provide default values on DEFINE SUB commands.

System default objects

System default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters "SYSTEM.DEFAULT" or "SYSTEM.DEF." For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these IBM MQ objects:

- Local queues

- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes
- Authentication information

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before you can use the IBM MQ operations and control panels to issue commands to an IBM MQ subsystem.

There are two system command objects:

1. The system-command input queue is a local queue on which commands are put before they are processed by the IBM MQ command processor. It must be called SYSTEM.COMMAND.INPUT. An alias named SYSTEM.ADMIN.COMMAND.QUEUE should also be defined, for compatibility with IBM MQ for Multiplatforms, and for use by the IBM MQ Console and administrative REST API.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

There are two extra objects for use by the IBM MQ Explorer:

- SYSTEM.MQEXPLORER.REPLY.MODEL queue
- SYSTEM.ADMIN.SVRCONN channel

SYSTEM.REST.REPLY.QUEUE is the reply queue used by the IBM MQ administrative REST API.

Commands are normally sent using nonpersistent messages so both the system command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) get nonpersistent messages by default. If you have an application that uses persistent messages for commands, set the DEFTYPE(PERMDYN) attribute for the reply-to queue, because the reply messages to such commands are persistent.

System administration objects

The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are seven system administration objects:

- The SYSTEM.ADMIN.CHANNEL.EVENT queue
- The SYSTEM.ADMIN.COMMAND.EVENT queue
- The SYSTEM.ADMIN.CONFIG.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.QMGR.EVENT queue

- The SYSTEM.ADMIN.TRACE.ROUTE.QUEUE queue
- The SYSTEM.ADMIN.ACTIVITY.QUEUE queue

Channels queues

To use distributed queuing, you need to define the following objects:

- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- A local queue with the name SYSTEM.CHANNEL.INITQ, which is used for channel commands.

You cannot define these queues as shared queues.

Cluster queues

To use IBM MQ clusters, you need to define the following objects:

- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons, you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

You cannot define these queues as shared queues.

Queue sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:

- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by IBM MQ. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Defining the system object dead-letter queue

The dead-letter queue is used if the message destination is not valid. IBM MQ puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, you should regard it as essential, especially if you are using distributed queuing or one of the IBM MQ bridges.

Do **not** define the dead-letter queue as a shared queue. A put to a local queue on one queue manager might get put to the dead letter queue. If the dead letter queue was a shared queue, a dead letter queue handler on a different system could process the message and put it on a queue with the same name, but because this is on a different queue manager, it would be the wrong queue, or have a different security profile. If the queue did not exist, it would fail to reprocess it.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR DEADQ(*queue-name*) command. For more information see Displaying and altering queue manager attributes.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue are not transmitted to the remote queue manager and remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command.

Internal queues

- **Pending data queue**
  - A queue defined for internal use, SYSTEM.PENDING.DATA.QUEUE, supports the use of durable subscriptions in a JMS publish/subscribe environment.
- **JMS 2.0 delivery delay staging queue**
  - If the delivery delay functionality provided by JMS 2.0 is used then an internal staging queue, SYSTEM.DDELAY.LOCAL.QUEUE, must be defined. This queue is used by the queue manager to temporarily store messages sent with a non-zero delivery delay until the delivery delay is completed, and the message is put to its target destination. A sample definition for this queue is provided, commented out, in CSQ4INSG.
  - When you define the SYSTEM.DDELAY.LOCAL.QUEUE queue, you must set the STGCLASS, MAXMSGL and MAXDEPTH attributes for the anticipated number of messages that will be sent with a delivery delay. Additionally when defining the SYSTEM.DDELAY.LOCAL.QUEUE queue ensure that only the queue manager can put messages to this queue. Care should be taken to ensure that no user identifier has the authority to put messages to this queue.

Channel authentication queue

For internal use of channel authentication the SYSTEM.CHLAUTH.DATA.QUEUE queue is required. Sample definitions are supplied with IBM MQ to help you define these objects. This sample is described in CSQ4INSA, which also defines some default rules.

z/OS Tuning your queue manager on IBM MQ for z/OS

There are few simple steps that you can take to ensure that your queue manager is tuned to avoid basic performance problems.

There are a number of ways in which you can improve the performance of your queue manager, which are controlled by queue manager attributes set by the ALTER QMGR command. This section contains information about how you can do this by setting the maximum number of messages allowed on the queue manager, or by performing 'housekeeping' on the queue manager. IBM MQ SupportPac MP16 - IBM MQ for z/OS Planification et optimisation de la capacité gives more information on performance and tuning.

Syncpoints

One of the roles of the queue manager is syncpoint control within an application. An application constructs a unit of work containing any number of MQPUT or MQGET calls terminated with an MQCMIT call.

As the number of MQPUT or MQGET calls within the scope of one MQCMIT increases, the performance cost of the commit increases significantly. Applications, in general, should be designed to not MQPUT/MQGET a large number of messages in a single synchpoint.

You can administratively limit the number of messages within any single syncpoint by using the MAXUMSGS queue manager attribute. If an application exceeds this limit it receives MQRC_SYNCPOINT_LIMIT_REACHED on the MQPUT,MQPUT1, or MQGET call which exceeds the limit. The application should then issue MQCMIT or MQBACK as appropriate.

The default value of MAXUMSGS is 10000. This value can be lowered if you want to enforce a lower limit, which can also help protect against looping applications. Before reducing MAXUMSGS make sure you understand your existing applications to ensure they do not exceed the limit, or can tolerate the MQRC_SYNCPOINT_LIMIT_REACHED return code

Expired messages

Messages that have expired are discarded by the next appropriate MQGET call. However, if no such call occurs, the expired messages are not discarded, and, for some queues, particularly those where message retrieval is done by MessageId, CorrelId, or GroupId and the queue is indexed for performance, many expired messages can accumulate. The queue manager can periodically scan any queue for expired messages, which are then deleted. You can choose how often this scanning takes place, if at all. There are two ways of doing this:

Explicit request
: You can control which queues are scanned and when. Issue the REFRESH QMGR TYPE(EXPIRY) command, specifying the queue or queues that you want to be scanned.

Periodic scan
: You can specify an expiry interval in the queue manager object by using the EXPRYINT attribute. The queue manager maintains information about the expired messages on each queue, and knows at what time a scan for expired messages is worthwhile. Each time that the EXPRYINT interval is reached, the queue manager looks for candidate queues that are worth scanning for expired messages, and scans only those queues that it deems to be worthwhile. It does not scan all queues. This avoids any processor time being wasted on unnecessary scans.

Shared queues are only scanned by one queue manager in the queue sharing group. Generally, the first queue manager to restart or the first to have the EXPRYINT set performs the scan.

Note: You must set the same EXPRYINT value for all queue managers within a queue sharing group.

z/OS Sample definitions supplied with IBM MQ for z/OS

Use this topic as a reference for the sample JCL, and code supplied with IBM MQ for z/OS.

The following sample definitions are supplied with IBM MQ in the thlqual.SCSQPROC library. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in Initialization commands).

Table 22. IBM MQ sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	CSQ4INP1 CSQ4INPR
CSQINP2	CSQ4INSA CSQ4INYS [1] CSQ4INSX CSQ4INSG CSQ4INSR CSQ4INSS CSQ4INSJ CSQ4INSM CSQ4INYG CSQ4INYR CSQ4INYC CSQ4INYD CSQ4INSC
CSQINPT	CSQ4INST CSQ4INYT
Other	CSQ4DISP CSQ4INPX CSQ4IVPQ CSQ4IVPG CSQ4MSTR CSQ4MSRR CSQ4QMIN

Note:

1. The order of these sample definitions is important: an error occurs if INYS, INSX, and INSG are ordered incorrectly.

CSQINP1 samples

Use the sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) when you are using one page set for each class of message, or thlqual.SCSQPROC(CSQ4INPR) when using multiple page sets for the major classes of message. It contains definitions of buffer pools, page set to buffer pool associations, and an

ALTER SECURITY command. Include the sample in the CSQINP1 concatenation of your queue manager started task procedure.

CSQINP2 samples

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:

- System default objects
- System command objects
- System administration objects
- Other objects for system use

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across queue manager shutdown and restart. You must not change the object names, but you can change their attributes if required.

When the following conditions are met, one message is put to the SYSTEM.DURABLE.SUBSCRIBER.QUEUE queue (even if publish subscribe is not active):

- The QMGR attribute PSMODE is set to DISABLED
- The sample object CSQ4INST statement `DEFINE SUB('SYSTEM.DEFAULT.SUB')` is present.

To avoid this, delete or comment out the `DEFINE SUB('SYSTEM.DEFAULT.SUB')` statement.

The JMS 2.0 delivery delay staging queue, SYSTEM.DDELAY.LOCAL.QUEUE only need be defined if JMS 2.0 delivery delay is used. By default, the queue definition is commented out, which you can uncomment if required.

CSQ4INSA system object and authentication sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSA) contains the channel authentication system queue definition. This queue holds the channel authentication records. It also contains the default channel authentication rules.

You must define the objects in this sample if CHLAUTH is ENABLED on the queue manager and you want to run channels, or you want to SET or DISPLAY CHLAUTH record. You only need to define them once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across a queue manager shutdown and restart, you must not change the queue name.

CSQ4INSS system object sample

You can define additional system objects if you are using queue sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains sample commands for use with CF structures and a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required commands.

When you are defining group or shared objects, you need to include them in the CSQINP2 DD concatenation for only one queue manager in the queue sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:

- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INSJ system JMS object sample

Defines queues used in the JMS publish/subscribe domain.

CSQ4INSM system object sample

If you are using advanced message security you must define additional system objects. Sample data set thlqual.SCSQPROC(CSQ4INSM) contains the queue definitions required.

CSQ4INSR object sample

Defines queues used by WebSphere Application Server and brokers.

CSQ4INYD object sample

If you are using distributed queuing and you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:

- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don't have to redefine these objects each time you restart the queue manager).

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed - a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

The sample data set: thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:

- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

CSQ4INYG object sample

The sample data set: thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is - you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the queue manager startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. This is preferable because it means that you don't have to redefine these objects each time that you restart IBM MQ.

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you can make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue
Read the Default transmission queue description before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects
The sample defines an initiation queue named CICS01.INITQ. This queue is used by the IBM MQ -supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4INYS/CSQ4INYR object samples

Storage class definitions for using:

- one page set for each class of message
- multiple page sets for major classes of message

For example, SYSTEM.COMMAND.INPUT uses STGCLASS('SYSVOLAT'), and SYSTEM.CLUSTER.TRANSMIT.QUEUE uses STGCLASS('REMOTE'). In CSQ4INYS, both of those storage classes use the same page set. In CSQ4INYR, those storage classes use different page sets in order to lessen the impact of the transmission queue filling.

CSQINPT samples

CSQ4INST

The sample data set: thlqual.SCSQPROC(CSQ4INST) contains the definition for the system default subscription.

You must define the object in this sample, but you need to do it only once when the publish/subscribe engine is first started. Including the definition in the CSQINPT data set is the best way to do this. It is maintained across queue manager shutdown and restart. You must not change the object name, but you can change their attributes if required.

CSQ4INYT
The sample data set: thlqual.SCSQPROC(CSQ4INYT) contains a set of commands that you might want to run when the publish/subscribe engine is started. This sample displays Topic and Subscription information.

Other

CSQ4DISP display sample

The sample data set: thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your queue manager. This includes the definitions for all IBM MQ objects and definitions such as storage classes and trace. These commands can generate a large amount of output. You can use this sample in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4INPX sample

The sample data set: thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

The sample data sets: thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time the queue manager is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

CSQ4MSTR and CSQ4MSRR samples
These are sample started task procedures for the queue manager: thlqual.SCSQPROC(CSQ4MSTR) and thlqual.SCSQPROC(CSQ4MSRR).

CSQ4MSRR uses CSQ4INYR in the CSQINP2 concatenation so that important queues are spread across different page sets.

You can remove the comments, so that you can use the CSQMINI card for newly created queue managers if required.

CSQ4QMIN sample

A sample QMINI data set, thlqual.SCSQPROC(CSQ4QMIN).

See QMINI data set for details of the QMINI data set and the **TransportSecurity** stanza.

z/OS Recovery and restart on z/OS

Use the links in this topic to find out about the features of IBM MQ for z/OS for restart and recovery.

IBM MQ for z/OS has robust features for restart and recovery. For information about how a queue manager recovers after it has stopped, and what happens when it is restarted, see the following subtopics:

- “How changes are made to data in IBM MQ for z/OS” on page 260
- “How consistency is maintained in IBM MQ for z/OS” on page 261
- “What happens during termination in IBM MQ for z/OS” on page 263
- “What happens during restart and recovery in IBM MQ for z/OS” on page 265
- “How in-doubt units of recovery are resolved” on page 267
- “Shared queue recovery” on page 269

Related concepts

z/OS IBM MQ for z/OS recovery actions

Related tasks

Planning for backup and recovery

z/OS Administering z/OS

Related reference

z/OS Messages for IBM MQ for z/OS

z/OS How changes are made to data in IBM MQ for z/OS

IBM MQ for z/OS must interact with other subsystems to keep all the data consistent. This topic contains information about *units of recovery*, what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single queue manager for an application program, that changes IBM MQ data from one point of consistency to another. A *point of consistency* - also called a *syncpoint* or *commit point* - is a point in time when all the recoverable data that an application program accesses is consistent.

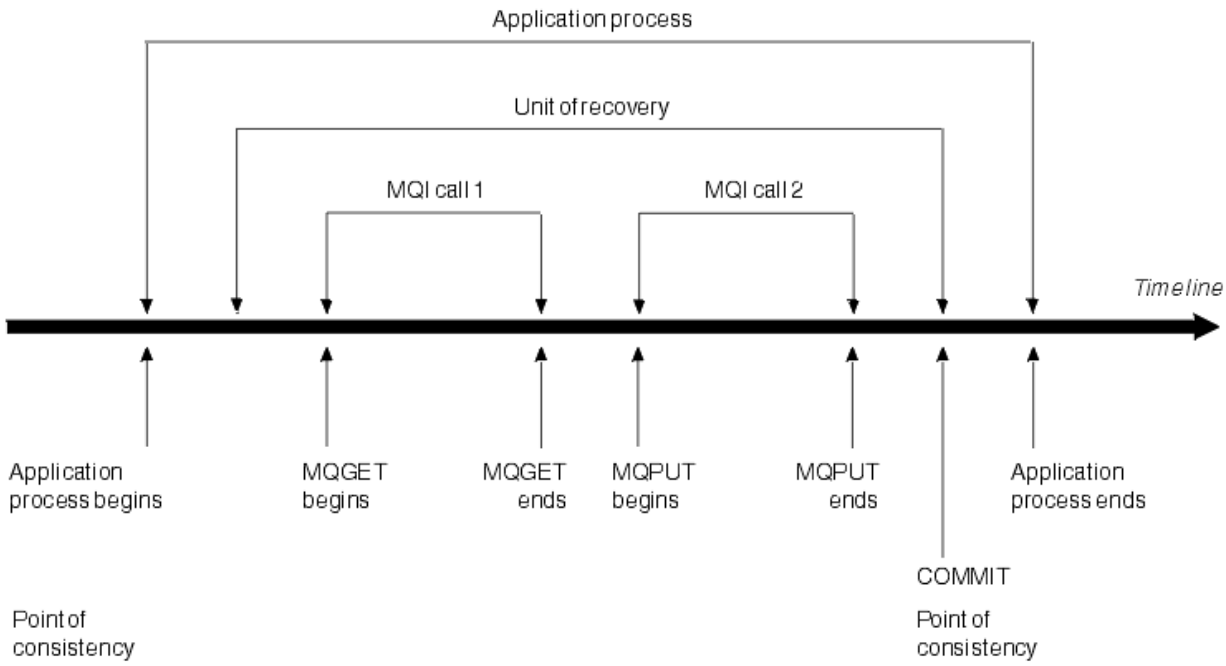


Figure 75. A unit of recovery within an application program

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 75 on page 260 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and IBM MQ cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program

can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, IBM MQ removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, IBM MQ backs out the work. The events are shown in Figure 76 on page 261.

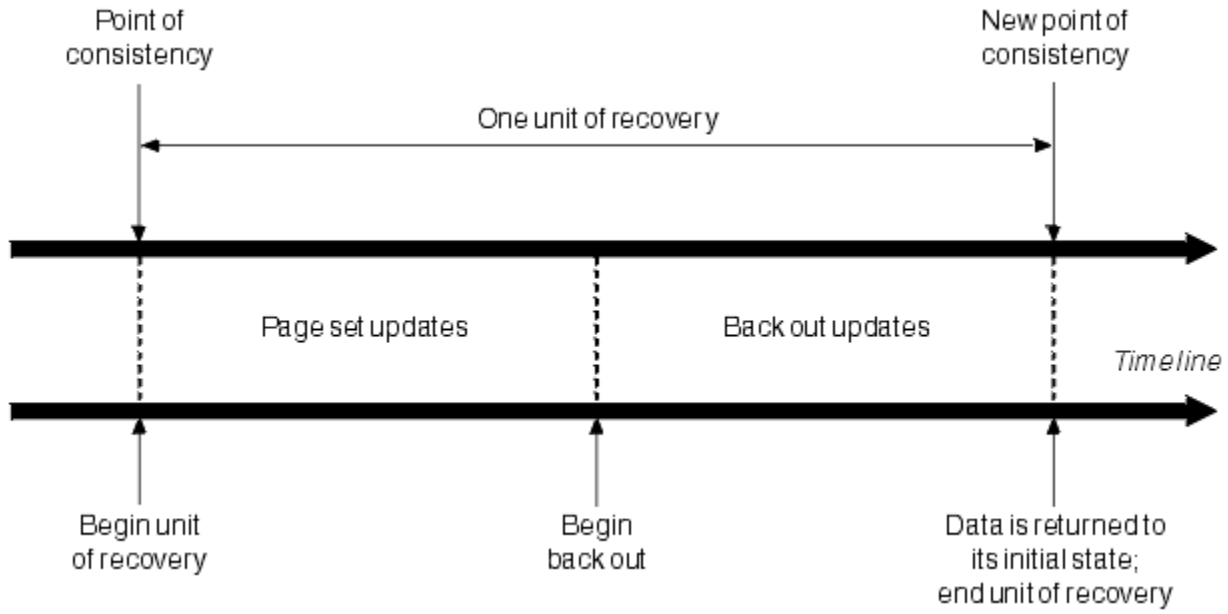


Figure 76. A unit of recovery showing back out

z/OS How consistency is maintained in IBM MQ for z/OS

Data in IBM MQ for z/OS must be consistent with batch, CICS, IMS, or TSO. Any data changed in one must be matched by a change in the other.

Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with IBM MQ, and IBM MQ is always the participant. In the batch or TSO environment, IBM MQ can participate in two-phase commit protocols coordinated by z/OS RRS.

During a *single-phase commit* (for example under TSO or batch), IBM MQ is always the coordinator in the interactions and completely controls the commit process.

In a WebSphere Application Server environment, the semantics of the JMS session object determine whether single-phase or two-phase commit coordination is used.

Consistency with CICS or IMS

The connection between IBM MQ and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit - for transactions that update resources owned by more than one resource manager.

  This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.
- Single-phase commit - for transactions that update resources owned by a single resource manager (IBM MQ).

  This protocol is optimized for logging and message flows.
- Bypass of syncpoint - for transactions that involve IBM MQ but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that IBM MQ uses to communicate with CICS or IMS are as follows:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

   At the end of the phase, the systems communicate. If they agree, each begins the next phase.
2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 77 on page 262 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in IBM MQ on the lower line.

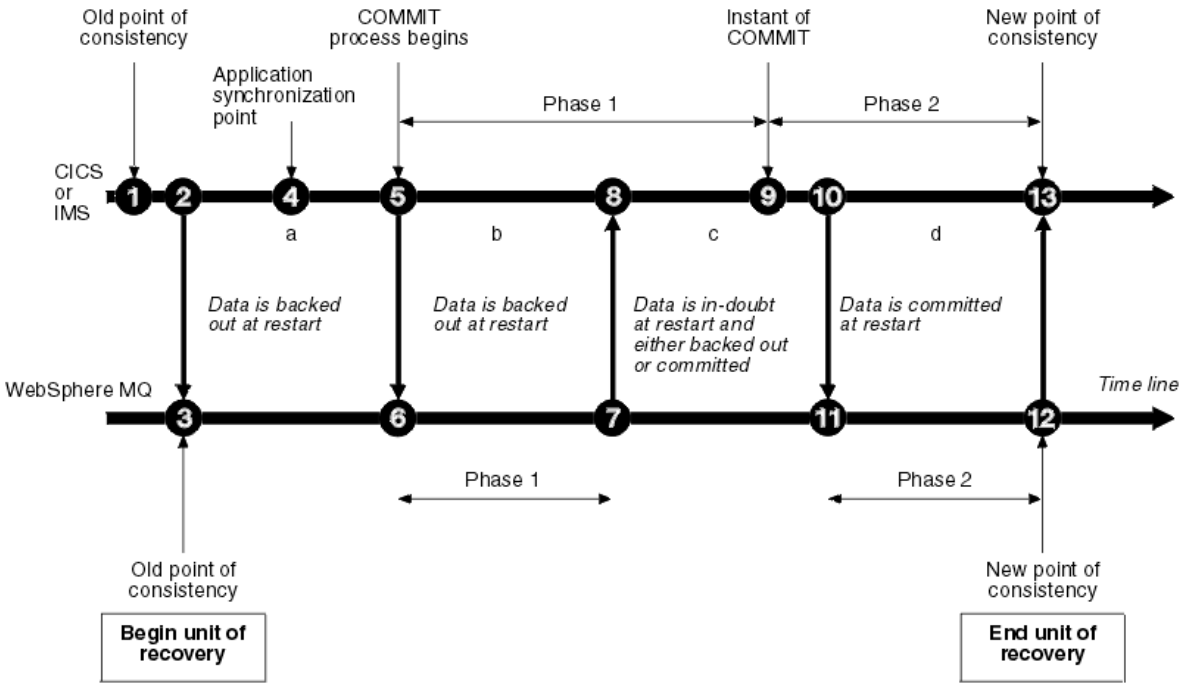


Figure 77. The two-phase commit process

The numbers in the following section are linked to those shown in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls IBM MQ to update a queue by adding a message.
3. This starts a unit of recovery in IBM MQ.
4. Processing continues in the coordinator until an application synchronization point is reached.

5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.
6. As the coordinator begins phase 1 processing, so does IBM MQ.
7. IBM MQ successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit - the irrevocable decision of the two subsystems to make the changes.

   The coordinator now begins phase 2 of the processing - the actual commitment.
10. The coordinator notifies IBM MQ to begin its phase 2.
11. IBM MQ logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for IBM MQ. IBM MQ then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When a queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For some units of recovery, IBM MQ has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 77 on page 262 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be one of the following:

In flight
The queue manager ended before finishing phase 1 (period a or b); during restart, IBM MQ backs out the updates.

In doubt
The queue manager ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, IBM MQ must back out its changes; if it happened after, IBM MQ must make its changes and commit them. At restart, IBM MQ waits for information from the coordinator before processing this unit of recovery.

In commit
The queue manager ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
The queue manager ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure) during restart, IBM MQ continues to back out the changes.

z/OS What happens during termination in IBM MQ for z/OS

A queue manager terminates normally in response to the STOP QMGR command. If a queue manager stops for any other reason, the termination is abnormal.

Note, that during queue manager termination, IBM MQ internally issues the command

```
DISPLAY CONN(*) TYPE(CONN) ALL WHERE (APPLTYPE NE SYSTEMAL)
```

so that you are aware of what threads might prevent the queue manager from completing shutdown.

SYSTEMAL matches APPLTYPES of either SYSTEM or CHINIT, so the DISPLAY CONN command filtering application types not matching SYSTEMAL, returns to the joblog information about threads that could be preventing normal shutdown.

Normal termination

In a normal termination, IBM MQ stops all activity in an orderly way. You can stop IBM MQ using either quiesce, force, or restart mode. The effects are given in Table 23 on page 264.

Table 23. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping a queue manager in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop a queue manager in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when IBM MQ is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop a queue manager, in any mode, the steps are:

1. Connections are ended.
2. IBM MQ ceases to accept commands.
3. IBM MQ ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by IBM MQ so that the restart RBA is recorded on the z/OS console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

IBM MQ resolves any data inconsistencies arising from abnormal termination during restart and recovery.

z/OS What happens during restart and recovery in IBM MQ for z/OS

IBM MQ uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent IBM MQ checkpoint in the log.

Introduction to restart and recovery

After IBM MQ has been initialized, the queue manager restart process takes place as follows:

- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:

- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until IBM MQ recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in “Rebuilding queue indexes” on page 266).

If dual BSDSs are in use, IBM MQ checks the consistency of the time stamps in the BSDS:

- If both copies of the BSDS are current, IBM MQ tests whether the two time stamps are equal. If they are not, IBM MQ issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while the queue manager was stopped. IBM MQ detects the situation at restart.
- If one copy of the BSDS was de-allocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, IBM MQ might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Batch applications are not notified when restart occurs *after* the application has requested a connection.

Understanding the log range required for recovery

During restart, the range of log data which must be read is dependent on many factors:

- At the time of an abnormal termination, there are typically many incomplete units of work in the system. As described earlier, restart processing will bring the system to a state of consistency, which may involve backing out inflight units of work, or recovering locks on indoubt units of work. Unit of work recovery requires that all unit of work log records for inflight, in-backout, and in-doubt units of work are available. IBM MQ will 'shunt' old units of work, so that unit of work recovery can be performed using a much smaller range of log data.
- At the time of an abnormal termination, there are typically many persistent updates which are only held in the buffer pool cache. They have not yet been written to disk. These changes must be read from the log, and reapplied to the data held in page sets. Page set recovery RBAs in the checkpoint describe the lowest log RBA required for updating the page sets to a consistent state.
- If old page sets have been introduced into the system, for example, a page set backup has been introduced to recover from a media failure, all the changes must be read from the log from the time the backup was taken. These changes are reapplied to the data held in the page set being recovered.

Page set recovery RBAs held in page 0 of the page set describe the lowest log RBA required for media recovery of a page set.

- If using persistent messages on shared queues, a range of log data is required to recover CFSTRUCTs which are holding persistent messages. The earliest log data that would be required to perform a CFSTRUCT recovery, is from around the time of the old CFSTRUCT BACKUP.

During normal running, the DISPLAY USAGE TYPE(DATASET) command can be used to view the recovery log range associated with these factors (it is unable to provide information due to reintroducing old page sets, of course). To avoid any issues that might prolong a queue manager restart in the event of an abnormal termination, regularly monitor the values output from DISPLAY USAGE TYPE(DATASET).

In addition, the queue manager issues informational messages relating to these factors:

- CSQJ160I and CSQJ161I warn of long running units of work.
- CSQR026I and CSQR027I provide information about whether these long running units of work have been successfully shunted.
- CSQE040I and CSQE041E warn that structure backups are getting old, and consequently a RECOVER CFSTRUCT operation would take a long time.

Determining which application has a long running unit of work

It is possible to determine the application with the long-running unit of work. To do this, you use the DISPLAY CONN command.

The DISPLAY CONN command returns connection information for all the applications connected to the queue manager, together with additional information that helps you determine which application(s) currently have a long-running unit of work. The information returned by the DISPLAY CONN command is similar to the information returned by the DISPLAY QSTATUS command, but the main difference is that DISPLAY CONN displays information about objects, and transactional information for a particular connection, rather than details of which connections are associated with a particular object.

For each connected application, the DISPLAY CONN command returns the following information:

- Basic information including the Connection Id and PID.
- Transactional information for that connection, including the time and date when the transaction was created (that is, when the first MQGET/PUT was made under syncpoint), and when the transaction first wrote to the log.
- Log time information indicating which application still has a long running unit of work.
- A list of all objects that the connection currently has open. Details for each object are returned as a separate message, with the Connection Id used as a key. Because there are different types of object such as queues and queue managers, the information displayed with the object is specific to its object type.

Rebuilding queue indexes

To increase the speed of MQGET operations on a queue where messages are not retrieved sequentially, you can specify that you want IBM MQ to maintain an index of the message or correlation identifiers or groupid for all the messages on that queue.

When a queue manager is restarted, these indexes are rebuilt for each queue. This applies only to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this increases the time taken to restart the queue manager.

You can choose to have indexes rebuilt asynchronously to queue manager startup by using the QINDXBLD parameter of the CSQ6SYSP macro. If you set QINDXBLD=NOWAIT, IBM MQ restarts without waiting for indexes to rebuild.

z/OS How in-doubt units of recovery are resolved

If IBM MQ loses its connection to another resource manager, it typically attempts to recover all inconsistent objects at restart.

If IBM MQ loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information required to resolve in-doubt units of recovery must come from the coordinating system. The next sections describe the process of resolution for different environments.

- How in-doubt units of recovery are resolved from CICS
- How in-doubt units of recovery are resolved from IMS
- How in-doubt units of recovery are resolved from RRS
- How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

How in-doubt units of recovery are resolved from CICS

Under some circumstances, CICS cannot run the IBM MQ process to resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the Messages, codes achèvement et codes anomalie IBM MQ for z/OS manual.

The resolution of in-doubt units does not effect CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while IBM MQ is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and IBM MQ. If a queue manager abends while connected to CICS, it is possible for CICS to commit or back out work without IBM MQ being aware of it. When the queue manager restarts, that work is termed *in doubt*.

IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from IBM MQ, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

For all resolved units, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the Administration de IBM MQ for z/OS.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMSdoes not effect DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801'. The existence of in-doubt units of recovery does not imply that DL/I records are locked until IBM MQ connects.

During queue manager restart, IBM MQ makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to IBM MQ, IMS indicates to IBM MQ whether to commit or back out units of work marked in IBM MQ as in doubt.

When in-doubt units are resolved:

1. If IBM MQ recognizes that it has marked an entry for commit and IMS has marked it to be backed out, IBM MQ issues message CSQQ010E. IBM MQ issues this message for all inconsistencies of this type between IBM MQ and IMS.
2. If IBM MQ has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, IBM MQ updates queues as necessary and releases the corresponding locks.

IBM MQ maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the Administration de IBM MQ for z/OS.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to IBM MQ, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between IBM MQ and other RRS-participating resource managers. If a failure occurs when IBM MQ has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and IBM MQ, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. IBM MQ cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to IBM MQ resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, IBM MQ sends one of the following messages to the z/OS console:

- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the Messages, codes achèvement et codes anomalie IBM MQ for z/OS manual.

For all resolved units of recovery, IBM MQ updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the Administration de IBM MQ for z/OS.

How in-doubt units of recovery with a GROUP unit of recovery disposition are resolved

In-doubt transactions that have a GROUP unit of recovery disposition can be resolved by the transaction coordinator by any queue manager in the queue sharing group (QSG) where the GROUPUR queue manager attribute is enabled. Whenever a transaction coordinator reconnects it typically requests a list of any outstanding in-doubt transactions and then resolves them using information from its logs.

When a transaction coordinator, that has connected with a GROUP unit of recovery disposition, requests the list of in-doubt transactions, the list returned comprises all in-doubt transactions with a GROUP unit of recovery disposition that exist throughout the queue sharing group. This list is not dependent on which queue manager those in-doubt transactions were started on. A queue manager processing such a request compiles the list by communicating with all other active queue managers in the queue sharing group using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The queue manager then reads the logs of any inactive queue managers, from their last checkpoint, to identify any additional in-doubt transactions that they would have reported had they been active.

When a transaction coordinator requests the resolution of an in-doubt transaction, the queue manager to which it is connected identifies whether the transaction was originated on itself and if so resolves it in the same way as transactions with a QMGR unit of recovery disposition. If the transaction was originated on another active queue manager in the QSG, a request to complete the resolution is routed to that queue manager using the SYSTEM.QSG.UR.RESOLUTION.QUEUE. In the case where the transaction was originated on an inactive queue manager in the QSG, any shared-queue work is resolved immediately, and a request to resolve any remaining private queue work is placed on the SYSTEM.QSG.UR.RESOLUTION.QUEUE. The inactive queue manager processes this request upon start-up before accepting new work. In this scenario, the original queue manager's logs still reflect that the unit of recovery is in doubt until it has restarted and processed the request.

z/OS Shared queue recovery

Use this topic to understand IBM MQ recovery and resilience of various components in the queue sharing group environment.

Transactional recovery

When an application issues a MQBACK call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread-level information stored in the queue manager ensures that the in-flight unit of work is rolled back. MQPUT and MQGET operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the coupling facility structures that it is currently connected to. If the connection between the z/OS instance and the coupling facility fails (for example, physical link failure or power-off of a coupling facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the coupling facility structures involved. Other queue managers in the same queue sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery is performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery, often referred to as Peer Level Recovery (PLR), is performed on a structure by structure basis and it is possible for a single queue manager to participate in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that IBM MQ resets any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared Db2 repository used by the queue sharing group. Ensure that adequate procedures are in place for the backup and recovery of the Db2 tables used to hold IBM MQ objects. You can also use the IBM MQ CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine IBM MQ objects, including shared queue and group definitions stored in Db2.

Logging

Queue sharing groups can support persistent messages, because the messages on shared queues can be logged in the queue manager logs.

Coupling facility and structure failures

There are two types of failure that can be reported for a coupling facility (CF) structure: structure failure and loss of connectivity. Sysplex services for data sharing (XES) inform IBM MQ of a CF structure failure or a CF failure with a structure failure event. If XES creates a loss of connectivity event this does not necessarily indicate that there is a problem with the structure, it might be that there is no connection available to communicate with the structure. It is possible that not all of the queue managers receive a loss of connectivity event for the structure; it depends on the configuration of connections to the CF. A loss of connectivity event can also be received because of operator commands, for example VARY PATH OFFLINE or CONFIG CHP OFFLINE.

The CF structures that are used by IBM MQ can be configured to use system-managed duplexing. This means that if there is a single failure, system-managed failover processing hides the failure of a structure or the loss of connectivity, and the queue manager is not informed of the failure. If there is a failure of both instances of a duplexed structure or connection, the queue manager receives the appropriate event and handles it in the same way as a failure event for a simplex structure. Details of how the queue manager handles the events are described in Scenarios.

In the unlikely event of a CF or structure failure, any nonpersistent messages stored in the affected application structures are lost. You can recover persistent messages using the RECOVER CFSTRUCT command. If a recoverable application structure has failed, any further application activity to this structure is prevented until the structure has been recovered.

To ensure that you can recover a CF structure in a reasonable period of time, take frequent backups, using the BACKUP CFSTRUCT command. You can choose to perform the backups on any queue managers in the queue sharing group or dedicate one queue manager to perform all the backups. Automate the process of taking backups to ensure that they are taken on a regular basis.

Each backup is written to the active log data set of the queue manager taking the backup. The shared queue Db2 repository records the name of the CF structure being backed up, the name of the queue manager doing the backup, the RBA range for this backup on that queue manager's log, and the backup time.

The administration structure contains information about incomplete units of work on shared queues at the time of any application structure failure so the administration structure must be available during RECOVER CFSTRUCT processing. If the administration structure has failed, all the queue managers in the queue sharing group must have rebuilt their administration structure entries before you can issue the RECOVER CFSTRUCT command.

Queue managers rebuild their administration structure entries automatically and without terminating. If a queue manager is not running at the time of the failure, its administration structure entries can be rebuilt by another queue manager in the queue sharing group that is running at the same or higher level.

To recover an application structure, issue a RECOVER CFSTRUCT command to the queue manager that you want to perform the recovery. You can recover a single CF structure or you can recover several CF structures simultaneously. You can recover using any queue manager in the queue sharing group, it does not have to be the one that performed the backup, or one that has been connected previously to the failed structure.

The RECOVER CFSTRUCT command uses the backup, located through the Db2 repository information (Db2 must therefore be available on the queue manager where recovery is being carried out), and recovers this to the point of failure.

The RECOVER CFSTRUCT command does this by applying log records from every queue manager in the queue sharing group that has performed an MQPUT or MQGET between the start of the backup and the time of failure, to any shared queue that maps to the CF structure. The resulting merging of the logs might require reading a considerable amount of log data because all the log data written by participating queue managers since the backup is read. You are strongly recommended to make frequent (for example, hourly) backups, especially if there are large messages within the backup.

