9.4

IBM MQ 기술 개요

참고

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 285 페이지의 『주의사항』에 있는 정보를 확인하십시오.

이 개정판은 새 개정판에 별도로 명시하지 않는 한, IBM® MQ 의 버전 9릴리스 4및 모든 후속 릴리스와 수정에 적용됩니다.

목차

IBM MQ 기술 개요

IBM MQ를 사용하여 애플리케이션을 연결하고 조직에서 정보 분배를 관리하십시오.

IBM MQ에서는 프로그램이 일관된 API(Application Programming Interface)를 사용하여 서로 다른 컴포넌트(프로세서, 운영 체제, 서브시스템 및 통신 프로토콜)가 존재하는 네트워크에서 서로 다른 항목과 통신할 수 있습니다. 이 인터페이스를 사용하여 설계 및 기록된 애플리케이션은 메시지 큐잉 애플리케이션이라고 합니다.

다음 하위 주제를 사용하여 메시지 큐잉 및 IBM MQ에서 제공되는 다른 기능에 대해 확인하십시오.

관련 개념

IBM MQ 소개

제품 요구사항 및 지원 정보를 제공하는 위치

관련 태스크

IBM MQ 아키텍처 계획

관련 참조

6 페이지의 『메시지 큐잉의 기본 기능 및 이점』
이 정보는 메시지 큐잉의 일부 기능 및 이점을 강조합니다. 이 정보에서는 메시지 큐잉의 보안 및 데이터 무결성과 같은 기능을 설명합니다.

메시지 큐잉 소개

IBM MQ 제품을 사용하면 프로그램이 일관된 API(Application Programming Interface)를 사용하여 다른 컴포넌트(프로세서, 운영 체제, 서브시스템 및 통신 프로토콜)의 네트워크에서 서로 통신할 수 있습니다.

이 인터페이스를 사용하여 설계되고 작성된 애플리케이션은 메시징 및 큐잉 스타일을 사용하기 때문에 메시지 큐잉 애플리케이션이라고 합니다.

- 메시징은 프로그램이 서로 직접 호출하지 않고 메시지의 각 기타 데이터를 송신하여 통신하는 것을 의미합니다.
- 큐잉은 메시지가 스토리지의 큐에 배치되어 프로그램이 서로 독립적으로 다른 속도와 시간에 다른 위치에서 논리 연결 없이 실행할 수 있도록 하는 것을 의미합니다.

메시지 큐잉이 다년간 데이터 처리에 사용되어 왔습니다. 메시지 큐잉은 전자 메일에서 가장 많이 사용됩니다. 큐잉 없이 전자 메시지를 장거리 송신하는 경우 라우트의 모든 노드가 메시지 전달에 대해 사용 가능해야 하고 주소가 로그온되어야 하며 사용자가 이 주소에 메시지 송신을 시도한다는 사실에 대해 인식하고 있어야 합니다. 큐잉 시스템에서 시스템이 메시지를 전달할 준비가 될 때까지 메시지는 중간 노드에 저장됩니다. 최종 목적지에서 주소가 메시지를 읽을 준비가 될 때까지 메시지는 전자 편지함에 저장됩니다.

그렇다 하더라도 다수의 복잡한 비즈니스 트랜잭션이 오늘날 큐잉 없이 처리됩니다. 대형 네트워크에서 시스템은 사용 준비 상태인 수천의 연결을 유지보수하고 있을 수 있습니다. 시스템의 한 부분에 문제가 생기는 경우 시스템의 많은 부분이 사용 불가능하게 됩니다.

메시지 큐잉을 프로그램용 전자 메일로 생각할 수 있습니다. 메시지 큐잉 환경에서 애플리케이션 스위트를 구성하는 각 프로그램은 특정 요청에 대한 응답으로 명확하고 독립적인 기능을 수행합니다. 다른 프로그램과 통신하려면 프로그램이 사전정의된 큐에 메시지를 넣어야 합니다. 다른 프로그램은 큐에서 메시지를 검색하고 메시지에 포함된 요청 및 정보를 처리합니다. 따라서 메시지 큐잉은 프로그램 대 프로그램 통신의 스타일입니다.

큐잉은 애플리케이션이 메시지를 처리할 준비가 될 때까지 메시지가 보유되는 메커니즘입니다. 큐잉을 통해 다음을 수행할 수 있습니다.

- 통신 코드를 작성할 필요 없이 프로그램(각각 다른 환경에서 실행될 수 있음) 간에 통신합니다.
- 프로그램이 메시지를 처리하는 순서를 선택하십시오.
- 메시지 수가 임계값을 초과할 때 둘 이상의 프로그램이 큐를 서비스하도록 구성하여 시스템에 로드를 밸런스화합니다.
- 1차 시스템이 사용 가능하지 않은 경우 대체 시스템이 큐를 서비스하도록 구성하여 애플리케이션의 사용가능성을 증가시키십시오.

메시지 큐 개념

간단하게 큐라고 하는 메시지 큐는 메시지를 송신할 수 있는 이름 지정된 목적지입니다. 큐를 서비스하는 프로그램이 메시지를 검색할 때까지 메시지는 큐에 누적됩니다.

큐 관리자에서 큐가 상주하고 관리됩니다(8 페이지의 『메시지 큐잉 용어』 참조). 큐의 물리적 특성은 큐 관리자가 실행되는 운영 체제에 따라 다릅니다. 큐는 컴퓨터 메모리의 휘발성 버퍼이거나 영구 스토리지 디바이스(예: 디스크)에서 데이터 세트일 수 있습니다. 큐의 실제 관리는 큐 관리자가 담당하며 참여하는 애플리케이션 프로그램에 대해 분명하지 않게 작성됩니다.

프로그램은 큐 관리자의 외부 서비스를 통해서만 큐에 액세스할 수 있습니다. 프로그램은 큐를 열고, 큐로(부터) 메시지를 넣고, 메시지를 가져오고 큐를 닫을 수 있습니다. 또한 큐의 속성을 설정 및 조회할 수 있습니다.

메시지 큐잉의 다른 스타일

포인트-투-포인트

큐에는 메시지가 한 개만 배치되며 하나의 애플리케이션이 해당 메시지를 수신합니다.

포인트-투-포인트 메시징에서 송신 애플리케이션이 수신 애플리케이션에 대한 정보를 알아야 수신 애플리케이션으로 메시지를 송신할 수 있습니다. 예를 들어, 송신 애플리케이션은 정보를 송신할 큐 이름을 알아야 하며 큐 관리자 이름도 지정할 수 있습니다.

발행/구독

발행 애플리케이션이 발행하는 각 메시지의 사본은 모든 관련 애플리케이션으로 전달됩니다. 관련 애플리케이션이 하나 또는 여러 개일 수 있으며, 전혀 없을 수도 있습니다. 발행/구독에서 관심 애플리케이션은 구독자로 알려져 있으며 메시지는 구독으로 식별되는 큐에 삽입됩니다.

발행/구독 메시징을 사용하면 정보의 제공자를 해당 정보의 이용자와 분리할 수 있습니다. 송신 애플리케이션과 수신 애플리케이션은 송수신할 정보를 위해 서로에 대해 알지 않아도 됩니다. 자세한 정보는 55 페이지의 『발행/구독 메시징』의 내용을 참조하십시오.

애플리케이션 설계자 및 개발자에 대한 메시지 큐잉의 이점

IBM MQ를 통해 애플리케이션 프로그램은 메시지 큐잉을 사용하여 메시지 구동 처리에 참여할 수 있습니다. 애플리케이션 프로그램은 적절한 메시지 큐잉 소프트웨어 제품을 사용하여 여러 플랫폼 사이에서 통신할 수 있습니다. 예를 들어, z/OS® 애플리케이션은 IBM MQ for z/OS를 통해 통신할 수 있습니다. 애플리케이션은 근본적인 통신의 메커닉으로부터 보호됩니다. 메시지 큐잉의 그 밖의 이점은 다음과 같습니다.

- 여러 애플리케이션에서 공유할 수 있는 소형 프로그램을 사용하여 애플리케이션을 설계할 수 있습니다.
- 이러한 빌딩 블록을 다시 사용하여 새 애플리케이션을 빠르게 빌드할 수 있습니다.
- 메시지 큐잉 기술을 사용하도록 작성된 애플리케이션은 큐 관리자가 작업하는 도중에 발생한 변경사항에 영향을 받지 않습니다.
- 통신 프로토콜을 사용할 필요가 없습니다. 큐 관리자는 사용자를 위해 통신의 모든 측면을 다룹니다.
- 메시지가 프로그램에 송신될 때 메시지를 수신하는 프로그램이 실행 중일 필요는 없습니다. 메시지는 큐에 보유됩니다.

메시지 큐잉을 사용하지 않는 애플리케이션에 비해 프로그래밍 기술에 대한 요구가 낮아지고, 필요한 개발자의 수가 적어지고 개발이 빨라지기 때문에 설계자는 애플리케이션 비용을 줄일 수 있습니다.

IBM MQ는 애플리케이션이 실행되는 위치마다 메시지 큐 인터페이스(또는 MQI)라는 공용 애플리케이션 프로그래밍 인터페이스를 구현합니다. 이를 통해 하나의 플랫폼에서 다른 플랫폼으로 애플리케이션 프로그램을 쉽게 포트시킬 수 있습니다.

MQI에 대한 자세한 내용은 MQI(Message Queue Interface) 개요를 참조하십시오.

메시지 큐잉의 기본 기능 및 이점

이 정보는 메시지 큐잉의 일부 기능 및 이점을 강조합니다. 이 정보에서는 메시지 큐잉의 보안 및 데이터 무결성과 같은 기능을 설명합니다.

메시지 큐잉 기술을 사용하는 애플리케이션의 기본 기능은 다음과 같습니다.

- 7 페이지의 『프로그램 간의 직접 연결이 없음』
- 8 페이지의 『시간 독립 통신』
- 8 페이지의 『소형 프로그램』
- 8 페이지의 『메시지 구동 처리』
- 8 페이지의 『이벤트 중심 처리』
- 8 페이지의 『메시지 우선순위』
- 8 페이지의 『보안』
- 8 페이지의 『데이터 무결성』
- 8 페이지의 『복구 지원』

참고: IBM MQ 클라이언트 및 서버를 고려할 때 새 플랫폼에서의 추가 IBM MQ MQI clients 지원을 위해 서버 애플리케이션을 변경할 필요가 없습니다. 마찬가지로, IBM MQ MQI client는 변경 없이도 추가 유형의 서버와 함께 작동할 수 있습니다.

프로그램 간의 직접 연결이 없음

메시지 큐잉은 간접적 프로그램 대 프로그램 통신용 기술입니다. 프로그램이 서로 통신하는 모든 애플리케이션 내에서 이 메시지 큐잉을 사용할 수 있습니다. 메시지를 큐 관리자가 소유한 큐에 넣는 하나의 프로그램과 큐에서 메시지를 가져오는 다른 하나의 프로그램에 의해서 통신이 발생합니다.

프로그램은 다른 프로그램이 큐에 넣는 메시지를 가져올 수 있습니다. 다른 프로그램은 수신 프로그램과 동일한 큐 관리자 또는 다른 큐 관리자에 연결될 수 있습니다. 이러한 다른 큐 관리자는 다른 시스템, 다른 컴퓨터 시스템 상에 있거나 다른 비즈니스 또는 엔터프라이즈 내에 있을 수도 있습니다.

메시지 큐를 사용하여 통신하는 프로그램 간에는 물리적 접속이 없습니다. 프로그램이 큐 관리자가 소유하는 큐에 메시지를 송신하고 다른 프로그램은 해당 큐에서 메시지를 검색합니다(7 페이지의 그림 1 참조).

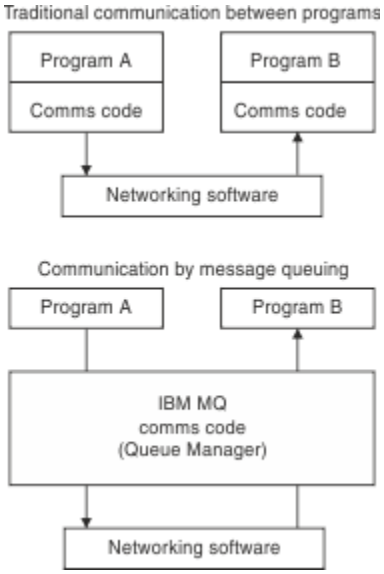


그림 *1.* 일반 통신과 메시지 큐잉 비교

전자 메일의 경우와 마찬가지로 트랜잭션의 일부인 개별 메시지는 저장 및 전달 네트워크를 통해 이동합니다.기초. 노드 간의 링크가 실패하면 링크가 복원되거나 연산자 또는 프로그램이 메시지를 다시 전송할 때까지 메시지가 보관됩니다.

메시지가 큐에서 큐로 이동하는 메커니즘은 프로그램으로부터 숨겨집니다. 따라서 프로그램이 좀 더 단순해집니다.

시간 독립 통신

다른 프로그램에 작업 수행을 요청하는 프로그램은 요청에 대한 응답을 기다릴 필요가 없습니다. 다른 작업을 수행하고 응답이 도착할 때 또는 나중에 응답을 처리할 수 있습니다. 메시징 애플리케이션을 작성할 때 프로그램이 언제 메시지를 송신하는지 또는 대상이 메시지를 받을 수 있는 상태인지에 대해 알거나 염려할 필요가 없습니다. 메시지는 손실되지 않으며 대상이 메시지를 처리할 준비가 될 때까지 큐 관리자가 보유합니다. 메시지는 프로그램이 제거할 때까지 큐에 남습니다. 이는 송신 및 수신 애플리케이션 프로그램이 연결 해제되었으며 송신자가 수신자의 메시지의 수신 확인을 기다릴 필요 없이 처리를 계속할 수 있음을 의미합니다. 대상 애플리케이션은 메시지가 송신될 때 실행 중일 필요조차 없습니다. 대상 애플리케이션은 시작된 후에 메시지를 검색할 수 있습니다.

소형 프로그램

메시지 큐잉을 사용하면 소형, 자급 프로그램 사용의 이점을 활용할 수 있습니다. 작업의 모든 파트를 순차적으로 수행하는 단일, 대형 프로그램 대신에 작업을 여러 개의 작고 독립적인 프로그램으로 분산시킬 수 있습니다. 요청 프로그램은 개별 프로그램 각각에 해당 기능을 수행하라고 요청하는 메시지를 송신합니다. 각 프로그램이 완료되면 그 결과를 하나 이상의 메시지로 되돌려 보냅니다.

메시지 구동 처리

메시지가 큐에 도착할 때 이 메시지는 트리거를 사용하여 애플리케이션을 자동으로 시작할 수 있습니다. 필요에 따라 하나의 메시지 또는 다수의 메시지가 처리될 때 애플리케이션을 중지할 수 있습니다.

이벤트 중심 처리

프로그램은 큐의 상태에 따라 제어할 수 있습니다. 예를 들어, 프로그램이 메시지가 큐에 도착하는 즉시 시작되도록 구성하거나 큐에 특정 우선순위보다 상위에 있는 10개의 메시지 또는 임의의 우선순위를 가지는 10개의 메시지가 있을 때까지 프로그램이 시작되지 않도록 지정할 수 있습니다.

메시지 우선순위

프로그램은 메시지를 큐에 넣을 때 메시지에 우선순위를 지정할 수 있습니다. 이는 큐에서 새 메시지가 추가되는 위치를 판별합니다.

프로그램은 메시지가 큐에 있는 순서 또는 특정 메시지를 가져오는 순서대로 큐에서 메시지를 가져올 수 있습니다. (프로그램이 이전에 송신한 요청에 대한 응답을 검색하고 있으면 프로그램은 특정 메시지를 가져오려고 할 수 있습니다.)

보안

큐 관리자를 사용하는 경우의 애플리케이션의 인증, 큐 관리자에서 큐와 같은 자원을 사용하는 경우의 권한 검사, 네트워크를 통해 이동하고 큐에 상주하는 메시지 데이터의 암호화를 포함하여 보안 기능이 제공됩니다. 보안에 대한 자세한 정보는 보안 개요를 참조하십시오.

데이터 무결성

데이터 무결성은 작업 단위로 제공됩니다. 작업 단위의 시작 및 종료 동기화가 MQGET 또는 MQPUT 각각에 대한 옵션으로 완전하게 지원되어 작업 단위의 결과가 커미트되거나 롤백되도록 허용합니다. 동기점 지원은 애플리케이션에 대해 선택된 동기점 조정의 형식에 따라 IBM MQ에 대해 내부적으로 또는 외부적으로 작동합니다.

복구 지원

가능한 복구의 경우, 모든 지속적 IBM MQ 업데이트가 로그됩니다. 복구가 필요한 경우에 모든 지속 메시지가 복원되고, 모든 인플라이트 트랜잭션이 롤백되며 모든 동기점 커미트 및 백아웃이 제어 중인 동기점 관리자에 의해 일반적인 방식으로 핸들링됩니다. 지속 메시지에 대한 자세한 정보는 메시지 지속성을 참조하십시오.

메시지 큐잉 용어

이 정보는 메시지 큐잉에 사용되는 일부 용어에 대한 자세한 설명을 제공합니다.

여기에는 다음이 포함됩니다.

- 채널
- 클러스터
- IBM MQ MQI client
- z/OS 그룹 내 큐잉
- 메시지
- 메시지 채널 에이전트
- 메시지 디스크립터
- 포인트-투-포인트
- 공개/등록
- 큐
- 큐 관리자
- z/OS 큐 공유 그룹
- z/OS 공유 큐
- 구독
- 주제

채널

채널은 한 큐 관리자에서 다른 큐 관리자로 메시지를 이동시키는 데 사용되며 기본 통신 프로토콜로부터 애플리케이션을 보호합니다. 큐 관리자들은 동일하거나 서로 다른 플랫폼에서 동일한 시스템 또는 다른 시스템에 존재할 수 있습니다. 전송되는 메시지는 여러 위치에서 생성할 수 있습니다.

- 하나의 노드에서 다른 노드로 데이터를 전송하는 사용자 작성 애플리케이션 프로그램.
- PCF 명령 또는 MQAI를 사용하는 사용자 작성 관리 애플리케이션.
- IBM MQ Explorer입니다.
- 도구 이벤트 메시지를 다른 큐 관리자에 전송하는 큐 관리자.
- 원격 관리 명령을 다른 큐 관리자에 전송하는 큐 관리자(예: MQSC 명령 또는 administrative REST API).

채널에 대한 자세한 정보는 29 페이지의 『채널 정의』의 내용을 참조하십시오.

클러스터

클러스터는 논리적으로 연관된 큐 관리자의 네트워크입니다.

클러스터링 없이 분산 큐잉을 사용하는 IBM MQ 네트워크에서 모든 큐 관리자는 독립적입니다. 한 큐 관리자가 다른 큐 관리자에게 메시지를 송신해야 하는 경우 리모트 큐 관리자에 대한 전송 큐 및 채널이 정의되어 있어야 합니다.

클러스터를 사용하는 두 가지 다른 이유는 시스템 관리를 줄이는 것과 사용가능성 및 워크로드 밸런싱을 높이기 위해서 입니다.

가장 작은 클러스터라도 설정이 끝나고 나면 단순화된 시스템 관리에 도움이 됩니다. 클러스터의 일부인 큐 관리자는 보다 소수의 정의만을 필요로 하기 때문에 정의에서 오류가 발생하는 위험을 줄일 수 있습니다.

클러스터링에 대한 자세한 정보는 클러스터를 참조하십시오.

IBM MQ MQI client

IBM MQ MQI 클라이언트는 IBM MQ의 독립적으로 설치 가능한 컴포넌트입니다. MQI 클라이언트를 통해 통신 프로토콜을 사용하는 IBM MQ 애플리케이션을 실행하고 다른 플랫폼에 있는 하나 이상의 MQI(Message Queue Interface)와 상호 작용하며 해당 큐 관리자에 접속할 수 있습니다.

IBM MQ MQI client 컴포넌트의 설치 및 사용 방법에 대한 자세한 내용은 다음 주제를 참조하십시오.

- AIX AIX®에 IBM MQ 클라이언트 설치
- Linux Linux®에 IBM MQ 클라이언트 설치
- Windows Windows에 IBM MQ 클라이언트 설치
- IBM i IBM i에 IBM MQ 클라이언트 설치

및 서버 및 클라이언트 간의 연결 구성.

그룹 내 큐잉

z/OS

큐 공유 그룹에 있는 큐 관리자는 일반 채널을 사용하여 통신할 수 있습니다. 또는 사용자가 그룹 내 큐잉(IGQ)이라는 기술을 사용하여 채널 정의 없이 빠른 메시지 전송을 수행할 수 있습니다. 이는 IBM MQ for z/OS에만 적용됩니다.

그룹 내 큐잉에 대한 자세한 정보는 194 페이지의 『Intra-group queuing』 의 내용을 참조하십시오.

메시지

메시지 큐잉에서 메시지는 하나의 프로그램에서 송신한 데이터의 콜렉션이며 다른 프로그램에도 사용됩니다. IBM MQ 메시지를 참조하십시오.