Structure failure scenarios

Scenarios

If a failure is reported for a CF structure, the action taken by connected queue managers depends on the following:

- The type of failure reported by the XES component of z/OS to IBM MQ.
- The structure type (application or administration)
- The queue manager level
- The CFLEVEL of the IBM MQ CFSTRUCT object (2, 3, 4 or 5. This is not the CFLEVEL of the CFCC microcode)
- The RECAUTO attribute of an IBM MQ CFSTRUCT object at CFLEVEL(5)

The following scenarios describe what happens when a failure is reported for the administration structure:

- If a structure failure event is received for the administration structure, the structure is reallocated and rebuilt automatically without the queue manager terminating. A new instance of the structure is allocated by XES when a queue manager attempts to connect to it.

  When the queue manager has connected to the new instance of the structure, the queue manager writes the entries for itself into the structure. This processing is carried out by the queue manager and is not part of XES rebuild processing.

If a queue manager was not running at the time of the failure, or terminates before recovery of its part of the administration structure has been completed, its administration structure entries can be rebuilt by another queue manager in the queue sharing group.

Administration structure entries of a queue manager can only be rebuilt by another queue manager running at the same level or higher. If administration structure entries of a queue manager cannot be rebuilt by another queue manager in the queue sharing group, restart the queue manager so that it can complete the rebuild of its part of the structure.

Certain actions are suspended until the administration structure entries for all queue managers have been rebuilt. The suspended actions include the following:

- Opening and closing of shared queues.
- Committing or backing out units of recovery.
- Serialized applications connecting to or disconnecting from the queue manager.
- Backing up or recovering an application structure.

Any serialized applications that have already connected to the queue manager can continue processing. Any serialized application attempting to connect with the MQCNO_SERIALIZE_CONN_TAG_QSG or MQCNO_RESTRICT_CONN_TAG_QSG parameters receive the MQRC_CONN_TAG_NOT_USABLE return code.

When the administration structure entries for the queue manager have been rebuilt, the suspended actions are resumed.

The following scenarios describe what happens when a failure is reported for an application structure:

- If a structure failure event is received for an application structure, and the CFLEVEL is 1 or 2, the queue manager terminates. Restart the queue manager. The first queue manager to attempt to connect to the structure again causes XES to allocate a new instance of the structure.
- If a structure failure event is received for an application structure, and the CFLEVEL is 3, 4, or 5 the queue managers connected to the structure continue to run. Applications that do not use the queues in the failed structure can continue normal processing.

  However, applications that attempt operations on queues in the failed structure receive an MQRC_CF_STRUC_FAILED error until the structure has been successfully rebuilt, at which point the application can open the queues again.

  Structure rebuild is started automatically for CFLEVEL(5) application structures defined with RECAUTO(YES). Otherwise, the structure will be rebuilt when the RECOVER CFSTRUCT command is issued.

Resilience to coupling facility connectivity failures

What is resilience to coupling facility connectivity failures?

Resilience to coupling facility connectivity failures refers to the ability of queue managers in a queue sharing group to tolerate loss of connectivity to a coupling facility structure without terminating. This function also attempts to rebuild the structure in another coupling facility with better connectivity in order to regain access to shared queues as soon as possible.

What is partial loss of connectivity?

IBM MQ defines partial loss of connectivity as a situation where one or more systems in the sysplex lose connectivity to the coupling facility where the structure being accessed by the system is allocated, but at least one system in the sysplex maintains connectivity to the same coupling facility.

What is total loss of connectivity?

IBM MQ defines a total loss of connectivity as a situation where no systems in the sysplex have connectivity to the coupling facility and the structure allocated within it.

Why would you enable this function?

Resilience to coupling facility connectivity failures improves the availability of IBM MQ, allowing non-shared queues to remain available after a queue manager has lost connectivity to one or more coupling facility structures. Additionally, queue managers that lose connectivity to a coupling facility structure automatically attempt to rebuild the structure in another available coupling facility, improving the availability of the shared queues within the queue sharing group.

Considerations when enabling this function

A queue manager that tolerates loss of connectivity to coupling facility structures without terminating may not be able to reconnect to a coupling facility structure for some time if there is no alternative coupling facility available. Shared queues defined on a structure that has suffered loss of connectivity remain unavailable until connectivity to the structure is restored. In this situation, applications that connect into members of the queue sharing group in order to perform shared queue work may find that the shared queues they need to access are not available. To avoid this situation it is recommended that queue managers should be configured to terminate when connectivity to a coupling facility structure is lost. This termination forces applications to connect to another member of the queue sharing group that has connectivity to the coupling facility structures where the shared queues the application requires are defined.

Managing Resilience to coupling facility connectivity failures

How do I enable this functionality?

The following steps must be performed in order to enable resilience to coupling facility connectivity

1. Ensure that the CFRM couple data set has been formatted to support system-managed rebuild. This allows queue managers to initiate a system-managed rebuild to re-create a structure into an available coupling facility. Use the **DISPLAY XCF,COUPLE,TYPE=CFRM** command to determine the format of the CFRM couple data set. To support system-managed rebuild, the CFRM couple data set should be formatted by specifying:

   ```
   "ITEM NAME(SMREBLD) NUMBER(1)"
   ```

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information on formatting a CFRM couple data set.
2. Ensure that an alternative coupling facility is available and is in the CFRM preference list for all IBM MQ coupling facility structures. This enables the queue managers to attempt to rebuild structures into an alternative available coupling facility to restore access to the structures as soon as possible.

   IBM MQ structures must be defined with ENFORCEORDER(NO) in CFRM policy, so that XCF is able to choose the optimum CF in the configuration if IBM MQ needs to reallocate the structure.

   Refer to the z/OS MVS Setting Up a Sysplex documentation for more information about structure preference lists.
3. Alter all application coupling facility structures that need to tolerate loss of connectivity to CFLEVEL(5). This is the minimum level that can tolerate a loss of connectivity.
4. Determine the values required for the **QMGR CFCONLOS** and the **CFSTRUCT CFCONLOS** attributes and alter these accordingly. The **QMGR CFCONLOS** attribute controls whether loss of connectivity to the administration structure is tolerated, and the **CFSTRUCT CFCONLOS** attribute controls whether loss of connectivity is tolerated by each application coupling facility structure. If the default values for these attributes are retained, the queue manager terminates following loss of connectivity to any coupling facility structure.
5. Determine the values required for the **CFSTRUCT RECAUTO** attribute for each application coupling facility structure, and alter these accordingly. This attribute controls whether coupling facility structures should be automatically recovered using logged data following total loss of connectivity. If the default value for this attribute is retained, no automatic recovery is performed for application structures following total loss of connectivity.

Scenario 1 - Loss of connectivity to the administration structure

Queue managers can tolerate loss of connectivity to the administration structure without terminating.

When connectivity to the administration structure is lost by any queue manager that has been configured to tolerate loss of connectivity to the administration structure, all members of the queue sharing group disconnect from the administration structure. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in the coupling facility with the best connectivity to all systems in the sysplex, and rebuild the administration structure data.

Note: This may not necessarily be the coupling facility which has the best connectivity to all systems that have active queue managers.

If a queue manager cannot reconnect to the administration structure, for example because none of the coupling facilities in the CFRM preference list for the administration structure are available, some shared queue operations remain unavailable until the queue manager can successfully reconnect to the administration structure and rebuild its administration structure data. Reconnection occurs automatically when a suitable coupling facility becomes available on the system.

Failure to connect to the administration structure during queue manager startup as a result of a lack of connectivity to the coupling facility, or no suitable coupling facility available to allocate the structure, is not tolerated. All active queue managers in the queue sharing group then attempt to reconnect to the administration structure, causing it to be reallocated in another coupling facility if one is available, and rebuild the administration structure data.

Scenario 2- Loss of connectivity to the application structure

Loss of connectivity to application structures at **CFLEVEL(5)** or higher can be tolerated without the queue manager terminating. Queue managers connected to application structures at **CFLEVEL(4)** or lower, or structures at **CFLEVEL(5)** that have not been configured to tolerate loss of connectivity, abend with reason code 00C510AB when connectivity to the structure is lost.

When connectivity is lost to an application structure that has been configured to tolerate loss of connectivity, all queue managers that lost connectivity to the structure disconnect. The subsequent behavior of the queue manager depends on whether the loss of connectivity is partial or total.

Partial loss of connectivity to an application structure

If the loss of connectivity is determined to be partial, queue managers that have lost connectivity to the structure attempt to initiate a system-managed rebuild in order to move the structure to another coupling facility with improved connectivity. If this rebuild is successful, both persistent and non-persistent messages in the structure are copied to the other coupling facility, and access to queues on the structure is restored. Queue managers that did not lose connectivity remain connected to the structure, however, operations that access the structure are delayed during the system-managed rebuild process.

If an application structure cannot be rebuilt to another coupling facility with improved connectivity, or some queue managers still do not have connectivity to the structure after it has been rebuilt in another coupling facility, queues defined on the structure remain unavailable on the queue managers that do not have connectivity to the structure until connectivity is restored to the coupling facility. Queue managers automatically reconnect to the structure when it becomes available and access to the shared queues defined on the structure are restored.

Total loss of connectivity to an application structure

If all MVS systems in the sysplex have lost connectivity to the coupling facility that the application structure is allocated in, z/OS deallocates the structure from the coupling facility whenever an attempt is made to reconnect to the structure. It is possible for the queue manager to attempt to reconnect to the structure for several reasons, such as an attempt by an application to open a shared queue, or a notification from the system that new coupling facility resources may have become available. It is therefore likely that all non-persistent messages in the affected structure are lost following total loss of connectivity to an application structure.

Recoverable application structures are automatically recovered following total loss of connectivity, if they have been defined with **RECAUTO(YES)**. The recovery starts almost immediately if an alternative coupling facility is available to allocate the structure in, or whenever such a coupling facility becomes available. If a structure has not been defined with **RECAUTO(YES)**, recovery can be started by issuing the **RECOVER CFSTRUCT** command. This recovers all persistent messages in the structure, but all non-persistent messages are lost. As this process involves reading the queue manager log it can take some time to complete, therefore it is recommended that structure backups be taken regularly to reduce the time until access to the shared queues on the structure is restored.

Queue managers attempt to reconnect to non-recoverable application structures as soon as an application attempts to open a shared queue that is defined on the structure or a notification is received from the system that new coupling facility resources have become available. If a suitable coupling facility is available to allocate the structure in, a new structure is allocated and access to the shared queues defined on the structure is restored. As persistent messages cannot be put to queues defined in non-recoverable structures, all messages on the shared queues are lost.

Operational behavior

If an IBM WebSphere MQ 7.1, or later, queue manager, configured to tolerate loss of connectivity to a particular coupling facility structure loses connectivity, the members of the queue sharing group attempt to automatically recover from the failure and reconnect to the structure. This activity may involve reallocating the structure in another coupling facility with better connectivity if one is available. However, operator intervention may still be required to recover from the loss of connectivity.

Typically the required operator action is to:

1. Resolve the cause of the failure that resulting in the loss of connectivity.
2. Ensure that a coupling facility where the IBM MQ structures can be allocated is available on all systems in the sysplex

Any structures that have been automatically reallocated in another coupling facility after the loss of connectivity event, can be moved to the coupling facility with the optimal connectivity to all queue managers in the queue sharing group. If required, this can be done by initiating the system-managed rebuild command **SETXCF START,REBUILD** as documented in z/OS MVS System Commands Reference.

In the case of a partial loss of connectivity to an application structure, the queue managers that lost connectivity to the structure attempt to initiate a system-managed rebuild. This process only allocates the structure in another coupling facility if that coupling facility has connectivity to all active queue managers currently connected to the structure. Therefore, it is possible that where the majority of queue managers in a queue sharing group have lost connectivity to an application structure, they are unable to rebuild the structure into another coupling facility due to the queue managers that are still connected to the original structure. In this situation the queue managers that are still connected to the original structure can be shut down to allow the structure to be rebuilt, or the **RESET CFSTRUCT ACTION(FAIL)** command can be issued to fail the structure. Recovery can be initiated on applicable structures by issuing the **RECOVER CFSTRUCT** command.

Note: When failing and recovering the structure, all non-persistent messages on the structure are lost.

z/OS Security concepts in IBM MQ for z/OS

Use this topic to understand the importance of security for IBM MQ, and the implications of not having adequate security settings on your system.

Why you must protect IBM MQ resources

IBM MQ handles the transfer of information that is potentially valuable. Applying security ensures that the resources IBM MQ owns and manages are protected from unauthorized access. Such access might lead to the loss or disclosure of the information.

You should ensure that none of the following resources are accessed or changed by any unauthorized user or process:

- Connections to IBM MQ
- IBM MQ objects such as queues, processes, and namelists
- IBM MQ transmission links, that is, IBM MQ channels
- IBM MQ system control commands
- IBM MQ messages
- Context information associated with messages

To provide the necessary security, IBM MQ uses the z/OS system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Security Server (previously known as RACF). IBM MQ does no security verification of its own. Where distributed queuing or clients are being used, you might require additional security measures, for which IBM MQ provides channel authentication records, channel exits, the MCAUSER channel attribute, and TLS.

The decision to allow access to an object is made by the ESM and IBM MQ follows that decision. If the ESM cannot make a decision, IBM MQ prevents access to the object.

What happens if you do not protect IBM MQ resources

If you do nothing about security, the most likely effect is that *all* users can access and change *every* resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for z/OS.

To enable security checking you must do the following:

- Install and activate an ESM (for example, Security Server).
- Define the MQADMIN class if you are using an ESM other than Security Server.
- Activate the MQADMIN class.

You must consider whether using mixed-case resource names would be beneficial to your enterprise. If you do use mixed-case resource names in your ESM profiles you must define and activate the MXADMIN class.

z/OS Data Set Encryption

Data Set Encryption (DSE) provides the capability to encrypt z/OS data sets, so that the data they contain can only be viewed or modified by user IDs granted the specific permission. This provides encryption of data at rest in the file system, and prevents inadvertent disclosure of sensitive information to users who have a legitimate business need and permissions to manage the data sets themselves.

Prior to IBM MQ for z/OS 9.1.4, IBM MQ for z/OS does not support use of DSE with the active logs, page sets, and shared message data sets (SMDS) that provide the primary persistence mechanisms for IBM MQ messages.

Instead, Advanced Message Security provides an end-to-end encryption solution for IBM MQ messaging, which encompasses the entire IBM MQ network, encryption of data in flight, at rest, and even inside the runtime IBM MQ processes.

Other VSAM and sequential data sets used in an IBM MQ subsystem can be encrypted using DSE. For example:

- Bootstrap data set (BSDS)
- Sequential files holding system configuration (MQSC) commands read at startup using CSQINPx DDNAMEs
- IBM MQ archive logs, often used for long term archival of IBM MQ log data for audit purposes.

You can encrypt using DSE by allocating a dataclass that is defined with a data set key label. For more information, see Planning your log archive storage.

From IBM MQ for z/OS 9.1.4, IBM MQ for z/OS supports use of DSE with the active logs and page sets in addition to the support provided in earlier releases.

IBM MQ for z/OS does not support use of DSE for shared message data sets (SMDS).

See the section, confidentiality for data at rest on IBM MQ for z/OS with data set encryption. for more information.

Related concepts
Security concepts
Channel authentication records
Authority to work with IBM MQ objects on z/OS
Cryptographic security protocols: TLS
Related tasks
Setting up security on z/OS
Comparing link level security and application level security
Related reference
Messages for IBM MQ for z/OS

z/OS Security controls and options in IBM MQ for z/OS

You can specify whether security is turned on for the whole IBM MQ subsystem, and whether you want to perform security checks at queue manager or queue sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done for the whole queue manager. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to IBM MQ (including clients and channels), you can turn security checking off for the queue manager or queue sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue sharing group, no other IBM MQ checking is done; if you leave it turned on, IBM MQ checks your security requirements for other IBM MQ resources.

You can also turn security on or off for particular sets of resources, such as commands.

Queue manager or queue sharing group level checking

You can implement security at queue manager level or at queue sharing group level. If you implement security at queue sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier. It also makes it easy to add a new queue manager to the queue sharing group because it inherits the existing security profiles.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue sharing group that requires different levels of security to the other queue managers in the group.

Queue sharing group level security

Queue sharing group level security checking is performed for the entire queue sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each z/OS image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access only at the queue sharing group level.

You can use queue sharing group level security profiles to protect all types of resource, whether local or shared.

Queue manager level security
You can use queue manager level security profiles to protect all types of resource, whether local or shared.

Combination of both levels
You can use a combination of both queue manager and queue sharing group level security.

You can override queue sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

For more information, see Profiles to control queue sharing group or queue manager level security.

Controlling the number of user IDs checked

RESLEVEL is a Security Server profile that controls the number of user IDs checked for IBM MQ resource security. Normally, when a user attempts to access an IBM MQ resource, Security Server checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile for each queue manager. Control is implemented by the access that a user ID has to this profile.

Mixed case or uppercase IBM MQ RACF classes

You can now use mixed case RACF profile support, which allows you to use mixed case resource names and define IBM MQ RACF profiles to protect them.

You can choose to either:

- Continue using uppercase only IBM MQ RACF Classes as in previous releases, or
- Use the new mixed case IBM MQ RACF classes.

Without the use of mixed case RACF profiles, you can still use mixed case resource names in IBM MQ for z/OS ; however, these resource names can only be protected by generic RACF profiles in the uppercase IBM MQ classes. When using mixed case IBM MQ RACF profile support you can provide a more granular level of protection by defining IBM MQ RACF profiles in the mixed case IBM MQ classes.

z/OS Resources you can protect in IBM MQ for z/OS

When a queue manager starts, or when instructed by an operator command, IBM MQ for z/OS determines which resources you want to protect.

You can control which security checks are performed for each individual queue manager. For example, you can implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager. It is done by issuing an MQCONN or MQCONNX request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular queue manager. However, if you do this any user can connect to that queue manager.

For the CICS adapter, only the CICS address space user ID is used for the connection security check, not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to IBM MQ, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

You can turn connection security checking on or off at either queue manager or queue sharing group level.

Command security

Command security checking is carried out when a user issues an MQSC command from any of the sources described in Sources from which you can issue MQSC and PCF commands on IBM MQ for z/OS. You can make a separate check on the resource specified by the command as described in “Command resource security” on page 279.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSC commands are entered from a console, the console must have the z/OS SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You must control who is allowed to update these data sets through normal data set protection.

You can turn command security checking on or off at either queue manager or queue sharing group level.