메시지 유형에 대한 정보는 메시지 유형을 참조하십시오.

메시지 채널 에이전트

채널의 한쪽 끝에는 메시지 채널 에이전트이 있습니다. 메시지 채널 에이전트 쌍(송신 에이전트와 수신 에이전트)은 채널을 구성하고 한 큐 관리자에서 다른 관리자로 메시지를 이동합니다.

메시지 채널 에이전트 사용 방법에 대한 정보는 분산 큐 관리 소개를 참조하십시오.

메시지 디스크립터

IBM MQ 메시지는 제어 정보 및 애플리케이션 데이터로 구성됩니다.

제어 정보는 메시지 디스크립터 구조(MQMD)에 정의되고 다음과 같은 사항을 포함합니다.

- 메시지의 유형
- 메시지의 ID
- 메시지 전달의 우선순위

애플리케이션 데이터의 구조 및 컨텐츠는 IBM MQ가 아닌 참여 프로그램에 의해 판별됩니다.

자세한 정보는 MQMD를 참조하십시오.

포인트-투-포인트 메시징

지점간 메시징에서 각 메시지는 단일 생성 애플리케이션에서 단일 이용 애플리케이션으로 이동합니다. 메시지는 메시지를 큐에 넣는 생성 애플리케이션을 통해 전송되며 이용 애플리케이션은 큐에서 메시지를 받습니다.

발행/구독 메시징

발행/구독 메시징에서 발행 애플리케이션이 발행한 각 메시지의 사본은 모든 관심 애플리케이션으로 전달됩니다. 관심 애플리케이션의 개수는 하나 또는 다수이거나 전혀 없을 수도 있습니다. 발행/구독에서 관심 애플리케이션은 구독자로 알려져 있으며 메시지는 구독으로 식별되는 큐에 삽입됩니다.

자세한 정보는 55 페이지의 『발행/구독 메시징』 의 내용을 참조하십시오.

큐

메시지를 전송할 수 있는 이름 지정된 대상입니다. 큐를 서비스하는 프로그램이 메시지를 검색할 때까지 메시지는 큐에 누적됩니다.

자세한 정보는 18 페이지의 『큐』의 내용을 참조하십시오.

큐 관리자

큐 관리자는 애플리케이션에 큐잉 서비스를 제공하는 시스템 프로그램입니다.

큐 관리자는 API(Application Programming Interface)를 제공하여 프로그램이 큐로(부터) 메시지를 넣고 가져올 수 있도록 합니다. 큐 관리자는 관리자가 새 큐를 작성하고, 기존 큐의 특성을 대체하고 큐 관리자의 조작을 제어할 수 있도록 추가 기능을 제공합니다.

IBM MQ 메시지 큐잉 서비스를 시스템에서 사용하려면 큐 관리자가 실행 중이어야 합니다. 단일 시스템에는 둘 이상의 실행 중인 큐 관리자가 있을 수 있습니다(예를 들어, 라이브 시스템에서 테스트 시스템을 분리하려는 경우). 애플리케이션에 대해 각 큐 관리자는 연결 핸들(*Hconn*)로 식별됩니다.

여러 다른 애플리케이션은 큐 관리자의 서비스를 동시에 사용할 수 있으며 이러한 애플리케이션은 전부 비관련일 수 있습니다. 프로그램이 큐 관리자의 서비스를 이용하게 하려면 해당 큐 관리자에 대한 연결을 설정해야 합니다.

애플리케이션이 다른 큐 관리자에 연결된 애플리케이션에 메시지를 송신하게 하려면 큐 관리자는 서로 통신할 수 있어야 합니다. IBM MQ에서는 저장 후 전달 프로토콜을 구현하여 이러한 애플리케이션 간의 안전한 메시지 전달을 보장합니다.

자세한 정보는 25 페이지의 『큐 관리자』의 내용을 참조하십시오.

큐 공유 그룹

z/OS

동일한 세트의 공유 큐에 액세스할 수 있는 큐 관리자는 큐 공유 그룹(QSG)이라는 그룹을 구성합니다. 공유 큐를 저장하는 커플링 기능(CF)을 사용하여 서로 통신합니다. 이는 IBM MQ for z/OS에만 적용됩니다.

자세한 정보는 150 페이지의 『Shared queues and queue sharing groups』의 내용을 참조하십시오.

공유 큐

z/OS

공유 큐는 메시지가 있는 로컬 큐의 유형으로 SYSPLEX에 있는 하나 이상의 큐 관리자가 액세스할 수 있습니다. 이 큐는 동일한 큐 관리자를 사용하여 둘 이상의 애플리케이션에서 공유하는 큐와 같지 않습니다. 이는 IBM MQ for z/OS에만 적용됩니다.

구독

발행/구독 애플리케이션은 특정 주제에 대한 메시지에 관심을 등록할 수 있습니다. 이를 수행하는 애플리케이션은 구독자로 알려져 있으며 기간 구독은 일치하는 메시지가 처리를 위해 큐에 대기되는 방식을 정의합니다.

구독에는 구독자의 ID 및 발행이 배치될 목적지 큐의 ID에 관한 정보가 들어있습니다. 또한 발행이 목적지 큐에 배치되는 방법에 대한 정보도 들어있습니다.

자세한 정보는 58 페이지의 『구독자 및 구독』의 내용을 참조하십시오.

토픽

토픽은 발행/구독 메시지에서 발행된 정보의 제목을 설명하는 문자열입니다.

토픽은 발행/구독 시스템에서 메시지를 성공적으로 전달하기 위한 핵심 사항입니다. 각 메시지에 특정 목적지 주소를 포함하는 대신 발행자가 각 메시지에 대한 토픽을 지정합니다. 큐 관리자가 해당 토픽을 구독하는 구독자 목록과 토픽을 일치시킨 다음 각 해당 구독자에게 메시지를 전달합니다.

자세한 정보는 60 페이지의 『토픽』의 내용을 참조하십시오.

메시지 및 큐

메시지 및 큐는 메시지 큐잉 시스템의 기본 컴포넌트입니다.

메시지 개념

메시지는 바이트 문자열이며 문자열을 사용하는 애플리케이션에 의미가 있습니다. 메시지는 하나의 애플리케이션 프로그램에서 다른 애플리케이션 프로그램으로 (또는 동일한 애플리케이션의 다른 파트 사이에서) 정보를 전송하는 데 사용됩니다. 애플리케이션은 동일한 플랫폼 또는 다른 플랫폼에서 실행될 수 있습니다.

IBM MQ 메시지는 다음으로 구성되어 있습니다.

- 애플리케이션 데이터.애플리케이션 데이터의 컨텐츠 및 구조는 이를 사용하는 애플리케이션 프로그램에 의해 정의됩니다.
- 메시지 디스크립터.메시지 디스크립터는 메시지를 식별하고 메시지 유형 및 송신 애플리케이션에 의해 메시지에 지정된 우선순위와 같은 추가 제어 정보를 포함합니다.

  메시지 디스크립터의 형식은 IBM MQ에 의해 정의됩니다. 메시지 디스크립터에 대한 자세한 설명은 MQMD - 메시지 디스크립터를 참조하십시오.
- 메시지 특성. 메시지에 대한 메타데이터. 메시지 특성의 컨텐츠는 이를 사용하는 애플리케이션 프로그램에 의해 정의됩니다. 자세한 정보는 메시지 특성을 참조하십시오.

메시지 길이

메시지 길이를 100MB의 최대 길이로 증가시킬 수 있기는 하지만 (여기에서 1MB는 1 048 576바이트임) 기본 최대 메시지 길이는 4MB입니다. 실제로 메시지 길이는 다음으로 제한될 수 있습니다.

- 수신 큐에 대해 정의된 최대 메시지 길이
- 큐 관리자에 대해 정의된 최대 메시지 길이
- 큐에 의해 정의된 최대 메시지 길이
- 수신 또는 송신 애플리케이션에 의해 정의된 최대 메시지 길이
- 메시지에 사용 가능한 스토리지 용량

애플리케이션이 요구하는 모든 정보를 송신하는 데 여러 메시지가 필요할 수 있습니다.

애플리케이션이 메시지를 송신하고 수신하는 방법

애플리케이션 프로그램은 **MQI 호출**을 사용하여 메시지를 송신하고 수신합니다.

예를 들어, 큐에 메시지를 배치하기 위해 애플리케이션은 다음을 수행합니다.

1. MQI `MQOPEN` 호출을 발행하여 필요한 큐를 엽니다.
2. MQI `MQPUT` 호출을 발행하여 메시지를 큐에 배치합니다.

다른 애플리케이션이 MQI `MQGET` 호출을 발행하여 동일한 큐에서 메시지를 검색할 수 있습니다.

MQI 호출에 대한 자세한 정보는 MQI 호출을 참조하십시오.

큐의 개념

큐는 메시지를 저장하는 데 사용되는 데이터 구조입니다.

각 큐는 큐 관리자가 소유합니다. 큐 관리자는 소유한 큐를 유지보수하고 적절한 큐로 수신한 모든 메시지 저장을 담당합니다. 애플리케이션 프로그램 또는 큐 관리자의 일반적인 조작으로 큐에 메시지를 배치할 수 있습니다.

사전정의된 큐 및 동적 큐

큐는 작성되는 방식에 따라 특성화할 수 있습니다.

- **사전정의된 큐**는 적절한 MQSC 또는 PDF 명령을 사용하는 관리자에 의해 작성됩니다.사전정의된 큐는 영구적이며 이를 사용하는 애플리케이션과는 독립적으로 존재하고 IBM MQ를 다시 시작해도 없어지지 않습니다.

- **동적 큐**는 애플리케이션이 모델 큐의 이름을 지정하는 MQOPEN 요청을 발행할 때 작성됩니다. 작성된 큐는 모델 큐라고 하는 템플리트 큐 정의를 기본으로 합니다.MQSC 명령 DEFINE QMODEL을 사용하여 모델 큐를 작성할 수 있습니다. 모델 큐의 속성(예: 모델 큐에 저장될 수 있는 최대 메시지 수)은 모델 큐에서 작성되는 동적 큐로 상속됩니다.

  모델 큐에는 동적 큐가 영구적 또는 임시적인지 여부를 지정하는 속성이 있습니다. 영구적 큐는 애플리케이션과 큐 관리자가 재시작해도 남아 있지만 임시적 큐는 재시작 시에 손실됩니다.

큐에서 메시지 검색

적합하게 권한 부여된 애플리케이션은 다음 검색 알고리즘에 따라 큐에서 메시지를 검색할 수 있습니다.