Command resource security

Some MQSC commands, for example defining a local queue, involve the manipulation of IBM MQ resources. When command resource security is active, each time a command involving a resource is issued, IBM MQ checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning "PAYROLL". If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

You can turn command resource security checking on or off at either queue manager or queue sharing group level.

Channel security considerations

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol, although you can use TLS. If you are using APPC, you can flow user ID information from the sending MCA through the network to the destination MCA for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

For more information about the types of channel security available see Channel authentication records and Security exit overview

Related reference

"API-resource security in IBM MQ for z/OS" on page 280
Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

z/OS *API-resource security in IBM MQ for z/OS*

Resources are checked when an application opens an object with an MQOPEN or an MQPUT1 call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into the following checks:

- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (using the MQOO_BROWSE option), but not to remove messages from the queue (using one of the MQOO_INPUT_* options).
If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an MQOPEN or MQPUT1 call).

You can turn queue security checking on or off at either queue manager or queue sharing group level.

Process

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

You can turn process security checking on or off at either queue manager or queue sharing group level.

Namelist

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

You can turn namelist security checking on or off at either queue manager or queue sharing group level.

Alternate user

Alternate user security controls whether one user ID can use the authority of another user ID to open an IBM MQ object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternative user ID when opening the reply-to queue.

The alternative user ID is specified in the *AlternateUserId* field of the object descriptor (MQOD).

You can use alternative user IDs on any IBM MQ object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for z/OS data set security.

If alternate user security is not active, any user can use any other user ID as an alternative user ID.

You can turn alternate user security checking on or off at either queue manager or queue sharing group level.

Context

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section
The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQPUT or an MQPUT1 call is made. The application might generate the data, the data might be passed on from another message, or the queue manager might generate the data by default. For example, server programs can use context data to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternative user.

You use context security to control whether the user can specify any of the context options on any MQOPEN or MQPUT call. For information about the context options, see the MQOPEN options relating to message context. For descriptions of the message descriptor fields relating to context, see MQMD - Message descriptorMQMD - Message descriptor.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

You can turn context security checking on or off at either queue, queue manager or queue sharing group level.

z/OS Availability on z/OS

IBM MQ for z/OS has many features for high availability. This topic describes some of the considerations for availability.

Several features of IBM MQ can increase system availability if the queue manager or channel initiator fails. For more information about these features, see the following sections:

- Sysplex considerations
- Shared queues
- Shared channels
- IBM MQ network availability
- Using the z/OS Automatic Restart Manager (ARM)
- Using the z/OS Extended Recovery Facility (XRF)
- Using the z/OS GROUPUR attribute for recovery in a queue sharing group
- Where to find more information about availability

Sysplex considerations

In a *sysplex*, a number of z/OS operating system images collaborate in a single system image and communicate using a coupling facility. IBM MQ can use the facilities of the sysplex environment for enhanced availability.

Removing the affinities between a queue manager and a particular z/OS image allows a queue manager to be restarted on a different z/OS image in the event of an image failure. The restart mechanism can be manual, use ARM, or use system automation, if you ensure the following:

- All page sets, logs, bootstrap data sets, code libraries, and queue manager configuration data sets are defined on shared volumes.
- The subsystem definition has sysplex scope and a unique name within the sysplex.
- The level of *early code* installed on every z/OS image at IPL time is at the same level.
- TCP virtual IP addresses (VIPA) is available on each TCP stack in the sysplex, and you have configured IBM MQ TCP listeners and inbound connections to use VIPAs rather than default host names.

For more information about using TCP in a sysplex, see *TCP/IP in a sysplex*, SG24-5235, an IBM Redbooks® publication.

You can additionally configure multiple queue managers running on different operating system images in a sysplex to operate as a queue sharing group, which can take advantage of shared queues and shared channels for higher availability and workload balancing.

Shared queues

In the queue sharing group environment, an application can connect to any of the queue managers within the queue sharing group. Because all the queue managers in the queue sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue sharing group can continue processing the queue. For information about high availability of shared queues, see “Advantages of using shared queues” on page 196.

To further enhance the availability of messages in a queue sharing group, IBM MQ detects if another queue manager in the group disconnects from the coupling facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in “Peer recovery” on page 269.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, Db2, and IBM MQ for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in "Using the z/OS Automatic Restart Manager (ARM)" on page 283.

Shared channels

In the queue sharing group environment, IBM MQ provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). IBM MQ uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue sharing group. This is described in "Shared channels" on page 216.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in Shared outbound channels.

IBM MQ network availability

IBM MQ messages are carried from queue manager to queue manager in an IBM MQ network using channels. You can change the configuration at a number of levels to improve the network availability of a queue manager, and the ability of an IBM MQ channel to detect a network problem and to reconnect.

TCP *Keepalive* is available for TCP/IP channels. It causes TCP to send packets periodically between sessions to detect network failures. The KAINT channel attribute determines the frequency of these packets for a channel.

AdoptMCA allows a channel, blocked in receive processing as a result of a network outage, to be terminated and replaced by a new connection request. You control AdoptMCA using the ADOPTMCA queue manager property with the MQSC utility or the AdoptNewMCAType property with the Programmable Command Formats interface.

ReceiveTimeout prevents a channel from being permanently blocked in a network receive call. The RCVTIME and RCVTMIN channel initiator parameters, determine the receive timeout characteristics for channels, as a function of their heartbeat interval. See Queue manager parameter for more details.

Using the z/OS Automatic Restart Manager (ARM)

You can use IBM MQ for z/OS in conjunction with the z/OS automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM restarts it on the same z/OS image. If z/OS fails, a whole group of related subsystems and applications also fail. ARM can restart all the failed systems automatically, in a predefined order, on another z/OS image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue sharing groups.

You can use ARM to restart a queue manager on a different z/OS image within the sysplex in the event of z/OS failure.

To enable automatic restart, you must do the following:

1. Set up an ARM coupling data set.
2. Define the automatic restart actions that you want z/OS to perform in an *ARM policy*.

3. Start the ARM policy.

If you want to restart queue managers in different z/OS images automatically, every queue manager in each z/OS image on which that queue manager might be restarted must be defined with a sysplex-wide unique 4-character subsystem name.

Using ARM with IBM MQ is described in Using ARM in an IBM MQ network.

Using the z/OS Extended Recovery Facility (XRF)

You can use IBM MQ in an extended recovery facility (XRF) environment. All IBM MQ-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternative XRF processors.

If you use XRF for recovery, you must stop the queue manager on the active processor and start it on the alternative processor. For CICS, you can do this using the command list table (CLT) provided by CICS, or the system operator can do it manually. For IMS, this is a manual operation and you must do it after the coordinating IMS system has completed the processor switch.

IBM MQ utilities must be completed or terminated before the queue manager can be switched to the alternative processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent the queue manager starting on the alternative processor before the queue manager on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of IBM MQ on the two systems. You must include the BSDS as a protected resource, and you must include the active and alternative XRF processors in the GRS ring.

Using the z/OS GROUPUR attribute for recovery in a queue sharing group

Queue sharing groups (QSG) allow additional transactional facilities which are described in this topic. The GROUPUR attribute allows XA client applications to have any in-doubt transaction recovery that may be required, performed on any member of the QSG.

If an XA client application connects to a queue sharing group (QSG) through a Sysplex it cannot guarantee which specific queue manager it connects to. Use of the GROUPUR attribute by queue managers within the queue sharing group can enable any in-doubt transaction recovery that may be necessary to occur on any member of the QSG. Even if the queue manager to which the application was initially connected is not available, transaction recovery can take place.

This feature frees the XA client application from any dependency on specific members of the QSG and thus extends the availability of the queue manager. The queue sharing group appears to the transactional application as a single entity providing all the IBM MQ features and without a single queue manager point of failure.

This functionality is not apparent to the transactional application.

Where to find more information about availability

You can find more information about these topics from the following sources:

Table 24. Where to find more information about availability

Topic	**Where to look**
Queue sharing groups	“Shared queues and queue sharing groups” on page 177
System parameters	Configuring system parameters

Table 24. Where to find more information about availability (continued)	
Topic	**Where to look**
Using the Automatic Restart Manager Utility programs	Using ARM in an IBM MQ network
MQSC commands	MQSC commands

z/OS Monitoring and statistics on IBM MQ for z/OS

IBM MQ for z/OS has a set of facilities for monitoring the queue manager, and gathering statistics.

IBM MQ supplies facilities for monitoring the system and collecting statistics. For further information about these facilities, see the following sections:

- “Online monitoring” on page 285
- “IBM MQ trace” on page 285
- “Events” on page 286

Online monitoring

IBM MQ includes the following commands for monitoring the status of IBM MQ objects:

- DISPLAY CHSTATUS displays the status of a specified channel.
- DISPLAY QSTATUS displays the status of a specified queue.
- DISPLAY CONN displays the status of a specified connection.

For more information about these commands, see The MQSC commands.

IBM MQ trace

IBM MQ supplies a trace facility that you can use to gather the following information while the queue manager is running:

Performance statistics

The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about Db2 usage (Db2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about SMDS usage (shared message data set statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the processor time spent processing MQI calls and about the number of MQPUT and MQGET requests made by a particular user.

- IBM MQ can also gather information about each task using IBM MQ. This data is gathered as a thread-level accounting record. For each thread, IBM MQ also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

IBM MQ events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple IBM MQ applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView®, to monitor reports and create the appropriate alerts.

Related tasks

Using IBM MQ trace

Using IBM MQ events

z/OS Unit of recovery disposition on z/OS

Certain transactional applications can use a GROUP, rather than a QMGR, unit of recovery disposition when connected to a queue manager in a queue sharing group (QSG) by specifying the QSG name when they connect instead of the queue manager name. This allows transaction recovery to be more flexible and robust by removing the requirement to reconnect to the same queue manager in the QSG.

Transactions started by applications that have connected using the queue sharing group name also have a GROUP unit of recovery disposition.

When a transactional application connects with a GROUP unit of recovery disposition it is logically connected to the queue sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it has started that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which may be unavailable at that time.

Applications that connect with a QMGR unit of recovery disposition have a direct affinity to the queue manager to which they are connected. In a recovery scenario the transaction coordinator must reconnect to the same queue manager to resolve any in-doubt transactions, irrespective of whether the queue manager belongs to a queue sharing group.

When applications specify a queue sharing group name, and thus connect to a queue manager in a QSG with a GROUP unit of recovery disposition, the queue sharing group is logically a separate resource manager. This means that in-doubt transactions are only visible to an application if it reconnects with the same unit of recovery disposition. In-doubt transactions with a QMGR unit of recovery disposition are not visible to applications that have connected with a GROUP unit of recovery disposition and vice versa.

Related concepts

“Enabling GROUP units of recovery” on page 286
A queue sharing group can configure and enable support for GROUP units of recovery.

“Application support” on page 287
Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

z/OS Enabling GROUP units of recovery

A queue sharing group can configure and enable support for GROUP units of recovery.

To use GROUP units of recovery on a queue manager within a QSG, enable the GROUPUR queue manager attribute. For more information about this concept, see “Unit of recovery disposition on z/OS” on page 286 before reading the rest of this topic.

When the GROUPUR queue manager attribute is enabled, the queue manager accepts new connections with a GROUP unit of recovery disposition. If you disable this attribute new connections with this disposition are not accepted, although applications already connected is unaffected until they disconnect.

When an application connects with a GROUP unit of recovery disposition and either inquires what transactions are in doubt or attempts to resolve a transaction that was started elsewhere in the queue sharing group (QSG), the queue manager to which it is now connected must be able to communicate with the other members of the queue sharing group so that it can process the request. To do this it uses a shared queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE. This queue must be on a recoverable application structure called CSQSYSAPPL. The structure must be recoverable because persistent messages are stored on this queue when processing resolution requests.

Before you can enable GROUP units of recovery, you must ensure that the coupling facility structure and the shared queue are defined. You can use the definitions in the CSQ4INSS sample. When the queue is defined, or detected during startup, each queue manager in the queue sharing group opens the queue so that it can receive incoming requests. If you want to delete or move the queue because it has been defined incorrectly you can request that the queue managers close their open handles on it by updating the queue object to inhibit MQGET requests. When you have made the necessary corrections, permitting applications to get messages from the queue once more directs each queue manager to reopen it. Use the DISPLAY QSTATUS command to identify what handles are open on a queue.

When you have completed this setup you can then enable GROUP units of recovery on each queue manager that you want transactional applications to be able to connect to with a GROUP unit of recovery disposition. This need not be all of the queue managers within the queue sharing group but if you choose to only enable this functionality on a subset of the queue sharing group you must ensure that your applications only attempt to connect to queue managers on which you have enabled it. For more information, see “Application support” on page 287.

When you attempt to enable the GROUPUR queue manager attribute, a number of configuration checks are performed. The queue manager checks that:

- It belongs to a queue sharing group.
- The shared-queue called SYSTEM.QSG.UR.RESOLUTION.QUEUE has been defined, according to the the definition in CSQ4INSS.
- The SYSTEM.QSG.UR.RESOLUTION.QUEUE is on a recoverable CF structure called CSQSYSAPPL.

If any of the above checks fail, the GROUPUR attribute remains disabled and a message code is returned.

These configuration checks are also performed at queue manager startup if the queue manager attribute is enabled. If any of the checks fail during startup GROUP units of recovery is disabled and the queue manager issues a message identifying which check failed. When you have performed the necessary corrective action you must then re-enable the queue manager attribute.

z/OS Application support

Use this page to determine which applications can connect with a GROUP unit of recovery disposition.

Support for the GROUP unit of recovery disposition is limited to certain types of transactional applications for which IBM MQ for z/OS is a resource manager but not the transaction coordinator. Currently supported transactional applications are:

- IBM MQ extended transactional client applications
- IBM MQ classes for JMS applications running in an application server, such as WebSphere Application Server.
- CICS applications running in CICS Transaction Server 4.2 or later, when the CICS MQCONN resource definition is configured with RESYNCMEMBER(GROUPRESYNC).

Related concepts

“ IBM MQ extended transactional client applications” on page 288
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 288
Use this page to determine how CICS can use the GROUP unit of recovery disposition.

z/OS *IBM MQ extended transactional client applications*

Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

An example of an IBM MQ extended transactional client application is one that uses JMS and runs in WebSphere Application Server, connecting to IBM MQ over TCP/IP, rather than local bindings. These client applications connect to IBM MQ for z/OS over network connections, such as via TCP/IP. For these applications, it is the value specified for the QMNAME parameter of the xa_info string passed in the xa_open call that specifies whether a QMGR or GROUP unit of recovery disposition is used. For more information about xa_open, see The format of an xa_open string and Additional error processing for xa_open. For JMS applications this is done by specifying the name of the queue sharing group (QSG) in the ConnectionFactory instead of the name of a specific queue manager.

For XA client applications to take advantage of using the GROUP unit of recovery disposition you must configure your TCP/IP setup to allow your client applications to be routed to the queue managers in the queue sharing group that have the GROUPUR attribute enabled, rather than a specific queue manager. One of the dynamic virtual IP address technologies that you can use to do this is the z/OS SysPlex Distributor. See z/OS Communications Server and Compétences de base z/OS : adressage virtuel dynamique for more details. If you want to enable GROUP units of recovery on a subset of the queue managers in your queue sharing group, ensure that your client applications cannot be routed to those on which it is not enabled.

Your client applications do not have to connect to the queue sharing group using shared channels.

z/OS *CICS applications*

Use this page to determine how CICS can use the GROUP unit of recovery disposition.

CICS 4.2 and later provides the group resynchronization option, RESYNCMEMBER(GROUPRESYNC) in an MQCONN resource definition. A CICS configured with this option can connect to any suitable queue manager in a queue sharing group which is running on the same LPAR as that CICS region. To support the CICS GROUPRESYNC option, a queue manager must be running at MQ V7.1 or later, and be enabled for GROUPUR support.

Transactions running within a CICS region connected to MQ using GROUPRESYNC create units of work with GROUP unit of recovery disposition.

You can use RESYNCMEMBER(GROUPRESYNC) to enable faster recovery after a queue manager failure as it enables the CICS region to immediately connect to an alternative eligible queue manager running on the same LPAR, resolving any indoubt transactions as necessary, without waiting for queue manager restart.

RESYNCMEMBER(GROUPRESYNC) also enables more flexible restart options for CICS. A CICS region with its MQ connection configured to use GROUPRESYNC and MQ shared queues can be restarted on any LPAR where there is a queue manager running as a member of the same queue sharing group.

z/OS IBM MQ and other z/OS products

Use this topic to understand how IBM MQ can work with other z/OS products.

Related concepts

“IBM MQ and CICS” on page 289
All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

“IBM MQ for z/OS and WebSphere Application Server” on page 295

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Related reference

"IBM MQ and IMS" on page 290
Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

"IBM MQ and the z/OS Batch, TSO, and RRS adapters" on page 294
Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

z/OS IBM MQ and CICS

All the CICS versions supported by IBM MQ 9.0.0, and later, use the CICS supplied version of the adapter and bridge.

For more information about configuring the IBM MQ CICS adapter, and the IBM MQ CICS bridge components, see the Configuring connections to IBM MQ section of the CICS documentation.

Related tasks

Using IBM MQ with CICS

z/OS CICS group attach

CICS group attach provides the ability for a CICS region to connect to any active member of an IBM MQ queue sharing group on the same LPAR rather than specifying an individual queue manager. CICS still connects to a single queue manager at a time.

You require at least two queue managers on the LPAR to support CICS group attach. Using group attach provides higher availability as you do not need a particular queue manager to be active. CICS connects to any queue manager in the queue sharing group on the LPAR.

For more information, see the CICS documentation on the MQCONN resource.

CICS attempts to connect to MQNAME passed as if it were a queue manager:

- If the queue manager exists and is active, the connection will work.
- If the connection fails, CICS queries the status of queue managers in the group to ascertain which are active on same LPAR.
- If multiple queue managers are active, CICS checks for RESYNCMEMBER(YES) and the UOW status to determine whether CICS needs to connect, or should connect, to a particular member, or wait if not active.
- If there is no need to connect to a particular member, CICS selects a queue manager (using a randomizing algorithm).
- CICS attempts to connect to chosen queue manager.
- If the attempt fails then, depending upon the return code, CICS chooses the next member, then goes through the selection loop again.
- If no queue managers are active, CICS issues multiple connections to the list of queue managers and waits on ECBLIST until the first queue manager becomes available.

Related concepts

"Group units of recovery (GROUPUR) for CICS" on page 290
The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

Related information

Support for IBM MQ queue sharing groups

z/OS Group units of recovery (GROUPUR) for CICS

The IBM MQ GROUPUR for CICS provides peer recovery for in-doubt units of work in a queue sharing group (QSG). One IBM MQ queue manager can resolve in-doubt units of work on behalf of another queue manager in the queue sharing group. This means that if CICS reconnects through group attach to a different queue manager in the QSG, it can resolve indoubt transactions from a previous IBM MQ connection.

If a CICS region is working with a queue manager, and the queue manager ends abnormally, then any indoubt transactions are recovered. This eliminates the need for the CICS region to wait for the queue manager that it was working with to restart, and then resolve any in doubt units of work. This means that you need at least two queue managers on the LPAR, so that CICS can connect to another queue manager in the event of an abnormal termination of the first queue manager.

The new RESYNCMEMBER(GROUPRESYNC) setting on the CICS MQCONN definition:

- Uses the IBM MQ group attach function and peer recovery.
- Requires a queue manager with the GROUPUR attribute enabled.
- Still supports the existing CICS MQCONN RESYNCMEMBER settings (YES and NO):
  - Uses the existing CICS group attach function and no peer recovery.
  - Changing RESYNCMEMBER settings takes effect next time CICS connects to IBM MQ.

Related concepts

"Enabling GROUP units of recovery" on page 286

A queue sharing group can configure and enable support for GROUP units of recovery.

z/OS IBM MQ and IMS

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

The optional additional IBM MQ - IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with IBM MQ, without the need to rewrite them.

For more information about these components, see the following subtopics:

Related concepts

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

Related tasks

Setting up the IMS adapter

Setting up the IMS bridge

Operating the IMS adapter

Related reference

MQIIH - IMS information header

z/OS The IMS adapter

The IMS adapter is an interface between IMS application programs and an IBM MQ subsystem.

The IBM MQ adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an IBM MQ subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to IBM MQ using the External Subsystem Attach Facility (ESAF) provided by IMS. Usually, IMS connects to IBM MQ automatically without operator intervention.

The IMS adapter provides access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to IBM MQ.

The adapter supports a two-phase commit protocol for changes made to resources owned by IBM MQ with IMS acting as the syncpoint coordinator. Conversations where IMS is not the syncpoint coordinator, for example APPC- protected (SYNCLVL=SYNCPT) conversations, are not supported by the IMS adapter.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 291.

You can use IBM MQ with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error.

Note: As of IMS 15.2 Extended Recovery Facility (XRF) is no longer supported. See the IMS documentation for more information.

Using the adapter

The application programs and the IMS adapter run in the same address space. The queue manager is separate, in its own address space.

You must link-edit each program that issues one or more MQI calls to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the IBM MQ-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to IBM MQ. However, the IMS terminal operator has no control over the IBM MQ address space. For example, the operator cannot shut down IBM MQ from an IMS address space.

Restrictions

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

- MQCB
- MQCB_FUNCTION
- MQCTL

The IMS trigger monitor

The IMS trigger monitor (**CSQQTRMN**) is an IBM MQ-supplied IMS application that starts an IMS transaction when an IBM MQ event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, you can start an instance of CSQQTRMN to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the

message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. When it has started, the trigger monitor runs until IBM MQ or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions that are started by the trigger monitor contain the following:

- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is protected by Security Server (previously known as RACF), you might need to define CSQQTRMN as a user ID to Security Server.

z/OS The IBM MQ - IMS bridge

The IBM MQ - IMS bridge is the component of IBM MQ for z/OS that allows direct access from IBM MQ applications to applications on your IMS system.

The IBM MQ - IMS bridge enables *implicit MQI support*. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by IBM MQ messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no IBM MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. IBM MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one IBM MQ message.

The IMS bridge is illustrated in Figure 78 on page 292.

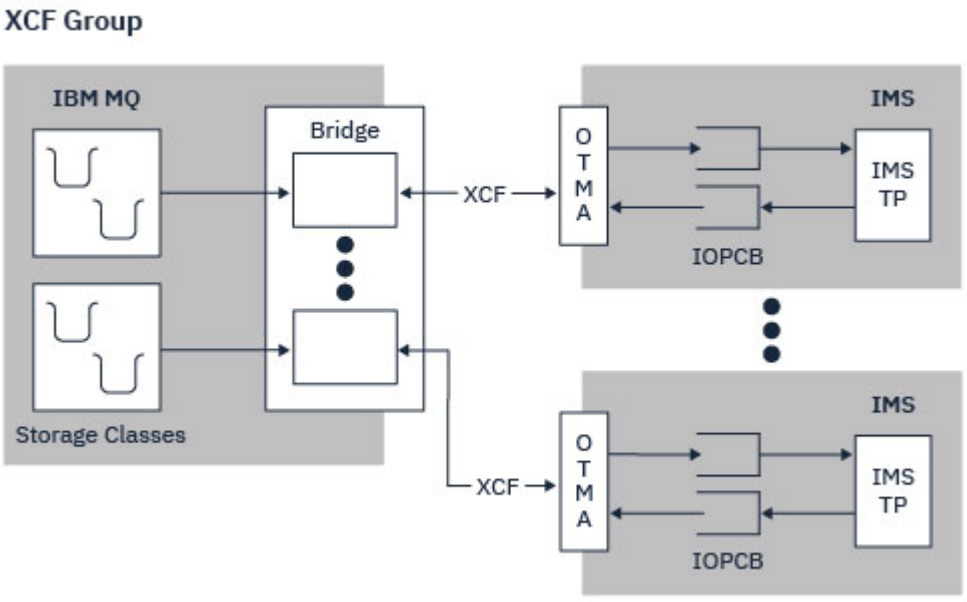


Figure 78. The IBM MQ - IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

See Setting up the IMS bridge for information on setting up an IMS bridge and adding an additional IMS connection to the same queue manager.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS. It functions as an interface for host-based communications servers accessing IMS TM applications through the z/OS Cross Systems coupling facility (XCF).

OTMA enables clients to connect to IMS to provide high performance for interactions between clients and IMS for a large network or large number of sessions. OTMA is implemented in a z/OS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA Resource Monitoring

Support for the x'3C' OTMA protocol messages, available in IMS v10 or higher, is included in the IBM MQ - IMS bridge in IBM MQ for z/OS. These messages are sent to OTMA clients by IMS to report its health status.

If an IMS partner is unable to process the volume of transaction requests being sent then it will notify IBM MQ that a flood warning has occurred. In response IBM MQ will slow down the rate at which requests are sent over the bridge.

If IMS is still unable to process the transaction requests and a full flood condition occurs all TPIPEs to the IMS partner are suspended. Upon notification from the IMS partner that the flood or flood-warning condition has been relieved IBM MQ will resume all suspended TPIPEs, if appropriate, and gradually increase the rate at which transaction requests are sent until the maximum rate is achieved. Console messages are issued by IBM MQ in response to a change in the status of IMS partners.

If IMS v10 partners are being used you should ensure that PTF UK45082 has been applied.

Submitting IMS transactions from IBM MQ

To submit an IMS transaction that uses the bridge, applications put messages on an IBM MQ queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the IBM MQ - IMS bridge to make assumptions about the data in the message.

IBM MQ then puts the message to an IMS queue (it is queued in IBM MQ first to enable the use of syncpoints to assure data integrity). The storage class of the IBM MQ queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the IBM MQ - IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on IBM MQ for z/OS.

Data returned from the IMS system is written directly to the IBM MQ reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the **ReplyToQMgr** field of the MQMD.)

Related concepts

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

Related tasks

Customizing the IMS bridge

Related reference

“IBM MQ and IMS” on page 290

Use this topic to understand how IBM MQ works with IMS. The IMS adapter allows you to connect your queue manager to IMS, and enables IMS applications to use the MQI.

z/OS IBM MQ and the z/OS Batch, TSO, and RRS adapters

Use this topic to understand how IBM MQ works with the z/OS Batch, TSO, and RRS adapters.

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between IBM MQ and z/OS application programs running under JES, TSO, or z/OS UNIX System Services. These adaptors enable z/OS application programs to use the MQI.

The adapters provide access to IBM MQ resources for programs running in the following modes or states:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

Connections between application programs and IBM MQ are at the task level. The adapters provide a connection thread from an application task control block (TCB) to IBM MQ.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by IBM MQ. It does not support multi-phase commit protocols. The RRS adapter enables IBM MQ applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by z/OS Resource Recovery Services (RRS).

The adapters use the z/OS STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the IBM MQ termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in IBM MQ (for example, a signal or a wait).

The Batch/TSO adapter

The IBM MQ Batch/TSO adapter provides IBM MQ support for z/OS Batch and TSO applications. All application programs that run under z/OS Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of z/OS that provides a system-wide service for coordinating two-phase commit across z/OS products. The IBM MQ Batch/TSO RRS adapter (the RRS adapter) provides IBM MQ support for z/OS Batch and TSO applications that want to use these services. The RRS adapter enables IBM MQ to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, Db2).

The RRS adapter provides two stubs; you must link-edit application programs that want to use RRS with one of these stubs.

CSQBRSTB
: This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

  You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code MQRC_ENVIRONMENT_ERROR.

CSQBRRSI
This stub allows you to use MQI calls **MQCMIT** and **MQBACK** ; IBM MQ actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see The RRS batch adapter.

Where to find more information about the z/OS Batch, TSO, and RRS adapters

You can find more information about the topics in this section in the following sources:

Table 25. Where to find more information about using z/OS Batch with IBM MQ	
Topic	**Where to look**
Setting up the Batch adapters	Task 19: Set up Batch, TSO, and RRS adapters
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

z/OS IBM MQ for z/OS and WebSphere Application Server

Use this topic to understand the use of IBM MQ for z/OS by the WebSphere Application Server.

Applications written in Java that are running under WebSphere Application Server can use the Java Message Service (JMS) specification to perform messaging. Point-to-point messaging in this environment can be provided by an IBM MQ for z/OS queue manager.

A benefit of using an IBM MQ for z/OS queue manager to provide the messaging is that connecting JMS applications can participate fully in the functionality of an IBM MQ network. For example, they can use the IMS bridge, or exchange messages with queue managers running on other platforms.

Connection between WebSphere Application Server and a queue manager

See Using IBM MQ and WebSphere Application Server together for more information.

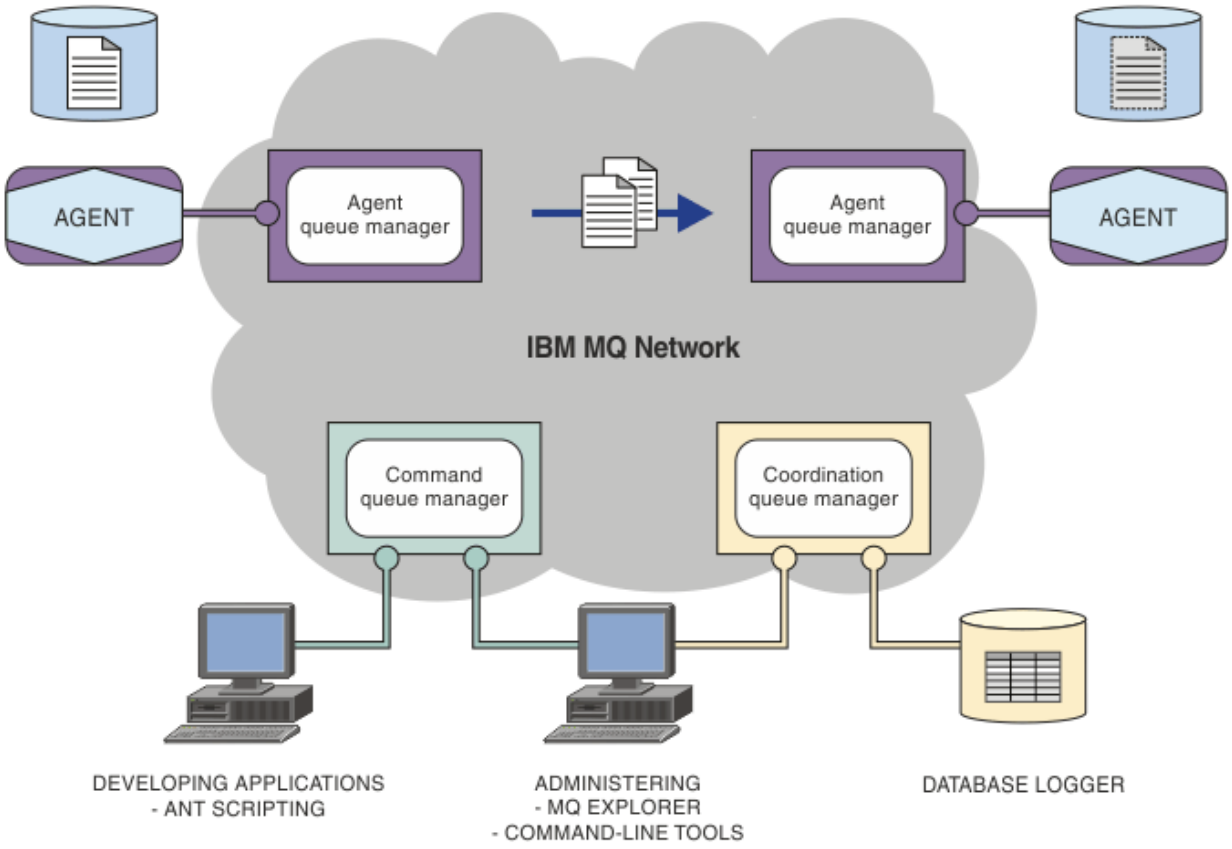
Using IBM MQ functions from JMS applications

By default, JMS messages held on IBM MQ queues use an MQRFH2 header to hold some of the JMS message header information. Many legacy IBM MQ applications cannot process messages with these headers, and require their own characteristic headers, for example the MQCIH for CICS Bridge, or MQWIH for IBM MQ Workflow applications. For more details about these special considerations, see Mapping JMS messages onto IBM MQ messages.

Managed File Transfer

Managed File Transfer transfère des fichiers entre des systèmes d'une manière gérée et auditable, quelle que soit la taille des fichiers ou des systèmes d'exploitation utilisés.

Vous pouvez utiliser Managed File Transfer pour développer une solution automatisée, évolutive et personnalisée qui vous permette de gérer, d'approuver et de sécuriser les transferts de fichiers. Managed File Transfer élimine les redondances coûteuses, diminue les coûts de maintenance et maximise vos investissements informatiques existants.



Le diagramme illustre une topologie Managed File Transfer simple. Il existe deux agents, chacun se connectant à son propre gestionnaire de file d'attente d'agent sur un réseau IBM MQ. Un fichier est transféré de l'agent d'un côté du diagramme, par l'intermédiaire du réseau IBM MQ, vers l'agent de l'autre côté du diagramme. En outre, le réseau IBM MQ comporte le gestionnaire de file d'attente de coordination et un gestionnaire de file d'attente de commandes. Les applications et les outils se connectant à ces gestionnaires de files d'attente pour configurer, administrer, exploiter et consigner l'activité Managed File Transfer dans le réseau IBM MQ.

Managed File Transfer peut être installé avec quatre options différentes, en fonction de votre système d'exploitation et de la configuration générale. Ces options sont Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Service et Managed File Transfer Tools. Pour plus d'informations, voir Options du produit Managed File Transfer.

Vous pouvez utiliser Managed File Transfer pour effectuer les tâches suivantes :

- Créer des transferts de fichiers gérés
  - Windows Linux Créer des transferts de fichiers à partir d'IBM MQ Explorer sur des plateformes Linux ou Windows.
  - Créer des transferts de fichiers à partir de la ligne de commande sur toutes les plateformes prises en charge.
  - Intégrer la fonction de transfert de fichiers dans l'outil Apache Ant.
  - Ecrire des applications qui contrôlent Managed File Transfer en plaçant les messages sur les files d'attente des commandes d'agent.
  - Planifier les transferts de fichiers à un moment ultérieur. Vous pouvez également déclencher des transferts de fichiers planifiés en fonction d'une plage d'événements du système de fichiers (par exemple, création d'un fichier).

  - Surveiller en permanence une ressource, telle qu'un répertoire, et, lorsque le contenu de cette ressource satisfait une condition prédéfinie, lancer une tâche. Il peut s'agir d'un transfert de fichiers, d'un script Ant ou d'un travail JCL.
  - Transférer des fichiers vers et depuis des files d'attente IBM MQ.
  - Transférer des fichiers vers et depuis des serveurs FTP, FTPS ou SFTP.
  - Transfert de fichiers vers et depuis des noeuds Connect:Direct.
  - Transférer des fichiers texte et binaires. Les fichiers texte sont convertis automatiquement entre les pages de code et les conventions de fin de ligne des systèmes source et cible.
  - Les transferts peuvent être sécurisés, à l'aide des normes de l'industrie pour les connexions SSL (Secure Socket Layer).
- Afficher les transferts en cours et les informations de journal sur tous les transferts sur votre réseau.
  - Windows Linux Afficher le statut des transferts en cours d'IBM MQ Explorer sur des plateformes Linux ou Windows.
  - Windows Linux Vérifier le statut des transferts terminés à l'aide d'IBM MQ Explorer sur des plateformes Linux ou Windows.
  - Utiliser la fonction du consignateur de base de données Managed File Transfer pour sauvegarder les messages de journal dans une base de données Db2 ou Oracle.

Managed File Transfer est développé sur IBM MQ, qui permet une distribution ponctuelle et garantie des messages entre les applications. Vous pouvez profiter des diverses fonctions d'IBM MQ. Par exemple, vous pouvez utiliser la compression des canaux pour compresser les données que vous envoyez entre les agents via les canaux IBM MQ et utiliser les canaux SSL pour sécuriser les données que vous envoyez entre les agents. Les fichiers sont transférés de manière fiable et peuvent tolérer un incident de l'infrastructure par l'intermédiaire de laquelle le transfert de fichiers est effectué. En cas de panne réseau, le transfert de fichiers redémarre à partir de là où il s'était arrêté lorsque la connectivité est restaurée.