- FIFO(First In, First Out)
- 메시지 디스크립터에 정의된 메시지 우선순위입니다. 동일한 우선순위를 가진 메시지는 FIFO(First In, First Out) 기준으로 검색됩니다.
- 특정 메시지에 대한 프로그램 요청

애플리케이션에서의 MQGET 요청은 사용되는 메소드를 판별합니다.

IBM MQ 오브젝트

큐 관리자는 IBM MQ 오브젝트에 대한 특성을 정의합니다. 이 특성 값은 IBM MQ에서 해당 오브젝트를 처리하는 방법에 영향을 줍니다. IBM MQ 명령 및 인터페이스를 사용하여 오브젝트를 작성하고 관리합니다. 애플리케이션에서 MQI(Message Queue Interface)를 사용하여 이들 오브젝트를 제어합니다. 오브젝트는 프로그램에서 취급할 때 MQOD(IBM MQ *object descriptor*)에서 식별합니다.

오브젝트 관리

오브젝트 관리에는 다음 태스크가 포함됩니다.

- 큐 관리자 시작 및 중지
- 애플리케이션용 오브젝트(특히 큐)를 작성
- 오브젝트의 속성 표시 또는 변경
- 오브젝트 삭제
- 다른(원격) 시스템에 있는 큐 관리자에 대한 통신 경로를 작성하기 위해 채널에 대한 작업
- 큐 관리자의 클러스터를 작성하여 전체 관리 프로세스를 단순화하고 워크로드를 밸런스화함

동적 큐의 경우를 제외하고 오브젝트에 대한 작업을 할 수 있으려면 먼저 큐 관리자에 대해 오브젝트를 정의해야 합니다.

IBM MQ 명령을 사용하여 오브젝트 관리 조작을 수행하는 경우 큐 관리자는 조작을 수행하는 데 필요한 권한 레벨을 보유했는지 확인합니다. 마찬가지로, 애플리케이션이 MQOPEN 호출을 사용하여 오브젝트를 열 때 큐 관리자는 애프리케이션이 필요한 권한 레벨을 가지고 있는지 확인한 후 해당 오브젝트에 대한 액세스를 허용합니다. 열려 있는 오브젝트의 이름에 대해 확인합니다.

다음 방법을 사용하여 오브젝트를 정의 및 관리할 수도 있습니다.

- 프로그래밍 가능 명령 형식 참조 및 관리 태스크 자동화에서 설명된 PCF 명령
- MQSC 명령에 설명된 MQSC 명령
- z/OS IBM MQ for z/OS 관리 에 설명된 IBM MQ for z/OS 조작 및 제어판
- Windows Linux IBM MQ Explorer (Windows 및 Intel 시스템 전용 Linux). 자세한 정보는 MQ 탐색기 소개를 참조하십시오.

또한 다음 방법을 사용하여 오브젝트를 관리할 수도 있습니다.

- 키보드로 입력하는 제어 명령. 제어 명령을 사용하여 IBM MQ for Multiplatforms 관리를 참조하십시오.
- 프로그램에서의 MQAI(IBM MQ Administration Interface) 호출. IBM MQ 관리 인터페이스(MQAI)를 참조하십시오.

ALW AIX, Linux, and Windows에서 IBM MQ 명령의 순서의 경우, MQSC 기능을 사용하여 파일에 보유된 일련의 명령을 실행할 수 있습니다. 자세한 정보는 MQSC 명령을 사용하여 IBM MQ 관리를 참조하십시오.

IBM i 정기적으로 사용하는 IBM MQ for IBM i 명령 시퀀스의 경우 CL 프로그램을 작성할 수 있습니다. 자세한 정보는 CL 명령을 사용하여 IBM MQ for IBM i 관리를 참조하십시오.

z/OS 정기적으로 사용하는 IBM MQ for z/OS 명령 순서의 경우 명령을 포함하는 메시지를 작성하고 이러한 메시지를 시스템 명령 입력 큐에 배치하는 관리 프로그램을 작성할 수 있습니다. 큐 관리자는 명령행 또는 조작 및 제어판에서 입력한 명령을 처리하는 방식과 동일한 방식으로 이러한 큐에 대한 메시지를 처리합니다. 이 기술은 IBM MQ 관리 프로그램 작성에 설명되어 있으며, IBM MQ for z/OS에서 제공하는 메일 관리자 샘플 애플리케이션에 예시되어 있습니다. 이 샘플에 대한 설명은 IBM MQ for z/OS용 샘플 프로그램을 참조하십시오.

오브젝트 속성

오브젝트의 특성은 그 속성으로 정의됩니다. 일부는 지정할 수 있고 그 외에는 보기만 가능합니다.

예를 들어, 큐가 수용할 수 있는 최대 메시지 길이는 큐의 **MaxMsgLength** 속성으로 정의할 수 있으며 큐를 작성할 때 이 속성을 지정할 수 있습니다. **DefinitionType** 속성은 큐가 작성된 방법을 지정하며 이 속성은 표시만 할 수 있습니다.

IBM MQ에는 다음 속성을 참조하는 두 가지 방식이 있습니다.

- 해당 PCF 이름 사용(예: **MaxMsgLength**)
- 해당 MQSC 명령 이름 사용(예: MAXMSGL)

큐 공유 그룹

z/OS

동일한 세트의 공유 큐에 액세스할 수 있는 큐 관리자는 큐 공유 그룹(QSG)이라는 그룹을 구성하고 공유 큐를 저장하는 커플링 기능(CF)을 사용하여 서로 통신합니다. QSG는 엄격히 오브젝트가 아닙니다.

공유 큐는 큐 공유 그룹에 있는 하나 이상의 큐 관리자가 액세스할 수 있는 메시지가 있는 로컬 큐의 유형입니다. 이 큐는 동일한 큐 관리자를 사용하여 둘 이상의 애플리케이션에서 공유하는 큐와 같지 않습니다.

큐 공유 그룹은 최대 4자의 이름을 가집니다. 이 이름은 네트워크에서 고유해야 하며 큐 관리자 이름과 달라야 합니다.

중요사항: 공유 큐 및 큐 공유 그룹은 IBM MQ for z/OS에서만 지원됩니다.

자세한 정보는 150 페이지의 『Shared queues and queue sharing groups』의 내용을 참조하십시오.

시스템 기본 오브젝트

시스템 기본 오브젝트는 큐 관리자가 작성될 때마다 자동으로 작성되는 오브젝트 정의 세트입니다. 설치 시 애플리케이션에서 사용하기 위해 이러한 오브젝트 정의를 복사하고 수정할 수 있습니다.

기본 오브젝트 이름에는 어간(SYSTEM)이 포함됩니다(예를 들어, 기본 로컬 큐는 SYSTEM.DEFAULT.LOCAL.QUEUE이고 기본 수신자 채널은 SYSTEM.DEF.RECEIVER임). 이러한 오브젝트의 이름을 바꿀 수 없으며 이러한 이름의 기본 오브젝트가 필요합니다.

오브젝트를 정의할 때 명시적으로 지정하지 않은 모든 속성은 적절한 기본 오브젝트에서 복사됩니다. 예를 들어, 로컬 큐를 정의하는 경우 지정하지 않은 속성은 기본 큐 SYSTEM.DEFAULT.LOCAL.QUEUE로부터 가져옵니다.

자세한 정보는 시스템 및 기본 오브젝트 를 참조하십시오.

오브젝트 유형

관리 태스크 중 다수가 다양한 IBM MQ 오브젝트 유형의 조작을 포함합니다.

IBM MQ 오브젝트 이름 지정에 대한 정보는 32 페이지의 『IBM MQ 오브젝트 이름 지정』의 내용을 참조하십시오.

큐 관리자에 작성된 기본 오브젝트에 대한 정보는 14 페이지의 『시스템 기본 오브젝트』의 내용을 참조하십시오.

다른 유형의 IBM MQ 오브젝트에 대한 정보는 다음을 참조하십시오.

인증 정보 오브젝트

인증 정보 오브젝트는 인증서 폐기 확인을 수행하는 데 필요한 정의를 제공합니다.

큐 관리자 인증 정보 오브젝트는 TLS(Transport Layer Security)에 대한 IBM MQ 지원의 일부입니다. 폐기된 인증서를 확인하는 데 필요한 정의를 제공합니다. 인증 기관은 더 이상 신뢰할 수 없는 인증서를 폐기합니다.

MQSC 명령 **DEFINE AUTHINFO**를 사용하여 인증 정보 오브젝트를 정의할 수 있습니다. 인증 정보 오브젝트의 속성에 대한 자세한 정보는 **DEFINE AUTHINFO**를 참조하십시오.

인증 정보 오브젝트와 함께 다음 IBM MQ 제어 명령을 사용할 수 있습니다.

- **setmqaut**(권한 부여 또는 취소)
- **dspmqaut**(오브젝트 권한 표시)
- **dmpmqaut**(덤프 권한)
- **rcrmqobj**(오브젝트 재작성)
- **rcdmqimg**(매체 이미지 기록)
- **dspmqfls**(파일 이름 표시)

TLS의 개요 및 인증 정보 오브젝트의 사용은 TLS (Transport Layer Security) 개념 및 IBM MQ 의 TLS 보안 프로토콜의 내용을 참조하십시오.

채널

채널은 한 큐 관리자에서 다른 큐 관리자로 통신 경로를 제공하는 오브젝트입니다.

자세한 정보는 26 페이지의 『채널』의 내용을 참조하십시오.

통신 정보 오브젝트

IBM MQ 멀티캐스트에서는 낮은 지연, 높은 팬아웃, 신뢰할 수 있는 멀티캐스트 메시징을 제공합니다. 통신 정보(COMMINFO) 오브젝트는 멀티캐스트 전송을 사용하는 데 필요합니다.

자세한 정보는 94 페이지의 『IBM MQ 멀티캐스트』의 내용을 참조하십시오.

COMMINFO 오브젝트는 멀티캐스트 전송과 연관된 속성을 포함하는 IBM MQ 오브젝트입니다. 이 속성에 대한 자세한 정보는 DEFINE COMMINFO를 참조하십시오. COMMINFO 오브젝트의 작성에 대한 자세한 정보는 멀티캐스트 시작하기를 참조하십시오.

리스너

리스너는 다른 큐 관리자 또는 클라이언트에서 네트워크 요청을 승인하는 프로세스이며 연관된 채널을 시작합니다.

리스너 프로세스 는 **runmqlsr** 제어 명령을 사용하여 시작할 수 있습니다.

리스너 오브젝트는 IBM MQ 오브젝트이며, 이를 통해 큐 관리자의 범위 내에서 리스너 프로세스의 시작 및 중단을 관리할 수 있습니다. 리스너 오브젝트의 속성을 정의하여 다음을 수행하십시오.

- 리스너 프로세스를 구성합니다.
- 큐 관리자가 시작 및 중지될 때 자동으로 리스너 프로세스를 시작 및 중지할지 여부를 지정합니다.

중요사항: z/OS 리스너 오브젝트는 IBM MQ for z/OS에서 지원되지 않습니다. 채널 시작기를 사용하여 IBM MQ for z/OS가 청취를 구현하는 방법에 대한 자세한 정보는 146 페이지의 『The channel initiator on z/OS』의 내용을 참조하십시오.

이름 목록

이름 목록은 클러스터 이름, 큐 이름 또는 인증 정보 오브젝트 이름 목록이 포함된 IBM MQ 오브젝트입니다. 클러스터에서 이름 목록은 큐 관리자가 저장소에 보유하는 클러스터의 목록을 식별하는 데 사용될 수 있습니다.

이름 목록은 다른 IBM MQ 오브젝트 목록이 포함된 IBM MQ 오브젝트입니다. 일반적으로 이름 목록은 큐 그룹을 식별하는 데 사용되는 트리거 모니터와 같은 애플리케이션에 의해 사용됩니다. 이름 목록 사용의 장점은 애플리케이션과 독립적으로 유지보수된다는 점이며 이름 목록을 사용하는 애플리케이션을 중지하지 않고도 업데이트할 수 있다는 것입니다. 또한 하나의 애플리케이션이 실패하는 경우 이름 목록은 영향을 받지 않으며 다른 애플리케이션은 계속해서 이름 목록을 사용할 수 있습니다.

이름 목록은 둘 이상의 IBM MQ 오브젝트가 참조하는 클러스터 목록을 유지보수하기 위해 큐 관리자 클러스터와 함께 사용되기도 합니다.

MQSC 명령 DEFINE NAMELIST 및 ALTER NAMELIST 를 사용하여 이름 목록을 정의하고 수정할 수 있습니다.

참고: z/OS 또는 z/OS에서 IBM MQ for z/OS 조작 및 제어판을 사용할 수 있습니다.

프로그램은 MQI를 사용하여 이러한 이름 목록에 포함되는 큐를 알아낼 수 있습니다. 이름 목록의 조직은 애플리케이션 설계자 및 시스템 관리자가 담당합니다.

사용 가능한 이름 목록 속성 목록은 이름 목록의 속성을 참조하십시오.

프로세스 정의

프로세스 정의 오브젝트를 통해 큐 관리자 사용에 대한 애플리케이션의 속성을 정의하여 운영자가 개입할 필요 없이 애플리케이션을 시작할 수 있습니다.

프로세스 정의 오브젝트는 IBM MQ 큐 관리자에서 트리거 이벤트에 대한 응답으로 시작하는 애플리케이션을 정의합니다. 프로세스 정의 속성에는 애플리케이션 ID, 애플리케이션 유형 및 애플리케이션에 특정한 데이터가 포함됩니다. 자세한 정보는 24 페이지의 『IBM MQ에서 특정 목적에 사용되는 큐』에서 이니시에이션 큐를 참조하십시오.

트리거를 사용한 IBM MQ 애플리케이션 시작에 설명된 대로 운영자 개입이 없이도 애플리케이션이 시작될 수 있도록 허용하려면, 애플리케이션의 속성을 큐 관리자에 알려야 합니다. 이러한 속성은 프로세스 정의 오브젝트에 정의되어 있습니다.

오브젝트를 작성할 때 **ProcessName** 속성은 수정됩니다. 그러나 IBM MQ 명령을 사용하여 다른 속성을 변경할 수 있습니다.

참고: z/OS 또는 z/OS에서 IBM MQ for z/OS 조작 및 제어판을 사용할 수 있습니다.

MQINQ - 오브젝트 속성 조회를 사용하여 모든 속성의 값에 대해 조회할 수 있습니다.

사용 가능한 프로세스 정의 속성 목록은 프로세스 정의 속성을 참조하십시오.

큐

IBM MQ 큐는 애플리케이션이 메시지를 넣을 수 있고 애플리케이션이 메시지를 가져올 수 있는 이름 지정된 오브젝트입니다.

자세한 정보는 18 페이지의 『큐』의 내용을 참조하십시오.

큐 관리자

IBM MQ 큐 관리자는 애플리케이션에 큐잉 서비스를 제공하며 그에 포함된 큐를 관리합니다.

자세한 정보는 25 페이지의 『큐 관리자』의 내용을 참조하십시오.

서비스

서비스 오브젝트는 큐 관리자가 시작 또는 중지될 때 실행할 프로그램을 정의하는 방법입니다.

프로그램 유형은 다음 중 하나입니다.

서버

서버는 SERVER로 지정된 매개변수 SERVTYPE을 가지는 서비스 오브젝트입니다. 서버 서비스 오브젝트는 지정된 큐 관리자가 시작될 때 실행되는 프로그램의 정의입니다. 서버 프로세스의 한 인스턴스만을 동시에 실행할 수 있습니다. 실행하는 중에 MQSC 명령인 DISPLAY SVSTATUS를 사용하여 서버 프로세스의 상태를 모니터할 수 있습니다. 일반적으로 서버 서비스 오브젝트는 데드 레터 핸들러 또는 트리거 모니터와 같은 프로그램의 정의이지만 실행될 수 있는 프로그램은 IBM MQ와 함께 제공되는 프로그램으로 제한되지 않습니다. 추가적으로 서버 서비스 오브젝트는 지정된 큐 관리자가 종료되어 프로그램이 종료될 때 실행할 수 있는 명령을 포함하도록 정의할 수 있습니다.

명령

명령은 COMMAND로 지정된 매개변수 SERVTYPE을 가지는 서비스 오브젝트입니다. 명령 서비스 오브젝트는 지정된 큐 관리자가 시작되거나 중지될 때 실행되는 프로그램의 정의입니다. 명령 프로세스의 다중 인스턴스는 동시에 실행될 수 있습니다. 명령 서비스 오브젝트는 프로그램이 일단 실행되면 큐 관리자가 프로그램을 모니터하지 않는다는 점에서 서버 서비스 오브젝트와 다릅니다. 일반적으로 명령 서비스 오브젝트는 단기적이고 하나 또는 둘 이상의 다른 태스크를 시작하는 것과 같은 특정 태스크를 수행하는 프로그램의 정의입니다.

중요사항: z/OS 서비스 오브젝트는 IBM MQ for z/OS에서 지원되지 않습니다.

자세한 정보는 서비스에 대한 작업을 참조하십시오.

스토리지 클래스

z/OS

스토리지 클래스는 하나 이상의 큐를 페이지 세트에 맵핑합니다.

이는 해당 큐에 대한 메시지가 해당 페이지 세트에 저장됨(버퍼링을 조건으로)을 의미합니다.

스토리지 클래스는 IBM MQ for z/OS에서만 지원됩니다.

스토리지 클래스에 대한 자세한 정보는 z/OS에서 IBM MQ 환경 계획을 참조하십시오.

토픽 오브젝트

토픽 오브젝트는 기본이 아닌 특정 속성을 토픽에 지정할 수 있는 IBM MQ 오브젝트입니다.

토픽은 특정 토픽 문자열로 발행 또는 구독하는 애플리케이션에서 정의합니다. 토픽 문자열은 토픽을 슬래시(/)로 구분하여 토픽의 계층을 지정할 수 있습니다. 이러한 계층은 토픽 트리로 시각화할 수 있습니다. 예를 들어, 애플리케이션이 토픽 문자열인 `/Sport/American Football` 및 `/Sport/Soccer`를 발행하면 `American Football` 및 `Soccer`의 두 개 하위를 가진 상위 노드 `Sport`가 있는 토픽 트리가 작성됩니다.

토픽은 토픽 트리에 있는 첫 번째 상위 관리 노드에서 해당 속성을 상속합니다. 특정 토픽 트리에 관리 토픽 노드가 없는 경우, 모든 토픽은 기본 토픽 오브젝트 SYSTEM.BASE.TOPIC에서 해당 속성을 상속합니다.

해당 노드의 토픽 문자열을 토픽 오브젝트의 TOPICSTR 속성에 지정하여 토픽 트리의 모든 노드에서 토픽 오브젝트를 작성할 수 있습니다. 또한 관리 토픽 노드에 다른 속성을 정의할 수도 있습니다. 이 속성에 대한 자세한 정보는 MQSC 명령 또는 PCF 명령을 사용하여 관리 자동화를 참조하십시오. 기본적으로, 각 토픽 오브젝트는 가장 가까운 상위 관리 토픽 노드에서 해당 속성을 상속합니다.

토픽 오브젝트는 애플리케이션 개발자로부터 전체 토픽 트리를 숨기는 데 사용될 수 있습니다. 토픽 `/Sport/American Football`에 대해 이름이 `FOOTBALL.US` 인 토픽 오브젝트가 작성되는 경우 애플리케이션은 동일한 결과의 문자열 `/Sport/American Football` 대신 이름이 `FOOTBALL.US` 인 오브젝트를 발행하거나 구독할 수 있습니다.

토픽 오브젝트의 토픽 문자열에 #, +, / 또는 * 문자를 입력한 경우 이 문자는 문자열 내에서 정상적인 문자로 처리되며 토픽 오브젝트와 연관된 토픽 문자열의 일부로 간주됩니다.

토픽 오브젝트에 대한 자세한 정보는 55 페이지의 『발행/구독 메시징』의 내용을 참조하십시오.

관련 개념

5 페이지의 『메시지 큐잉 소개』
IBM MQ 제품을 사용하면 프로그램이 일관된 API(Application Programming Interface)를 사용하여 다른 컴포넌트(프로세서, 운영 체제, 서브시스템 및 통신 프로토콜)의 네트워크에서 서로 통신할 수 있습니다.

관련 참조
MQSC 명령

큐

IBM MQ 큐 및 큐 속성에 대한 소개입니다.

메시지가 큐에 저장되어 넣기 애플리케이션이 해당 메시지에 대한 응답을 기다리는 경우 이 응답을 기다리는 동안 다른 작업을 할 수 있습니다. 애플리케이션은 메시지 큐 인터페이스 개요에 설명된 MQI(Message Queue Interface)를 사용하여 큐에 액세스합니다.

메시지를 큐에 넣기 전에 먼저 큐가 작성되어 있어야만 합니다. 큐 관리자가 큐를 소유하며 해당 큐 관리자는 다수의 큐를 소유할 수 있습니다. 하지만 각 큐는 큐 관리자 내에서 고유한 이름을 가져야 합니다.