En consolidant le transfert de fichiers avec votre réseau IBM MQ existant, vous pouvez éviter d'utiliser les ressources requises pour gérer deux infrastructures distinctes. Si vous n'êtes pas déjà un client d'IBM MQ, en créant un réseau IBM MQ pour prendre en charge Managed File Transfer, vous développez le réseau principal d'une future implémentation d'architecture SOA. Si vous êtes déjà un client d'IBM MQ, Managed File Transfer peut profiter de votre infrastructure IBM MQ et notamment d'IBM MQ Internet Pass-Thru et d'IBM Integration Bus.

Vous pouvez tirer parti des solutions à haute disponibilité IBM MQ pour améliorer la résilience de votre configuration Managed File Transfer . Si vos agents utilisent des gestionnaires de files d'attente de données répliquées (RDQM), vous devez les configurer pour qu'ils utilisent la fonction d'adresse IP flottante. Cela signifie que les agents utilisent la même adresse IP pour communiquer avec l'une des trois instances de gestionnaire de files d'attente de données répliquées en cours d'exécution et se reconnectent automatiquement lors de la reprise en ligne (voir Haute disponibilité des gestionnaires de files d'attente de données répliquées et Création et suppression d'une adresse IP flottante). Si vous utilisez la solution de gestionnaire de files d'attente multi-instance, les applications utilisent une adresse IP différente pour communiquer avec chaque instance, qui est gérée par la reconnexion client lors de la reprise en ligne (voir Multi-instance queue managers et Channel and client reconnection).

Managed File Transfer s'intègre à un certain nombre d'autres produits IBM :

IBM Integration Bus

Fichiers de processus transférés par Managed File Transfer dans le cadre d'un flux IBM Integration Bus. Pour plus d'informations, voir Working with MFT from IBM Integration Bus.

IBM Sterling Connect:Direct

Transférez des fichiers vers et depuis un réseau Connect:Direct existant à l'aide d'un pont Managed File Transfer Connect:Direct. Pour plus d'informations, voir The Connect:Direct bridge.

IBM Tivoli Composite Application Manager

IBM Tivoli Composite Application Manager fournit un agent que vous pouvez utiliser pour surveiller les informations publiées sur le gestionnaire de file d'attente de coordination.

Concepts associés

Options du produit Managed File Transfer

«Présentation de la topologie MFT», à la page 298
Présentation de la manière dont les agents Managed File Transfer sont connectés au gestionnaire de file d'attente de coordination dans un réseau IBM MQ.

«Fonctionnement de MFT avec IBM MQ», à la page 298
Managed File Transfer interagit de diverses façons avec IBM MQ.

Fonctionnement de MFT avec IBM MQ

Managed File Transfer interagit de diverses façons avec IBM MQ.

- Managed File Transfer transfère des fichiers entre des processus d'agent en divisant chaque fichier en un ou plusieurs messages et en transmettant les messages sur votre réseau IBM MQ.
- Les processus d'agent déplacent les données de fichier à l'aide de messages non persistants afin de limiter l'impact sur vos journaux IBM MQ. En communiquant les uns avec les autres, les processus d'agent régulent le flux de messages contenant des données de fichier. Ainsi, les messages contenant des données de fichier ne s'accumulent pas dans les files d'attente de transmission IBM MQ et au cas où un message non persistant ne serait pas distribué, les données de fichier sont envoyées à nouveau.
- Les agents Managed File Transfer utilisent plusieurs files d'attente IBM MQ. Pour plus d'informations, voir MFT system queues and the system topic.
- Bien que certaines de ces files d'attente soient strictement réservées à un usage interne, un agent peut accepter des demandes sous forme de messages de commande formatés spécialement, qui sont envoyés dans une file d'attente spécifique lue par l'agent. Les commandes de ligne de commande et le plug-in IBM MQ Explorer envoient des messages IBM MQ à l'agent pour l'inviter à effectuer l'action voulue. Vous pouvez écrire des applications IBM MQ qui interagissent avec l'agent de cette façon. Pour plus d'informations, voir Controlling MFT by putting messages on the agent command queue.
- Les agents Managed File Transfer envoient des informations sur leur état et la progression et le résultat des transferts vers un gestionnaire de files d'attente MQ qui a été désigné comme gestionnaire de file d'attente de coordination. Ces informations sont publiées par le gestionnaire de file d'attente de coordination et les applications souhaitant surveiller la progression des transferts ou conserver des enregistrements des transferts effectués peuvent s'y abonner. Les commandes de ligne de commande et le plug-in IBM MQ Explorer peuvent utiliser les informations qui sont publiées. Vous pouvez écrire des applications IBM MQ qui utilisent ces informations. Pour plus d'informations sur la rubrique dans laquelle les informations sont publiées, voir SYSTEM.FTE rubrique.
- Les composants principaux de Managed File Transfer bénéficient de la capacité des gestionnaires de files d'attente IBM MQ à stocker et retransmettre des messages. En d'autres termes, en cas d'indisponibilité, les parties non affectées de votre infrastructure peuvent continuer de transférer des fichiers. Cette possibilité s'étend au gestionnaire de file d'attente de coordination, où une combinaison de stockage et de retransmission et d'abonnements durables permet au gestionnaire de file d'attente de coordination d'être indisponible sans que des informations essentielles sur les transferts de fichiers qui ont eu lieu ne soient perdues.

Présentation de la topologie MFT

Présentation de la manière dont les agents Managed File Transfer sont connectés au gestionnaire de file d'attente de coordination dans un réseau IBM MQ.

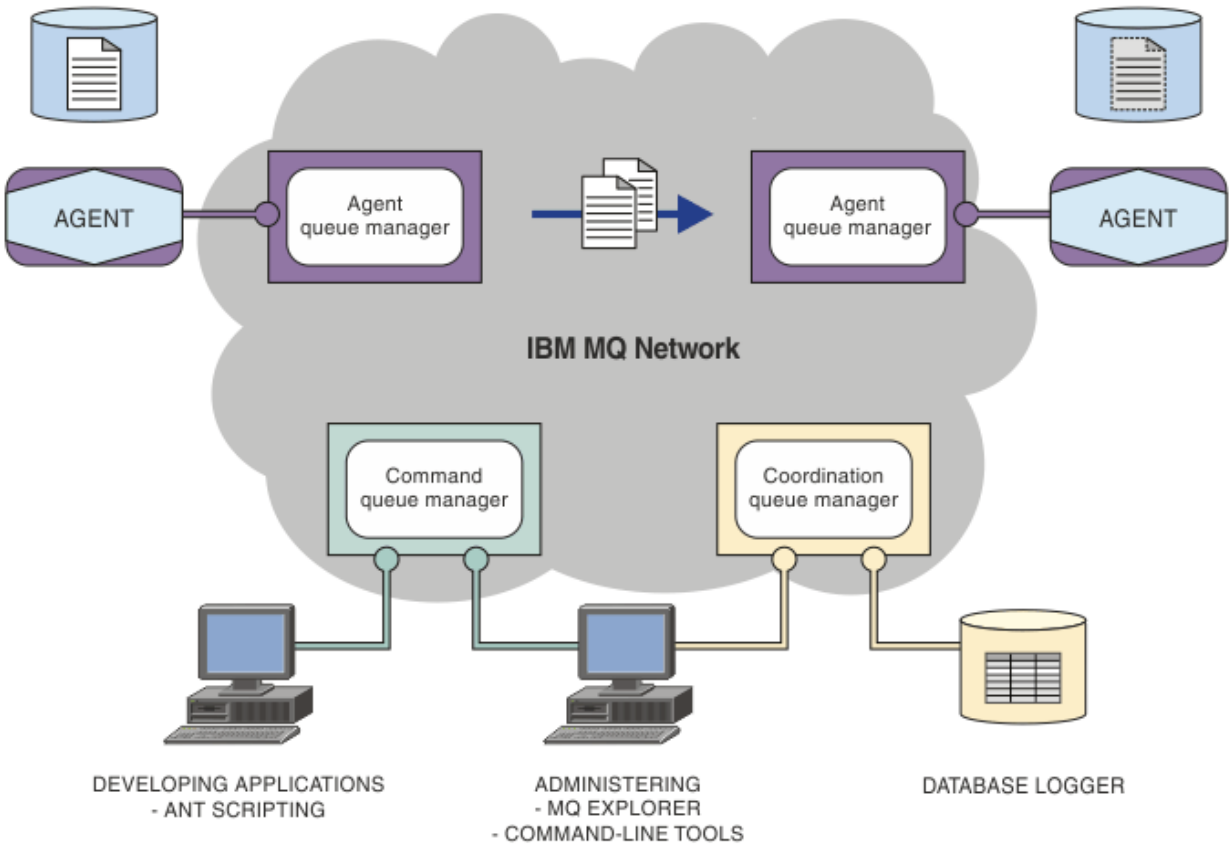
Les agents Managed File Transfer envoient et reçoivent les fichiers transférés. Chaque agent possède son propre ensemble de files d'attente sur son gestionnaire de files d'attente associé et l'agent est connecté à son gestionnaire de files d'attente en mode liaisons ou client. Il peut également utiliser le gestionnaire de file d'attente de coordination comme gestionnaire de files d'attente.

Le gestionnaire de file d'attente de coordination publie les informations d'audit et de transfert de fichier. Il représente un point unique pour la collecte des informations sur l'agent, le statut de transfert et l'audit de transfert. Il n'est pas nécessaire que le gestionnaire de file d'attente de coordination soit disponible pour que les transferts aient lieu. Si le gestionnaire de file d'attente de coordination devient temporairement indisponible, les transferts continuent normalement. Les messages de statut et d'audit sont stockés dans les gestionnaires de files d'attente des agents jusqu'à ce que le gestionnaire de file d'attente de coordination devienne disponible et puisse les traiter normalement.

Les agents s'enregistrent auprès du gestionnaire de file d'attente de coordination et publient les informations détaillées les concernant dans ce dernier. Les informations sur les agents sont utilisées par le plug-in Managed File Transfer pour permettre le démarrage des transferts depuis IBM MQ Explorer. Les informations sur les agents collectées sur le gestionnaire de file d'attente de coordination sont également utilisées par les commandes pour afficher les informations sur les agents et le statut des agents.

Les informations sur le statut de transfert et l'audit de transfert sont publiées sur le gestionnaire de file d'attente de coordination. Les informations sur le statut de transfert et l'audit de transfert sont utilisées par le plug-in Managed File Transfer pour surveiller la progression des transferts à partir d'IBM MQ Explorer. Les informations sur l'audit de transfert stockées dans le gestionnaire de file d'attente de coordination peuvent être conservées à des fins d'audit.

Le gestionnaire de file d'attente de commandes permet de se connecter au réseau IBM MQ ; il s'agit du gestionnaire de files d'attente auquel vous vous connectez lorsque vous exécutez des commandes Managed File Transfer.



Concepts associés

«Managed File Transfer», à la page 295
Managed File Transfer transfère des fichiers entre des systèmes d'une manière gérée et auditable, quelle que soit la taille des fichiers ou des systèmes d'exploitation utilisés.

«Fonctionnement de MFT avec IBM MQ», à la page 298
Managed File Transfer interagit de diverses façons avec IBM MQ.

Scénario pour Managed File Transfer

Présentation de MFT REST API

L'REST API prend en charge certaines commandes Managed File Transfer, notamment le listage des transferts, et des détails sur les agents de transfert de fichier.

L'REST API inclut des options permettant de répertorier tous les transferts Managed File Transfer en cours et d'interroger le statut des agents Managed File Transfer. Pour plus d'informations, voir Initiation à l'REST API MFT.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru (MQIPT) est un composant facultatif d'IBM MQ que vous pouvez utiliser pour implémenter des solutions de messagerie entre des sites distants sur Internet.

Pour obtenir les fichiers d'installation MQIPT pour IBM MQ 9.4.x, accédez à IBM Fix Central pour IBM MQ.

Vous pouvez utiliser MQIPT pour connecter n'importe quelle version prise en charge de IBM MQ. Il n'est pas nécessaire d'installer d'autres composants IBM MQ à la même version que MQIPT.

Si vous avez acheté l'autorisation d'utilisation d'IBM MQ, vous pouvez installer autant de copies de MQIPT que nécessaire. Les installations de MQIPT ne sont pas comptabilisées dans le cadre de votre autorisation d'utilisation d'IBM MQ. Pour plus d'informations sur l'octroi de licence IBM MQ , voir Informations sur les licencesIBM MQ.

Remarque : Cette documentation concerne MQIPT dans IBM MQ 9.4. Pour la documentation du package de prise en charge de MQIPT (version 2.1) dans l'IBM Documentation, voir MQIPT (SupportPac MS81) dans la documentation IBM MQ 9.0.

Remarque : Si vous utilisez MQIPT 2.1 ou une version antérieure, vous êtes encouragé à effectuer une mise à niveau vers MQIPT for IBM MQ 9.4, car la date de fin de prise en charge du module de prise en charge MQIPT était le 30th septembre 2020.

IBM MQ Internet Pass-Thru s'exécute en tant que service autonome pouvant recevoir et transmettre des flux de messages IBM MQ, entre deux gestionnaires de files d'attente IBM MQ ou entre un client IBM MQ et un gestionnaire de files d'attente IBM MQ.

MQIPT active cette connexion lorsque le client et le serveur ne se trouvent pas sur le même réseau physique.

Une ou plusieurs instances de MQIPT peuvent être placées dans le chemin d'accès de communication entre deux gestionnaires de files d'attente IBM MQ, ou entre un client IBM MQ et un gestionnaire de files d'attente IBM MQ. Les instances de MQIPT permettent aux deux systèmes IBM MQ d'échanger des messages sans avoir besoin d'une connexion TCP/IP directe entre les deux systèmes. Cette architecture est utile si la configuration de pare-feu interdit une connexion TCP/IP directe entre les deux systèmes.

MQIPT écoute les connexions entrantes sur un ou plusieurs ports TCP/IP. Ces connexions peuvent contenir des messages IBM MQ normaux, des messages IBM MQ tunnellisés dans HTTP ou des messages chiffrés à l'aide du protocole TLS (Transport Layer Security) ou SSL (Secure Sockets Layer). MQIPT peut gérer plusieurs connexions simultanées.

Le canal IBM MQ qui établit la demande de connexion TCP/IP initiale est désigné comme l'*appelant*. Le canal auquel il tente de se connecter est le *répondeur*, et le gestionnaire de files d'attente qu'il tente de contacter est le *gestionnaire de files d'attente de destination*.

MQIPT conserve des données en mémoire lorsqu'il les transmet de la source vers la destination. Aucune donnée n'est sauvegardée sur le disque (à l'exception de la mémoire que le système d'exploitation paginait sur le disque). La seule fois où MQIPT accède explicitement au disque, c'est pour lire son fichier de configuration et écrire des enregistrements de trace et de journal de connexion.

La gamme complète des types de canal IBM MQ peut se connecter via une ou plusieurs instances de MQIPT. La présence de MQIPT dans un chemin de communication n'a aucun effet sur les caractéristiques

fonctionnelles des composants IBM MQ connectés. Toutefois, les performances du transfert de messages peuvent être affectées.

MQIPT peut être utilisé avec IBM MQ comme décrit dans «Configurations possibles de MQIPT», à la page 304.

Pour installer MQIPT, voir Installation de MQIPT.

Tâches associées

Configuration de IBM MQ Internet Pass-Thru

Administration et configuration d'IBM MQ Internet Pass-Thru

Référence associée

IBM MQ Internet Pass-Thru Référence de configuration

utilisations de MQIPT

Potentiellement, vous pouvez utiliser IBM MQ Internet Pass-Thru (MQIPT) de nombreuses façons.

MQIPT peut être utilisé comme concentrateur de canal

Si vous utilisez MQIPT de cette façon, les canaux en provenance ou à destination de plusieurs hôtes distincts sont perçus par un pare-feu comme s'ils étaient tous originaires ou à destination du même hôte MQIPT. Il est ainsi plus facile de définir et de gérer les règles de filtrage des pare-feux.

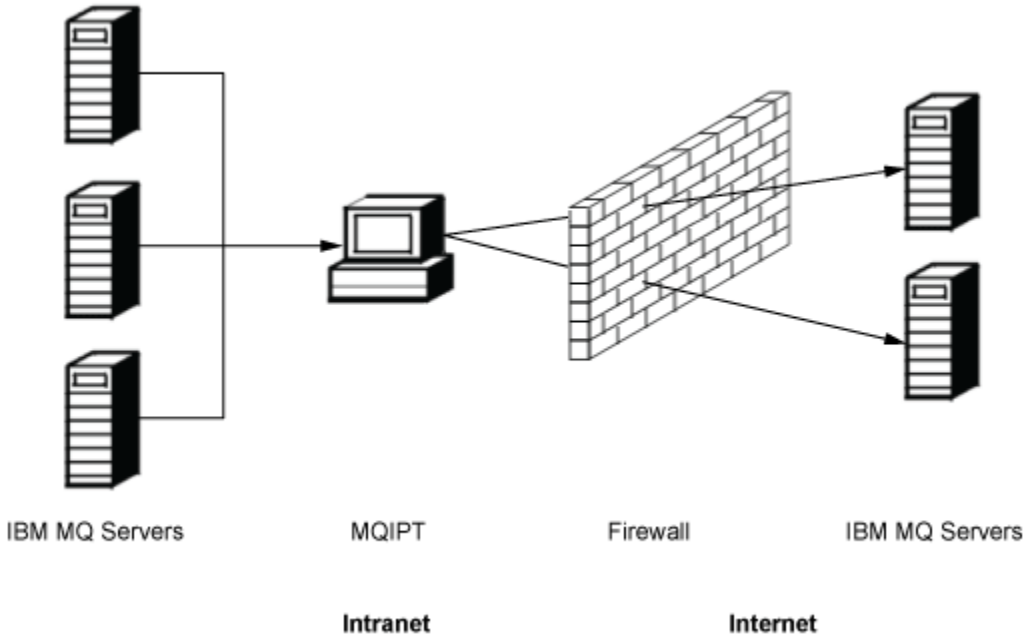


Figure 79. Exemple d'instance MQIPT utilisée comme concentrateur de canal

MQIPT peut être placé dans une zone démilitarisée pour fournir un seul point d'accès

Si MQIPT est placé à l'intérieur d'un pare-feu dans une zone démilitarisée (il s'agit d'une configuration de pare-feu permettant de sécuriser les réseaux locaux), sur un ordinateur utilisant une adresse IP connue et sécurisée, MQIPT peut être utilisé pour écouter les connexions de canal entrantes IBM MQ. Il peut alors les transmettre à l'intranet sécurisé. Le pare-feu interne doit autoriser cet ordinateur sécurisé à établir des connexions entrantes. Dans cette configuration, MQIPT empêche les demandes externes de recevoir les vraies adresses IP des ordinateurs qui se trouvent sur l'intranet sécurisé. MQIPT fournit ainsi

un seul point d'accès. Si nécessaire, MQIPT peut être configuré pour accepter les connexions TLS et transmettre les données à la destination à l'aide d'une connexion TLS distincte, ce qui met fin à la session TLS dans la zone démilitarisée.