큐는 큐 관리자를 통해 유지보수됩니다. 대부분의 경우 각 큐는 해당 큐 관리자가 실제로 관리하지만 애플리케이션 프로그램에 대해서는 분명하지 않습니다. IBM MQ for z/OS 공유 큐는 큐 공유 그룹에 있는 모든 큐 관리자가 관리할 수 있습니다.

큐를 작성하는 데 IBM MQ 명령(MQSC), PCF 명령 또는 플랫폼별 인터페이스를 사용할 수 있습니다. 예를 들어, IBM MQ for z/OS 조작 및 제어판은 플랫폼에 따라 다릅니다.

애플리케이션에서 임시 작업용 로컬 큐를 동적으로 작성할 수 있습니다. 예를 들어, 응답 대상 큐를 작성할 수 있습니다(애플리케이션이 종료된 후에는 필요하지 않음). 자세한 정보는 22 페이지의 『동적 및 모델 큐』의 내용을 참조하십시오.

큐를 사용하기 전에 큐를 열어 큐에 대해 수행하려는 작업을 지정해야 합니다. 예를 들어, 다음의 작업을 위해 큐를 열 수 있습니다.

- 메시지 찾아보기 전용(메시지 검색은 아님)
- 메시지 검색(및 다른 프로그램과 액세스 공유 또는 독점 액세스 사용)
- 큐에 메시지 넣기
- 큐의 속성 조회
- 큐의 속성 설정

큐를 열 때 지정할 수 있는 전체 옵션 목록은 MQOPEN - 오브젝트 열기를 참조하십시오.

큐의 속성

큐의 일부 속성은 큐가 정의될 때 지정되며 이후에 변경할 수 없습니다(예: 큐의 유형). 다른 큐 속성은 변경할 수 있는 속성으로 그룹화할 수 있습니다.

- 큐 처리 중에 큐 관리자에 의해 지정(예: 큐의 현재 용량)
- 명령에 의해서만 지정(예: 큐의 텍스트 설명)
- MQSET 호출을 사용하는 애플리케이션에 의해 지정(예: Put 조작이 큐에서 허용되는지 여부)

MQING 호출을 사용하여 모든 속성의 값을 찾을 수 있습니다.

둘 이상의 큐 유형에 공용인 속성은 다음과 같습니다.

QName
큐의 이름

QType
큐의 유형

QDesc
큐에 대한 텍스트 설명

InhibitGet
프로그램이 큐에서 메시지를 가져오도록 허용하는지 여부. 하지만, 리모트 큐에서 메시지를 가져올 수는 없습니다.

InhibitPut
프로그램이 큐에 메시지를 넣도록 허용하는지 여부

DefPriority
큐에 넣은 메시지의 기본 우선순위

DefPersistence
큐에 넣은 메시지의 기본 지속성

범위
이 큐에 대한 항목이 이름 서비스에 존재하는지 여부 제어

z/OS **Scope** 속성은 z/OS에서 지원되지 않습니다.

이 속성에 대한 자세한 설명은 큐의 속성을 참조하십시오.

큐 정의 방법

MQSC DEFINE 명령 또는 PCF 큐 작성 명령을 사용하여 IBM MQ 에 큐를 정의할 수 있습니다. 이 명령은 큐의 유형 및 해당 속성을 지정합니다. 예를 들어, 로컬 큐 오브젝트에는 애플리케이션이 MQI 호출에서 해당 큐를 참조할 때 발생하는 일을 지정하는 속성이 있습니다. 속성의 예는 다음과 같습니다.

- 애플리케이션이 큐에서 메시지를 검색할 수 있는지의 여부(GET 사용)
- 애플리케이션이 메시지를 큐에 넣을 수 있는지의 여부(PUT 사용)
- 큐에 대한 액세스가 하나의 애플리케이션에만 독점적인지 또는 애플리케이션 사이에서 공유되는지 여부
- 큐에 동시에 저장될 수 있는 최대 메시지 수 (최대 큐 용량)
- 큐에 넣을 수 있는 메시지의 최대 길이

큐를 정의하는 데 사용할 수 있는 다양한 플랫폼별 인터페이스가 있습니다.

관련 개념
51 페이지의 『클러스터 큐』
클러스터 큐는 클러스터 큐 관리자에 의해 호스팅되며 클러스터의 다른 큐 관리자가 사용할 수 있는 큐입니다.

44 페이지의 『데드-레터 큐』
데드-레터 큐(또는 미전달 메시지)는 올바른 목적지로 라우트할 수 없는 경우 메시지를 송신하는 큐입니다. 일반적으로, 각 큐 관리자에는 데드-레터 큐가 있습니다.

PCF 명령을 사용하여 관리 자동화
IBM MQ Console: 큐에 대한 작업

관련 태스크
MQSC 명령을 사용하여 IBM MQ 관리
MQ 탐색기로 큐 관리자 및 오브젝트 작성 및 구성
IBM i CL 명령을 사용하여 IBM MQ for IBM i 관리
z/OS IBM MQ for z/OS 에서 MQSC및 PCF 명령을 실행할 수 있는 소스

관련 참조
52 페이지의 『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.

관련 정보
150 페이지의 『What is a shared queue?』

로컬 큐

전송, 시작, 데드 레터, 명령, 기본값, 채널 및 이벤트 큐는 로컬 큐의 유형입니다.

프로그램이 연결된 큐 관리자가 큐를 소유하고 있는 경우 큐는 프로그램에 로컬로 표시됩니다. 메시지를 로컬 큐에서 가져오거나 로컬 큐에 넣을 수 있습니다.

큐 정의 오브젝트는 큐에 넣은 실제 메시지뿐만 아니라 큐 정의 정보도 포함하고 있습니다.

각 큐 관리자는 특수 용도로 사용하는 몇몇 로컬 큐를 가질 수 있습니다.

전송 큐

애플리케이션이 리모트 큐로 메시지를 송신할 때 로컬 큐 관리자는 전송 큐라는 특수 로컬 큐에 메시지를 저장합니다. 애플리케이션은 전송 큐에 직접 또는 리모트 큐 정의를 통해 간접적으로 메시지를 넣습니다.

큐 관리자가 리모트 큐 관리자로 메시지를 보낼 때 다음 순서를 사용하여 전송 큐를 식별합니다.

1. 리모트 큐의 로컬 정의의 XMITQ 속성에 이름 지정된 전송 큐.
2. 리모트 큐 관리자와 동일한 이름을 가지는 전송 큐. 이 값은 리모트 큐 로컬 정의의 XMITQ의 기본값입니다.
3. 로컬 큐 관리자의 DEFXMITQ 속성에 이름 지정된 전송 큐.

메시지 채널 에이전트는 전송 큐와 연관된 채널 프로그램으로서 메시지를 다음 목적지로 전달합니다. 다음 목적지란 메시지 채널이 연결된 큐 관리자를 말합니다. 메시지의 마지막 목적지와 큐 관리자는 다를 수 있습니다. 메시지가 다음 목적지로 전달되면, 전송 큐에서 삭제됩니다. 메시지는 최종 목적지에 도달하는 과정에서 많은 큐 관리자를 통과해야 할 수 있습니다. 라우트에 따라 각각 다음 목적지로 전송하기 위해 대기 중인 메시지를 보유하는 각 큐 관리자에 전송 큐를 정의해야 합니다. 일반적인 전송 큐는 메시지의 최종 목적지가 서로 달라도 다음 목적지에 대한 메시지를 보유합니다. 클러스터 전송 큐는 여러 목적지에 대한 메시지를 보유합니다. 각 메시지의 `correlID`는 다음 목적지로 전송할 메시지가 있는 채널을 식별합니다.

큐 관리자에 여러 전송 큐를 정의할 수 있습니다. 다른 클래스의 서비스에 사용 중인 각 전송 큐가 있는 동일한 목적지에 대해 여러 전송 큐를 정의할 수도 있습니다. 예를 들어, 동일한 목적지로 가는 소형 메시지와 대형 메시지에 각기 다른 전송 큐를 작성할 수 있습니다. 서로 다른 메시지 채널을 사용하여 메시지를 전송하면 대형 메시지로 인한 소형 메시지 지연이 발생하지 않습니다. 클러스터 토픽 또는 클러스터 큐에 대한 모든 메시지는 기본적으로 단일 클러스터 전송 큐 `SYSTEM.CLUSTER.TRANSMIT.QUEUE`에 위치합니다. 옵션으로서 기본값을 변경할 수 있으며, 서로 다른 클러스터 큐 관리자로 이동하는 메시지 트래픽을 서로 다른 클러스터 전송 큐로 분리할 수 있습니다. `DEFCLXQ`를 `CHANNEL`로 큐 관리자 속성을 설정하면 각 클러스터 송신자 채널은 별도의 클러스터 전송 큐를 작성합니다. 대안으로서, 사용할 클러스터-송신자 채널에 대해 클러스터 전송 큐를 수동으로 정의할 수 있습니다.

전송 큐는 메시지 채널 에이전트를 트리거하여 메시지를 전방으로 보낼 수 있습니다. 트리거를 사용한 IBM MQ 애플리케이션 시작을 참조하십시오.

z/OS IBM MQ for z/OS에서 그룹 내 큐잉을 사용하는 경우, 그룹 내 큐잉 에이전트가 전송 큐를 제공합니다. 공유 전송 큐는 IBM MQ for z/OS에서 그룹 내 큐잉을 사용할 때 사용됩니다.

이니시에이션 큐

이니시에이션 큐는 애플리케이션 큐에서 트리거 이벤트가 발생할 때 큐 관리자가 트리거 메시지를 넣는 로컬 큐입니다.

트리거 이벤트는 프로그램이 큐 처리를 시작하도록 하는 이벤트입니다. 예를 들어 이벤트는 10개가 넘는 수신 메시지가 될 수 있습니다. 트리거의 작동 방법에 대한 자세한 정보는 트리거를 사용한 IBM MQ 애플리케이션 시작을 참조하십시오.

데드-레터(미전달 메시지) 큐

데드-레터(미전달 메시지) 큐는 큐 관리자가 전달할 수 없는 메시지를 넣는 로컬 큐입니다.

큐 관리자가 메시지를 데드-레터 큐에 넣을 때 메시지에 헤더를 추가합니다. 헤더 정보에는 큐 관리자가 데드-레터 큐에 메시지를 넣는 이유가 포함됩니다. 또한 원래 메시지의 목적지, 큐 관리자가 데드-레터 큐에 메시지를 넣는 날짜 및 시간도 포함됩니다.

애플리케이션은 전달할 수 없는 메시지에 대한 큐를 사용할 수도 있습니다. 자세한 정보는 데드-레터(미전달 메시지) 큐 사용을 참조하십시오.