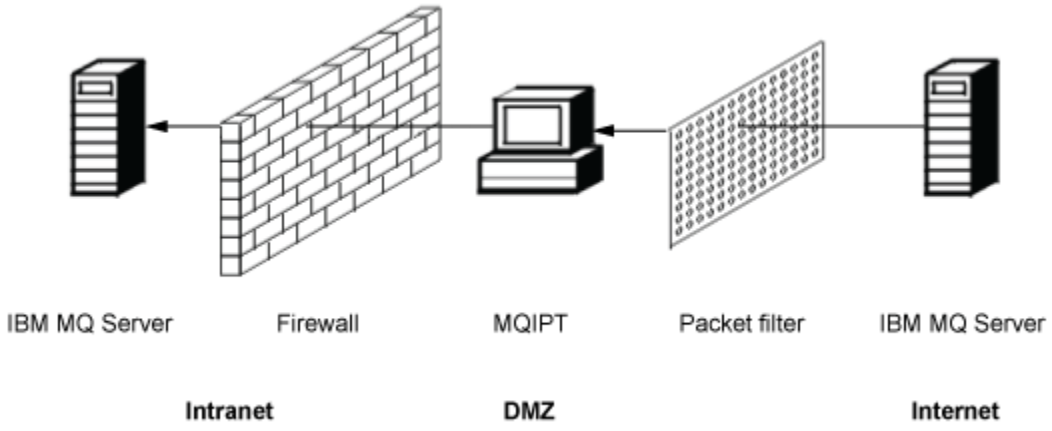


Figure 80. Exemple d'instance MQIPT dans un pare-feu de zone démilitarisée

MQIPT peut communiquer via la tunnellisation HTTP

Si deux instances de MQIPT sont déployées en ligne, elles peuvent communiquer via HTTP. La fonction de tunnellisation HTTP permet aux demandes d'être transmises via des pare-feux, par l'intermédiaire des proxys HTTP existants. Le premier MQIPT insère le protocole IBM MQ dans HTTP et le second extrait le protocole IBM MQ à partir de son encapsuleur HTTP et le transmet au gestionnaire de files d'attente de destination.

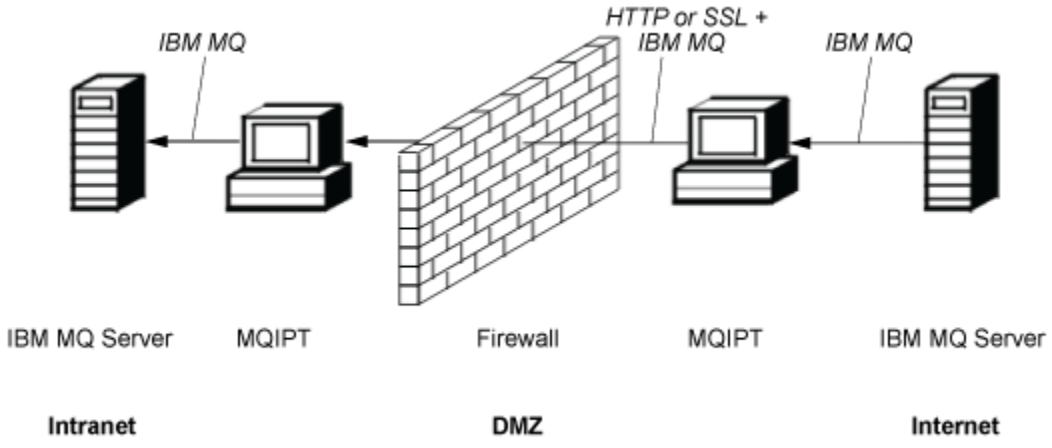


Figure 81. Exemple d'instance de MQIPT avec tunnellisation HTTP

MQIPT peut chiffrer les messages

Si MQIPT est configuré conformément à l'exemple précédent, les demandes peuvent être chiffrées avant leur transmission via des pare-feux. La première instance de MQIPT chiffre les données et la deuxième les déchiffre à l'aide du protocole SSL/TLS avant de les envoyer au gestionnaire de files d'attente de destination.

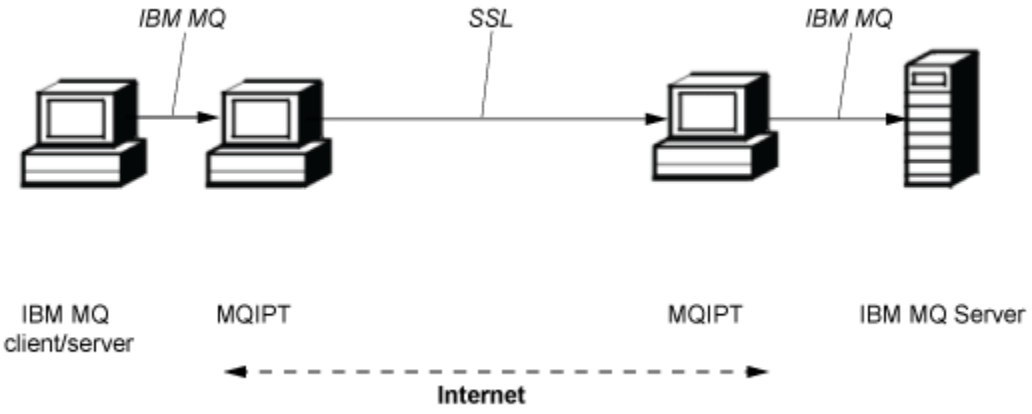


Figure 82. Exemple de MQIPT avec SSL/TLS

Fonctionnement de MQIPT

Dans sa configuration la plus simple, MQIPT fait office de réexpéditeur de protocole IBM MQ. Il est à l'écoute sur un port TCP/IP et accepte les demandes de connexions provenant des canaux IBM MQ.

Si une demande syntaxiquement correcte est reçue, MQIPT établit une autre connexion TCP/IP entre lui-même et le gestionnaire de files d'attente IBM MQ. Il transmet alors tous les paquets de protocole qu'il reçoit de sa connexion entrante au gestionnaire de files d'attente de destination, et renvoie les paquets de protocole provenant du gestionnaire de files d'attente de destination à la connexion entrante d'origine.

Aucune modification du protocole IBM MQ (client/serveur ou gestionnaire de files d'attente vers gestionnaire de files d'attente) n'est nécessaire, car aucune des deux extrémités n'est directement informée de la présence de l'intermédiaire. De nouvelles versions du code client ou serveur d'IBM MQ ne sont pas requises.

Pour utiliser MQIPT, le canal appelant doit être configuré pour utiliser le nom d'hôte et le port de MQIPT, et non le nom d'hôte et le port du gestionnaire de files d'attente de destination. La propriété **CONNAME** du canal IBM MQ est utilisée. MQIPT lit les données entrantes et se contente de les transmettre au gestionnaire de files d'attente de destination. Les autres zones de configuration, telles que l'ID utilisateur et le mot de passe dans un canal client/serveur, sont également transmises au gestionnaire de files d'attente de destination.

Utilisation de plusieurs gestionnaires de files d'attente

MQIPT peut être utilisé pour autoriser l'accès à plusieurs gestionnaires de files d'attente de destination. Dans ce cas, un mécanisme indiquant à MQIPT à quel gestionnaire de files d'attente il doit se connecter doit exister, pour que MQIPT puisse utiliser le numéro de port TCP/IP entrant pour déterminer à quel gestionnaire de files d'attente se connecter.

Par conséquent, vous pouvez configurer MQIPT pour qu'il soit à l'écoute sur plusieurs ports TCP/IP. Chaque port d'écoute est mappé à un gestionnaire de files d'attente de destination via une MQIPT route . Vous pouvez définir jusqu'à 100 routes, qui associent un port d'écoute TCP/IP au nom d'hôte et au port du gestionnaire de files d'attente de destination. Cela signifie que le nom d'hôte (adresse IP) du gestionnaire de files d'attente de destination n'est jamais visible pour le canal émetteur. Chaque route peut prendre en charge plusieurs connexions entre son port d'écoute et sa destination, chaque connexion agissant de façon indépendante.

Fichier de configuration MQIPT

MQIPT utilise un fichier de configuration appelé `mqipt.conf`. Ce fichier contient les définitions de toutes les routes et leurs propriétés associées. Pour plus d'informations sur `mqipt.conf`, voir Administration et configuration de IBM MQ Internet Pass-Thru.

Lorsque MQIPT est lancé, il démarre chaque route répertoriée dans le fichier de configuration. Les messages sont écrits dans la console système indiquant le statut de chaque route. Lorsque MQCPI078 s'affiche pour une route, cela signifie que cette route est prête à accepter des demandes de connexion.

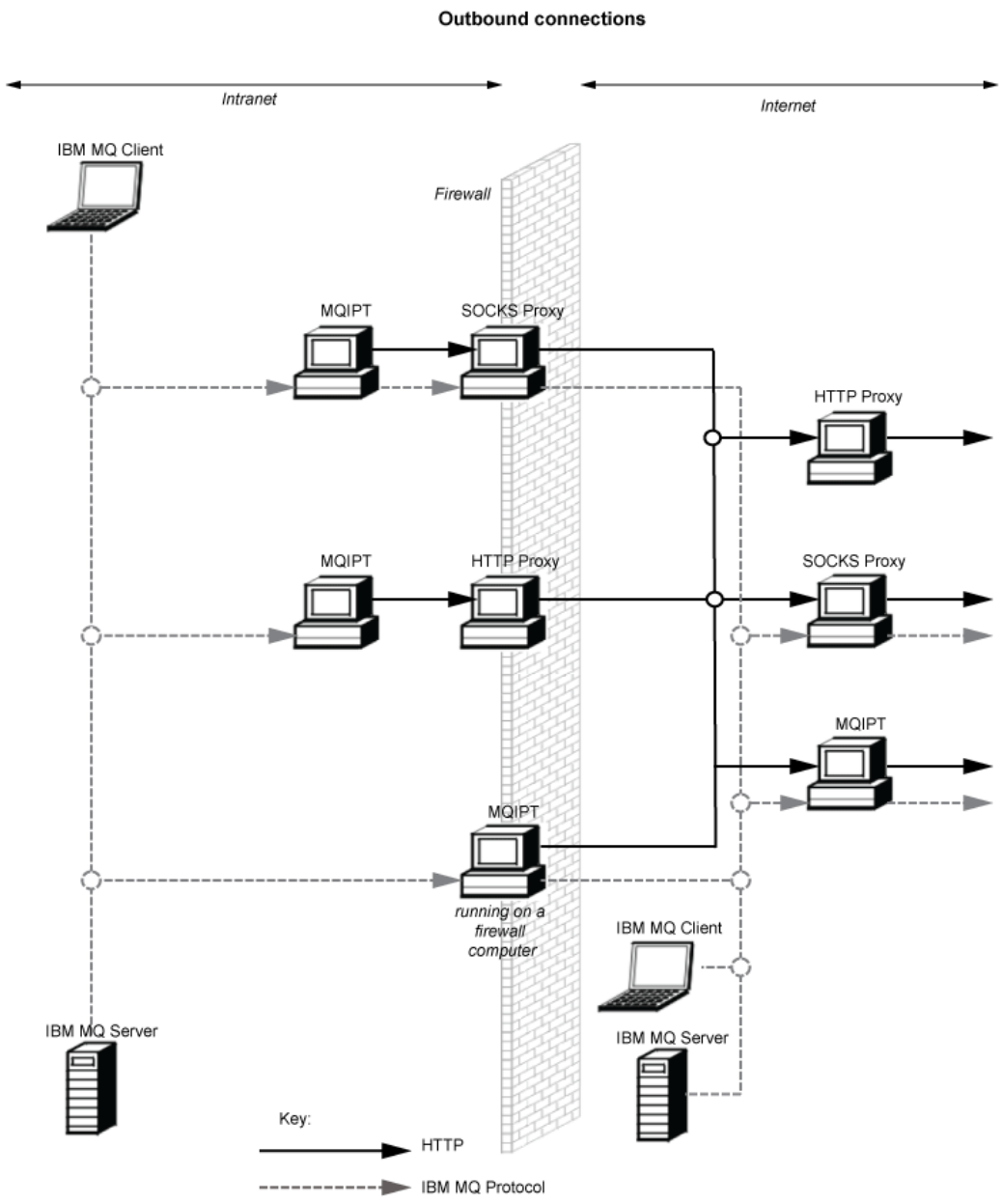
Configurations possibles de MQIPT

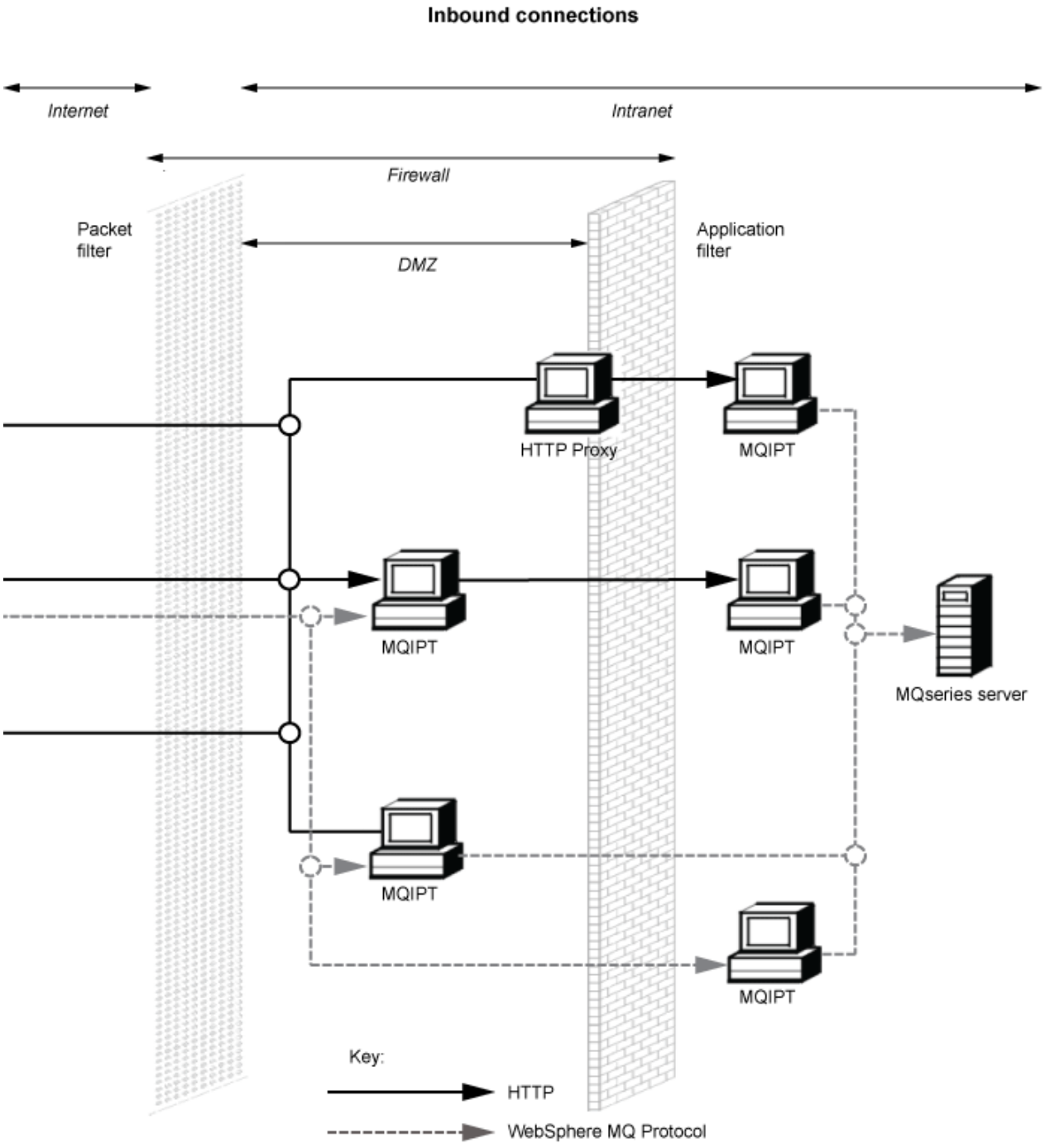
MQIPT peut être utilisé avec IBM MQ et IBM Integration Bus.

La figure ci-après en plusieurs parties représente de nombreuses configurations possibles de MQIPT dans une topologie IBM MQ. Elle illustre les différentes méthodes d'envoi de messages utilisées par MQIPT. Elle représente les clients et les serveurs sur un réseau intranet, à l'intérieur d'un pare-feu et sur Internet en dehors du pare-feu, transmettant des messages à MQIPT, au proxy HTTP ou au proxy SOCKS, qui les retransmettent.

Les messages sont reçus par un proxy MQIPT ou un proxy HTTP dans une zone démilitarisée avant leur transmission à un serveur via le pare-feu entrant.

Notez que le proxy HTTP, le proxy SOCKS, et les ordinateurs MQIPT côté intranet du pare-feu représentent la possibilité de chaîner plusieurs ordinateurs sur Internet. Par exemple, un ordinateur MQIPT pourra communiquer via un ou plusieurs ordinateurs proxy SOCKS ou HTTP, ou d'autres ordinateurs MQIPT, avant d'atteindre sa cible.


Outbound connections
Intranet
Internet
IBM MQ Client
Firewall
MQIPT
SOCKS Proxy
HTTP Proxy
MQIPT
HTTP Proxy
SOCKS Proxy
MQIPT
MQIPT
running on a firewall computer
IBM MQ Client
IBM MQ Server
IBM MQ Server
Key:
HTTP
IBM MQ Protocol



Configurations compatibles

Scénarios de connexion compatibles où un client ou un gestionnaire de files d'attente IBM MQ communique avec MQIPT. La même route ou une seconde route MQIPT est utilisée pour communiquer avec un gestionnaire de files d'attente de destination.