시스템 명령 큐

시스템 명령 큐는 적절하게 권한 부여된 애플리케이션이 IBM MQ 명령을 송신할 수 있는 큐입니다. 이러한 큐는 플랫폼의 지원에 따라 PCF, MQSC 및 CL 명령을 수신하며 이러한 명령에 대해 조치하는 큐 관리자에 대해 준비된 상태입니다.

z/OS IBM MQ for z/OS 에서는 큐를 SYSTEM.COMMAND.INPUT 라고 합니다. 다른 플랫폼에서는 SYSTEM.ADMIN.COMMAND.QUEUE라고 합니다. 승인된 명령은 플랫폼에 따라 다릅니다. 자세한 내용은 프로그래밍 가능 명령 형식 참조의 내용을 참조하십시오.

시스템 기본 큐

시스템 기본 큐에는 시스템에 대한 큐의 초기 정의가 포함됩니다. 큐 정의를 작성할 때, 큐 관리자는 적절한 시스템 기본 큐에서 정의를 복사합니다. 큐 정의 작성은 동적 큐 작성과 다릅니다. 동적 큐 정의는 동적 큐 템플리트로 선택한 모델 큐를 기준으로 합니다.

이벤트 큐
이벤트 큐는 이벤트 메시지를 보유합니다. 이러한 메시지는 큐 관리자 또는 채널이 보고합니다.

리모트 큐

프로그램에 있어서 큐가 다른 큐 관리자에 의해 소유되는 경우 큐는 프로그램이 연결되는 큐 관리자에 대해 리모트입니다.

통신 링크가 설정된 위치에서 프로그램은 리모트 큐로 메시지를 송신할 수 있습니다. 프로그램은 리모트 큐에서 메시지를 가져올 수 없습니다.

리모트 큐를 정의할 때 작성된 큐 정의 오브젝트는 로컬 큐 관리자가 메시지를 송신하려는 큐를 찾는 데 필요한 정보만을 보유합니다. 이 오브젝트를 리모트 큐의 로컬 정의라고 합니다. 리모트 큐의 모든 속성은 리모트 큐를 소유하는 큐 관리자가 보유하며 이는 리모트 큐가 큐 관리자에게는 로컬 큐이기 때문입니다.

리모트 큐를 열 때 이 큐를 식별하려면 다음 중 하나를 지정해야 합니다.

- 리모트 큐를 정의하는 로컬 정의의 이름 애플리케이션의 관점에서 이는 로컬 큐를 여는 것과 같습니다. 애플리케이션은 큐가 로컬 또는 리모트인지를 알 필요가 없습니다.

  IBM i를 제외한 모든 플랫폼에서 리모트 큐의 로컬 정의를 작성하려면 DEFINE QREMOTE 명령을 사용하십시오.

  IBM i IBM i에서 CRTMQMQ 명령을 사용하십시오.
- 해당 리모트 큐 관리자에게 알려진 대로의 리모트 큐 관리자의 이름 및 큐의 이름

리모트 큐의 로컬 정의에는 18 페이지의 『큐의 속성』에 설명된 공용 속성에 추가하여 세 가지 속성이 더 있습니다. 세 가지 속성은 다음과 같습니다.

RemoteQName
큐의 소유 큐 관리자가 알고 있는 이름입니다.

RemoteQMgrName
소유하고 있는 큐 관리자의 이름

XmitQName
메시지를 다른 큐 관리자에게 전달할 때 사용되는 로컬 전송 큐의 이름

이 속성에 대한 자세한 정보는 큐의 속성을 참조하십시오.

리모트 큐의 로컬 정의에 대하여 MQINQ 호출을 사용하는 경우 큐 관리자는 원격 시스템에서 일치하는 로컬 큐의 속성이 아닌 로컬 정의의 속성(리모트 큐 이름, 리모트 큐 관리자 이름 및 전송 큐 이름)만을 리턴합니다.

전송 큐의 내용도 참조하십시오.

알리어스 큐

알리어스 큐는 다른 큐 또는 토픽에 액세스하는 데 사용할 수 있는 IBM MQ 오브젝트입니다. 이는 둘 이상의 프로그램이 다른 이름을 사용하여 액세스하여 동일한 큐에 대해 작업할 수 있음을 의미합니다.

기본 큐라고 하는 알리어스 이름의 해석 결과인 큐 는 플랫폼에서 지원되는 다음과 같은 큐 유형 중 하나가 될 수 있습니다.

- 로컬 큐
- 리모트 큐의 로컬 정의
- z/OS 공유 큐는 IBM MQ for z/OS에서만 사용 가능한 로컬 큐의 유형입니다.
- 사전정의된 큐
- 동적 큐

알리어스 이름은 토픽으로 해석할 수도 있습니다. 현재 애플리케이션이 메시지를 큐에 넣고 있으면 큐 이름을 토픽의 알리어스로 작성하여 토픽으로 발행하도록 애플리케이션을 작성할 수 있습니다. 애플리케이션 코드의 변경은 필요하지 않습니다.

참고: 알리어스는 동일한 큐 관리자의 다른 알리어스로 직접 해석될 수 없습니다.

알리어스 큐 사용에 대한 예는 시스템 관리자가 기본 큐 이름(알리어스가 해석의 대상 큐) 및 다른 큐 이름에 다른 액세스 권한을 부여하는 것입니다. 이는 프로그램 또는 사용자에게 알리어스 큐 사용에 대한 권한은 부여되지만 기본 큐에 대한 권한은 부여될 수 없음을 의미합니다.

그렇지 않은 경우 알리어스 이름에 대해 Put 조작을 금지하지만 기본 큐에 대해서는 Put 조작을 허용하도록 권한 부여를 설정할 수 있습니다.

일부 애플리케이션에서 알리어스 큐 사용은 시스템 관리자가 애플리케이션을 변경할 필요 없이 알리어스 큐 오브젝트의 정의를 쉽게 변경할 수 있음을 의미합니다.

IBM MQ는 프로그램이 알리어스 이름을 사용하려고 할 때 이 이름에 대해 권한 검사를 수행합니다. 권한 검사에서는 알리어스가 해석하는 이름에 액세스하기 위한 권한이 프로그램에 부여되었는지 확인하지 않습니다. 따라서 프로그램은 해석된 큐 이름이 아닌 알리어스 큐 이름에 액세스할 수 있는 권한을 부여받을 수 있습니다.

18 페이지의 『큐』에 설명된 일반 큐 속성에 추가하여 알리어스 큐는 **BaseQName** 속성을 가질 수 있습니다. 알리어스 이름이 해석되는 기본 큐의 이름입니다. 이 속성에 대한 자세한 설명은 BaseQName (MQCHAR48)을 참조하십시오.

알리어스 큐의 *InhibitGet* 및 **InhibitPut** 속성(18 페이지의 『큐』 참조)은 알리어스 이름에 속합니다. 예를 들어, 알리어스 큐 이름 ALIAS1이 기본 큐 이름 BASE로 해석하는 경우 ALIAS1에 대한 금지가 ALIAS1에만 영향을 주며 BASE는 금지되지 않습니다. 하지만 BASE에 대한 금지는 ALIAS1에 영향을 줍니다.

또한 *DefPriority* 및 **DefPersistence** 속성도 알리어스 이름에 속합니다. 예를 들어 동일한 기본 큐의 다른 알리어스에 다른 기본 우선순위를 지정할 수 있습니다. 또한 알리어스를 사용하는 애플리케이션에 대한 변경없이 이러한 우선순위를 변경할 수 있습니다.

동적 및 모델 큐

이 정보는 동적 큐, 임시 및 영구적 동적 큐의 특성, 동적 큐의 사용, 동적 큐를 사용할 때의 고려사항 및 모델 큐에 대한 통찰을 제공합니다.

애플리케이션 프로그램이 MQOPEN 호출을 발행하여 모델 큐를 열 때 큐 관리자는 모델 큐와 같은 속성을 가진 로컬 큐의 인스턴스를 동적으로 작성합니다. 모델 큐의 *DefinitionType* 필드 값에 따라, 큐 관리자는 임시 또는 영구적 동적 큐를 작성합니다(동적 큐 작성 참조).

임시 동적 큐의 특성

임시 동적 큐에는 다음 특성이 있습니다.

- z/OS 이는 공유 큐일 수 없으며 큐 공유 그룹의 큐 관리자에서 액세스할 수 없습니다.

  참고로 큐 공유 그룹은 IBM MQ for z/OS에서만 사용 가능합니다.
- 비지속 메시지만 보유합니다.
- 복구 가능하지 않습니다.
- 큐 관리자가 시작될 때 삭제됩니다.
- 큐를 작성한 MQOPEN 호출을 발행한 애플리케이션이 큐를 닫거나 종료할 때 삭제됩니다.
  - 큐에 커미트된 메시지가 있는 경우 삭제됩니다.

- 이 때 큐에 대해 미해결된 커미트되지 않은 MQGET, MQPUT 또는 MQPUT1 호출이 있을 경우 이러한 호출이 커미트된 후에 큐는 논리적으로 삭제된 것으로 표시되며 닫기 처리의 일부로 또는 애플리케이션이 종료될 때에만 실제로 삭제됩니다.
- 이때 큐가 작성 또는 다른 애플리케이션에 의해 사용 중이면 큐가 논리적으로 삭제된 것으로 표시되며 큐를 사용하여 마지막 애플리케이션을 종료할 때에만 큐가 실제로 삭제됩니다.
- 논리적으로 삭제된 큐(큐를 닫은 경우는 제외)에 대한 액세스 시도가 이유 코드 MQRC_Q_DELETED로 실패했습니다.
- 큐를 작성한 해당 MQOPEN 호출에 대해 MQCO_NONE, MQCO_DELETE 및 MQCO_DELETE_PURGE가 MQCLOSE 호출에 지정되었을 때 이 호출 모두 MQCO_NONE으로 처리됩니다.

영구적 동적 큐의 특성

영구적 동적 큐에는 다음의 특성이 있습니다.

- 지속적 또는 비지속 메시지를 보유합니다.
- 시스템 실패 이벤트에서 복구 가능합니다.
- 애플리케이션(큐를 작성한 MQOPEN 호출 발행 애플리케이션일 필요는 없음)이 MQCO_DELETE 또는 MQCO_DELETE_PURGE 옵션을 사용하여 큐를 성공적으로 닫을 때 삭제됩니다.
  - 큐에 여전히 메시지(커미트된 또는 커미트되지 않은 메시지)가 있는 경우 MQCO_DELETE 옵션을 사용한 닫기 요청이 실패합니다. 큐에 커미트된 메시지(닫기의 일부로 삭제되는 메시지)가 있는 경우에도 MQCO_DELETE_PURGE 옵션을 사용한 닫기 요청은 성공하지만 큐에 대해 미해결된 커미트되지 않은 MQGET, MQPUT 또는 MQPUT1 호출이 있을 경우에는 실패합니다.
  - 삭제 요청이 성공했지만 큐가 사용 중(작성 또는 다른 애플리케이션에 의해)이면 이 큐는 논리적으로 삭제된 것으로 표시되며 큐를 사용하여 마지막 애플리케이션이 큐를 닫았을 때에만 실제로 삭제됩니다.
- 닫기 애플리케이션이 큐를 작성한 MQOPEN 호출을 발행하지 않는 한 큐 삭제에 대한 권한이 부여되지 않은 애플리케이션이 큐를 닫는 경우 삭제되지 않습니다. 해당하는 MQOPEN 호출을 유효성 검증하는 데 사용되는 사용자 ID(또는 MQOO_ALTERNATE_USER_AUTHORITY가 지정된 경우 대체 사용자 ID)에 대해 권한 검사를 수행합니다.
- 정상 큐와 같은 방식으로 삭제될 수 있습니다.

동적 큐의 사용

다음에 대해서 동적 큐를 사용할 수 있습니다.

- 애플리케이션이 종료된 후 보유될 큐를 요구하지 않는 애플리케이션입니다.
- 다른 애플리케이션이 처리하는 메시지에 대한 응답을 요구하는 애플리케이션입니다. 이런 애플리케이션은 모델 큐를 열어서 응답 대상 큐를 동적으로 작성할 수 있습니다. 예를 들어, 클라이언트 애플리케이션은 다음을 수행할 수 있습니다.
  1. 동적 큐를 작성하십시오.
  2. 요청 메시지의 메시지 디스크립터 구조의 **ReplyToQ** 필드에 동적 큐의 이름을 제공하십시오.
  3. 서버에 의해 처리되고 있는 큐에 요청을 배치하십시오.

서버는 응답 대상 큐에 응답 메시지를 배치할 수 있습니다. 마지막으로 클라이언트는 응답을 처리하고 삭제 옵션을 사용하여 응답 대상 큐를 닫습니다.

동적 큐 사용 시 고려사항

동적 큐 사용 시 다음 사항을 고려하십시오.

- 클라이언트 서버 모델에서 각 클라이언트는 고유 동적 응답 대상 큐를 작성하고 사용해야 합니다. 동적 응답 대상 큐가 둘 이상의 클라이언트 사이에서 공유되는 경우 응답 대상 큐 삭제가 지연될 수 있습니다. 이는 큐에 대해 미해결된 커미트되지 않은 활동이 있거나 큐가 다른 클라이언트에 의해 사용 중이기 때문입니다. 추가적으로 큐는 논리적으로 삭제된 것으로 표시될 수 있으며 후속 API 요청(MQCLOSE 제외)에 대해 액세스 가능하지 않을 수 있습니다.
- 애플리케이션 환경이 동적 큐가 애플리케이션 사이에서 공유되어야 함을 요구하는 경우 큐에 대한 모든 활동이 커미트될 때에만 큐가 닫히는지(삭제 옵션 사용) 확인하십시오. 이는 마지막 사용자에 의해 수행되어

야 합니다. 이 큐의 삭제가 지연되지 않는지 확인하고 논리적으로 삭제된 것으로 표시되어 큐에 액세스가 불가능한 시간을 최소화합니다.

모델 큐

모델 큐는 사용자가 동적 큐를 작성할 때 사용하는 큐 정의의 템플리트입니다.

큐 속성의 템플리트로 사용하려는 모델 큐에 이름을 지정하여 IBM MQ 프로그램에서 동적으로 로컬 큐를 작성할 수 있습니다. 이때 새 큐의 일부 속성을 변경할 수 있습니다. 그러나 **DefinitionType**은 변경할 수 없습니다. 예를 들어 영구적 큐가 필요한 경우 정의 유형이 영구적으로 설정된 모델 큐를 선택하십시오. 일부 대화식 애플리케이션은 응답을 처리한 후 이러한 큐를 유지보수할 필요가 없기 때문에 조회에 대한 응답을 보유하는 데 동적 큐를 사용할 수 있습니다.

모델 큐의 이름을 MQOPEN 호출의 오브젝트 디스크립터(MQOD)에 지정합니다. 모델 큐의 속성을 사용하여 큐 관리자는 로컬 큐를 동적으로 작성합니다.

동적 큐에 대한 전체 이름 또는 이름의 어간(예: ABC)을 지정하고 큐 관리자가 여기에 고유한 파트를 추가하도록 하거나 큐 관리자가 고유한 전체 이름을 지정하도록 할 수 있습니다. 큐 관리자가 이름을 지정하는 경우 이 이름을 MQOD 구조에 삽입합니다.

MQPUT1 호출을 모델 큐에 직접 발행할 수 없지만 MQPUT1을 모델 큐를 열어 작성된 동적 큐에 MQPUT1을 발행할 수 있습니다.

MQSET 및 MQINQ는 모델 큐에서 실행될 수 없습니다. MQOO_INQUIRE 또는 MQOO_SET를 포함하는 모델 큐를 열면 동적으로 작성된 큐에서 후속 MQINQ 및 MQSET 호출이 작성됩니다.

모델 큐의 속성은 로컬 큐의 속성 서브세트입니다. 자세한 설명은 큐의 속성을 참조하십시오.

*IBM MQ*에서 특정 목적에 사용되는 큐

IBM MQ에서는 그 조작과 관련된 특정 목적의 로컬 큐를 사용합니다.

IBM MQ에서 이러한 큐를 사용할 수 있도록 하려면 먼저 큐를 정의해야 합니다.

이니시에이션 큐

이니시에이션 큐는 트리거에서 사용되는 큐입니다. 큐 관리자는 트리거 이벤트가 발생할 때 트리거 메시지를 이니시에이션 큐에 넣습니다. 트리거 이벤트는 큐 관리자가 감지하는 조건의 논리적 조합입니다. 예를 들어, 큐의 메시지 수가 사전정의된 용량에 도달하면 트리거 이벤트가 생성될 수 있습니다. 이러한 이벤트는 큐 관리자가 지정된 이니시에이션 큐에 트리거 메시지를 넣는 원인이 됩니다. 이 트리거 메시지는 이니시에이션 큐를 모니터하는 특수 애플리케이션인 트리거 모니터로 검색할 수 있습니다. 트리거 모니터는 트리거 메시지에 지정된 애플리케이션 프로그램을 시작합니다.

큐 관리자가 트리거를 사용하는 경우 최소 하나 이상의 이니시에이션 큐가 해당 큐 관리자에 대해 정의되어야 합니다. 트리거할 오브젝트 관리, runmqtrm, 트리거를 사용하여 IBM MQ 애플리케이션 시작을 참조하십시오.

전송 큐

전송 큐는 리모트 큐 관리자를 목적지로 하는 메시지를 임시로 저장하는 큐입니다. 로컬 큐 관리자가 메시지를 직접 송신하는 각 리모트 큐 관리자에 대해 최소 하나 이상의 전송 큐를 정의해야 합니다. 이 큐는 원격 관리에서도 사용됩니다. 로컬 큐 관리자에서 원격 관리를 참조하십시오. 분산 큐잉에서 전송 큐의 사용에 대한 정보는 IBM MQ 분산 큐잉 기술을 참조하십시오.

각 큐 관리자에는 기본 전송 큐가 있을 수 있습니다. 클러스터에 포함되지 않는 큐 관리자가 메시지를 리모트 큐에 넣는 경우, 기본 조치는 기본 전송 큐를 사용하는 것입니다. 목적지 큐 관리자와 이름이 동일한 전송 큐가 있는 경우, 메시지는 해당 전송 큐에 배치됩니다. **RQMNAME** 매개변수가 목적지 큐 관리자와 일치하며 **XMITQ** 매개변수가 정의되어 있는 큐 관리자 알리어스 정의가 있는 경우, 메시지는 **XMITQ**로 이름 지정된 전송 큐에 배치됩니다. **XMITQ** 매개변수가 없으면 메시지가 해당 메시지에 이름 지정된 로컬 큐에 배치됩니다.

클러스터 전송 큐

클러스터 내의 각 큐 관리자에는 클러스터 전송 큐 SYSTEM.CLUSTER.TRANSMIT.QUEUE와 모델 클러스터 전송 큐 SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE가 있습니다. 이러한 큐의 정의는 큐 관리자를 정의할 때 기본적으로 작성됩니다. 큐 관리자 속성 **DEFCLXQ**를 CHANNEL로 설정한 경우 생성된 각 클러스터-송신자 채널에 대해 영구적인 동적 클러스터 전송 큐가 자동으로 생성됩니다. 큐를

SYSTEM.CLUSTER.TRANSMIT. *ChannelName*라고 합니다. 또한 클러스터 전송 큐를 수동으로 정의할 수도 있습니다.

클러스터에 포함되는 큐 관리자는 이러한 큐 중 하나에서 동일한 클러스터에 있는 다른 큐 관리자로 메시지를 송신합니다.

이름 해석 중에, 클러스터 전송 큐는 기본 전송 큐보다 우선하고 특정 클러스터 전송 큐는 SYSTEM.CLUSTER.TRANSMIT.QUEUE보다 우선합니다.

데드-레터 큐
데드-레터(미전달 메시지) 큐는 올바른 목적지로 라우트되지 못한 메시지를 저장하는 큐입니다. 예를 들어, 목적지 큐가 가득 찬 경우에는 메시지를 라우트할 수 없습니다. 제공된 데드-레터 큐를 SYSTEM.DEAD.LETTER.QUEUE라고 합니다.

분산 큐잉의 경우 포함된 각 큐 관리자에서 데드-레터 큐를 정의하십시오.

명령 큐
명령 큐인 SYSTEM.ADMIN.COMMAND.QUEUE는 적합하게 권한 부여된 애플리케이션이 처리를 위한 MQSC 명령을 보낼 수 있는 로컬 큐입니다. 이러한 명령은 명령 서버라고 하는 IBM MQ 컴포넌트로 검색됩니다. 명령 서버는 명령을 유효성 검증하고 큐 관리자에 의한 처리를 위해 올바른 명령을 전달하며 적절한 응답 대상 큐로 응답을 리턴합니다.

명령 큐는 해당 큐 관리자가 작성될 때 각 큐 관리자에 대해 자동으로 작성됩니다.

응답 대상 큐
애플리케이션이 요청 메시지를 송신하면 메시지를 수신하는 애플