Configurations compatibles avec une seule route MQIPT

Vous pouvez utiliser une seule route MQIPT pour communiquer avec IBM MQ.

Les colonnes du Tableau 26, à la page 307 contiennent les informations suivantes :

1. Le protocole utilisé entre IBM MQ et la route MQIPT. La connexion peut être créée par un client ou un gestionnaire de files d'attente IBM MQ, et peut utiliser les formats et protocoles IBM MQ (FAP) ou un protocole SSL/TLS.
2. Mode dans lequel fonctionne la route MQIPT. Le format de la communication sur Internet entre MQIPT et IBM MQ est déterminé par la configuration de la route MQIPT. Notez que lorsque la table mentionne SSL, vous pouvez également utiliser TLS.
3. Le protocole utilisé entre la route MQIPT et le gestionnaire de files d'attente de destination.

Tableau 26. Configurations valides avec une seule instance de MQIPT

1. IBM MQ Protocole source	**2. Mode de la route MQIPT**	**3. IBM MQ Protocole cible**
FAP	Proxy FAP (par défaut)	FAP
	Serveur FAP et client SSL	SSL/TLS
SSL/TLS	Proxy SSL	SSL/TLS
	Serveur SSL et client FAP	FAP
	Serveur SSL et client SSL	SSL/TLS

Configurations compatibles avec plusieurs routes MQIPT

Vous pouvez choisir d'utiliser plusieurs routes, sur une ou plusieurs instances de MQIPT, afin de communiquer avec IBM MQ.

Les colonnes du Tableau 27, à la page 308 contiennent les informations suivantes :

1. Le protocole utilisé entre IBM MQ et la première route MQIPT. La connexion peut être créée par un client ou un gestionnaire de files d'attente IBM MQ, et peut utiliser les formats et protocoles IBM MQ (FAP) ou un protocole SSL/TLS.
2. Mode dans lequel fonctionne la première route MQIPT. Le format de la communication sur Internet entre MQIPT et IBM MQ est déterminé par la configuration de la route MQIPT. Notez que lorsque la table mentionne SSL, vous pouvez également utiliser TLS.
3. Mode dans lequel fonctionne la seconde route MQIPT.
4. Le protocole utilisé entre la seconde route MQIPT et le gestionnaire de files d'attente de destination.

Tableau 27. Configurations valides avec plusieurs instances de MQIPT

1. IBM MQ Protocole source	2. Mode de la première route MQIPT	3. Mode de la seconde route MQIPT	4. IBM MQ Protocole cible
FAP (par défaut)	Proxy FAP (par défaut)	Proxy FAP (par défaut)	FAP
	Serveur FAP et client SSL	Proxy SSL	SSL/TLS
		Serveur SSL et client FAP	FAP
		Serveur SSL et client SSL	SSL/TLS
	Client HTTP	Serveur HTTP et client SSL	SSL/TLS
	Client HTTPS	Serveur HTTPS et client SSL	SSL/TLS
	Client HTTP	Serveur HTTP	FAP
	Client HTTPS	Serveur HTTPS	FAP
SSL/TLS	Proxy SSL	Proxy SSL	SSL/TLS
		Serveur SSL et client FAP	FAP
		Serveur SSL et client SSL	SSL/TLS
	Client HTTP	Serveur HTTP	FAP
	Client HTTPS	Serveur HTTPS	SSL/TLS
	Client HTTP	Serveur HTTP et client SSL	FAP
	Client HTTPS	Serveur HTTPS et client SSL	SSL/TLS

Configurations de canal prises en charge

Tous les types de canaux IBM MQ sont pris en charge, mais la configuration est limitée aux connexions TCP/IP. Pour un client ou un gestionnaire de files d'attente IBM MQ, MQIPT est présenté comme gestionnaire de files d'attente de destination. Lorsque la configuration de canal requiert un hôte de destination et un numéro de port, le nom d'hôte et le numéro de port d'écoute de MQIPT sont indiqués.

Canaux client/serveur

MQIPT écoute les demandes de connexion client entrantes, puis les transmet via la tunnellisation HTTP, via SSL/TLS, ou en tant que paquets de protocole IBM MQ standard. Si MQIPT utilise la tunnellisation HTTP ou SSL/TLS, il transmet les demandes via une connexion à une autre instance de MQIPT. S'il n'utilise pas la tunnellisation HTTP, il transmet les demandes via une connexion à ce qu'il considère comme le gestionnaire de files d'attente de destination (bien que celui-ci puisse être à son tour une autre instance de MQIPT). Une fois que le gestionnaire de files d'attente de destination a accepté la connexion client, les paquets sont relayés entre le client et le serveur.

Canaux émetteur/récepteur de cluster

Si MQIPT reçoit une demande entrante d'un canal émetteur de cluster, il considère que le gestionnaire de files d'attente prend en charge SOCKS et que la vraie adresse de destination sera obtenue au cours de l'établissement de liaison de SOCKS. Il transmet la demande à l'instance de MQIPT suivante ou au gestionnaire de files d'attente de destination exactement de la même manière que pour les canaux de connexion client. Il en va de même pour les canaux émetteurs de cluster définis automatiquement.

Emetteur/récepteur
Si MQIPT reçoit une demande entrante d'un canal émetteur, il la transmet à l'instance de MQIPT suivante ou au gestionnaire de files d'attente de destination exactement de la même manière que pour les canaux de connexion client. Le gestionnaire de files d'attente de destination valide la demande entrante et démarre le canal récepteur, si besoin est. Toutes les communications entre les canaux émetteur et récepteur (y compris les flux de sécurité) sont relayées.

Demandeur/serveur
Cette combinaison est traitée de la même manière que les configurations précédentes. La validation de la demande de connexion est effectuée par le canal serveur sur le gestionnaire de files d'attente de destination.

Demandeur/émetteur
La configuration de "rappel" est utile si les deux gestionnaires de files d'attente ne sont pas autorisés à se connecter directement l'un à l'autre, mais que tous les deux sont autorisés à se connecter à MQIPT et à accepter les connexions provenant de celui-ci.

Serveur/demandeur et serveur/récepteur
Ils sont gérés par MQIPT de la même manière qu'il gère la configuration `Sender/Receiver` .

Arrêt d'un canal et conditions d'échec

Lorsque MQIPT détecte la fermeture (normale ou anormale) d'un canal IBM MQ, il propage la fermeture du canal. Si vous fermez une route à l'aide de MQIPT, tous les canaux passant par cette route sont fermés.

MQIPT propose une fonction de délai d'inactivité facultative. Si MQIPT détecte qu'un canal a été inactif pendant une période de temps supérieure au délai d'attente, il arrête immédiatement les deux connexions en question.

Les systèmes IBM MQ à chaque extrémité du canal perçoivent ces conditions d'arrêt anormal soit comme des échecs réseau, soit comme un arrêt du canal par leur partenaire. Le canal peut alors redémarrer et effectuer une reprise (si l'échec se produit durant une période d'attente de validation de protocole) comme si MQIPT n'était pas utilisé.

Sécurité des messages

La gestion répartie des files d'attente IBM MQ garantit la distribution des messages. C'est notamment le cas lorsque MQIPT est présent entre les deux extrémités du canal. MQIPT ne stocke pas de données de message et n'est pas impliqué dans la procédure de point de synchronisation qui garantit la distribution des messages.

Lorsque vous utilisez des messages IBM MQ rapides et non persistants, si la route MQIPT échoue ou est redémarrée lorsqu'un message IBM MQ est en cours de transfert, celui-ci peut être perdu. Avant de redémarrer la route, assurez-vous que tous les canaux IBM MQ utilisant la route MQIPT sont inactifs.

Pour plus d'informations sur la sécurité des messages dans IBM MQ, voir Safety of messages.

Gestionnaires de files d'attente multi-instances et haute disponibilité

MQIPT peut être utilisé avec des gestionnaires de files d'attente multi-instances dans des environnements à haute disponibilité.

MQIPT ne dispose d'aucun état permanent et il n'y a donc aucun avantage à basculer MQIPT sur un autre système. À la place, plusieurs instances de MQIPT avec des fichiers de configuration `mqipt.conf` identiques s'exécutent sur des systèmes différents. Surveillez la disponibilité de chaque instance de MQIPT et redémarrez l'instance si nécessaire (sur le même système). Vous disposez ainsi d'un ensemble d'instances identiques de MQIPT pouvant être utilisées pour acheminer les connexions. Vous devez ensuite vous assurer qu'IBM MQ peut acheminer les connexions à MQIPT et que MQIPT peut les transmettre au gestionnaire de files d'attente de destination.

Les canaux IBM MQ sortants peuvent être dirigés vers une instance MQIPT disponible de plusieurs façons, par exemple :

- Utilisez un équilibreur de charge ou un routeur à haute disponibilité, tel qu'IBM Network Dispatcher du produit WebSphere Edge Components.
- Spécifiez plusieurs noms de connexion dans la définition de canal IBM MQ, sous forme de liste dont les éléments sont séparés par une virgule. IBM MQ tente alors de se connecter à chaque adresse MQIPT tour à tour jusqu'à ce qu'il trouve une instance MQIPT disponible.

Vous devez également diriger les connexions de MQIPT vers le gestionnaire de files d'attente de destination. Si la configuration à haute disponibilité garantit que l'adresse IP est reprise en ligne avec le gestionnaire de files d'attente, aucune configuration MQIPT spéciale n'est requise : indiquez l'adresse IP de destination dans la propriété de route **Destination**. Laissez ensuite l'opération de reprise en ligne déplacer l'adresse IP avec le gestionnaire de files d'attente.

Toutefois, si l'adresse IP du gestionnaire de files d'attente change après une reprise en ligne, vous devez configurer MQIPT pour qu'il transmette la connexion à la destination appropriée. Plusieurs méthodes sont possibles :

- Ecrivez un exit de routage qui vérifie les adresses IP et le numéro de port qui sont accessibles, puis remplacez la destination de routage de chaque connexion. Des exemples d'exit de routage sont fournis avec MQIPT. Ils peuvent être adaptés à cet effet.
- Utilisez un équilibreur de charge à haute disponibilité pour rediriger la connexion.
- Définissez plusieurs routes MQIPT, une pour chaque adresse IP et pour chaque port sur lequel le gestionnaire de files d'attente peut être en cours d'exécution. Dirigez ensuite les connexions IBM MQ vers les différentes routes MQIPT, par exemple en répertoriant tous les numéros de port et toutes les adresses IP de la route dans une liste dont les éléments sont séparés par une virgule, dans le nom de connexion du canal de communications sortantes.

Il est également important d'optimiser tous les composants de bout en bout sur chemin d'accès du réseau :

1. L'échec des tentatives de connexion à des systèmes indisponibles doit être rapide, afin que les tentatives de reconnexion puissent passer à la première destination disponible.

   Pour les routes MQIPT SSL, réglez la propriété de route **SSLClientConnectTimeout** afin de garantir un délai court d'échec de connexion aux destinations indisponibles. Consultez la documentation IBM MQ pour plus d'informations sur le réglage du paramètre IBM MQ. De même, consultez la documentation de votre système d'exploitation pour prendre connaissance des détails du réglage de TCP/IP pour le système d'exploitation. Dans tous les cas, les tentatives de connexion ayant échoué doivent renvoyer rapidement un incident réseau (par exemple, un paquet de réinitialisation TCP), ou doivent expirer sans délai inutile.
2. Les connexions actives à un système ayant échoué doivent être éliminées rapidement afin de permettre l'établissement de nouvelles connexions.

   Vous devez également prendre en compte l'impact d'une reprise en ligne lorsque les connexions utilisent intensément MQIPT. Il est probable que les connexions réseau soient interrompues pendant une reprise en ligne. Pour des applications client, vous pouvez utiliser la fonction de reconnexion automatique du client d'IBM MQ pour rétablir des connexions interrompues. Pour les canaux de messages, vous pouvez spécifier un intervalle court entre les nouvelles tentatives afin de garantir une reconnexion rapide du canal. Consultez la documentation IBM MQ pour plus d'informations sur la reconnexion automatique du client et la configuration de la relance des canaux de message.

IBM MQ Console et REST API

Vous pouvez utiliser IBM MQ Console et REST API pour administrer IBM MQet effectuer des opérations de messagerie à l'aide de HTTP.

- Vous pouvez utiliser l' IBM MQ Console pour effectuer des tâches d'administration de base à partir d'un navigateur Web. Pour plus d'informations, voir Administration à l'aide de IBM MQ Console.
- Vous pouvez utiliser l' administrative REST API pour administrer des objets IBM MQ , tels que des gestionnaires de files d'attente et des files d'attente, ainsi que des agents et des transferts Managed File Transfer . Pour plus d'informations, voir Administration à l'aide de REST API.

- Vous pouvez utiliser messaging REST API pour effectuer une messagerie point-à-point simple et une messagerie de publication. Pour plus d'informations, voir Messagerie à l'aide de REST API.

Options d'installation

IBM MQ Console et REST API s'exécutent sur un serveur WebSphere Liberty , appelé mqweb. Depuis IBM MQ 9.3.5, vous pouvez installer le serveur mqweb en tant que composant facultatif dans une installation IBM MQ ou en tant qu'installation IBM MQ Web Server autonome.

V 9.4.0 Linux **Installation IBM MQ Web Server autonome**

Depuis IBM MQ 9.4.0, le serveur mqweb peut s'exécuter dans une installation autonome d' IBM MQ Web Server. Une installation IBM MQ Web Server autonome vous permet d'installer et d'exécuter le serveur mqweb sur des systèmes distincts de vos installations IBM MQ . L'installation d'un IBM MQ Web Server autonome offre une plus grande flexibilité quant aux systèmes et au nombre de systèmes sur lesquels vous choisissez d'exécuter vos serveurs mqweb. Si nécessaire, plusieurs instances du serveur mqweb peuvent être exécutées sur différentes machines pour fournir l'évolutivité et la disponibilité dont vous avez besoin.

Si vous avez acheté l'autorisation d'utilisation d' IBM MQ , vous pouvez installer autant de copies que nécessaire du IBM MQ Web Serverautonome. Les installations de IBM MQ Web Server ne sont pas comptabilisées dans le cadre de votre autorisation d'utilisation d'IBM MQ. Pour plus d'informations sur l'octroi de licence IBM MQ, voir Informations sur les licences IBM MQ.

Les restrictions suivantes s'appliquent dans une installation IBM MQ Web Server autonome:

- IBM MQ Console peut être utilisé pour administrer uniquement les gestionnaires de files d'attente éloignées.
- messaging REST API ne peut être utilisé qu'avec des gestionnaires de files d'attente éloignées.
- Le administrative REST API n'est pas disponible.

Le IBM MQ Web Server autonome est pris en charge uniquement sur les plateformes Linux .

Pour plus d'informations sur l'installation du IBM MQ Web Serverautonome, voir Installation du IBM MQ Web Serverautonome.

Composant facultatif d'une installation IBM MQ

Vous pouvez choisir d'installer le composant IBM MQ Console et REST API dans le cadre d'une installation IBM MQ .

Toutes les fonctions IBM MQ Console et REST API sont disponibles lorsque le serveur mqweb s'exécute dans une installation IBM MQ .

- Le IBM MQ Console peut être utilisé pour administrer les gestionnaires de files d'attente locales et éloignées.
- messaging REST API peut être utilisé avec des gestionnaires de files d'attente locales et éloignées.
- Le administrative REST API peut être utilisé pour administrer les gestionnaires de files d'attente locales et éloignées.

Pour utiliser le composant IBM MQ Console et REST API , installez le composant suivant dans le cadre de votre installation IBM MQ :

- AIX Sous AIX, installez l'ensemble de fichiers `mqm.web.rte` .
- IBM i Sous IBM i, installez le composant WEB.
- Linux Sous Linux, installez le composant `MQSeriesWeb` .
- Windows Sous Windows, installez la fonction `Web Administration` .
- z/OS Sous z/OS, installez la fonction `IBM MQ for z/OS UNIX System Services Web Components` .

Remarques

:NONE.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Consultez votre représentant IBM local pour obtenir des informations sur les produits et services actuellement disponibles dans votre région. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service IBM puisse être utilisé. Tout produit, programme ou service fonctionnellement équivalent qui ne porte pas atteinte à un droit de propriété intellectuelle IBM peut être utilisé à la place. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Pour obtenir des informations sur les licences relatives aux informations sur deux octets (DBCS), contactez le service de la propriété intellectuelle IBM de votre pays ou envoyez vos demandes de renseignements, par écrit, à :

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

Le paragraphe suivant ne s’applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE "EN L'ETAT" SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et/ou programmes décrits dans ce document.

Les références à des sites Web non IBM sont fournies uniquement à titre d'information et n'impliquent en aucune façon une adhésion de ces sites Web. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Coordinateur d'interopérabilité logicielle, département 49XA
3605 Autoroute 52 N

Rochester, MN 55901
U.S.A.

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans le présent document et tous les éléments sous disponibles s'y rapportant sont fournis par IBM conformément aux dispositions du Contrat sur les produits et services IBM, aux Conditions Internationales d'Utilisation de Logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples mentionnent des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

Licence sur les droits d'auteur :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Documentation sur l'interface de programmation

Les informations d'interface de programmation, si elles sont fournies, sont destinées à vous aider à créer un logiciel d'application à utiliser avec ce programme.

Ce manuel contient des informations sur les interfaces de programmation prévues qui permettent au client d'écrire des programmes pour obtenir les services d' IBM MQ.

Toutefois, lesdites informations peuvent également contenir des données de diagnostic, de modification et d'optimisation. Ces données vous permettent de déboguer votre application.

Important : N'utilisez pas ces informations de diagnostic, de modification et d'optimisation en tant qu'interface de programmation car elles sont susceptibles d'être modifiées.

Marques

IBM, le logo IBM , ibm.com, sont des marques d' IBM Corporation dans de nombreux pays. La liste actualisée de toutes les marques d' IBM est disponible sur la page Web "Copyright and trademark

information"www.ibm.com/legal/copytrade.shtml. Les autres noms de produits et de services peuvent être des marques d'IBM ou d'autres sociétés.

Microsoft et Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

UNIX est une marque de The Open Group aux Etats-Unis et dans certains autres pays.

Linux est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Ce produit inclut des logiciels développés par le projet Eclipse (https://www.eclipse.org/).

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques d'Oracle et/ou de ses sociétés affiliées.

Référence :

(1P) P/N: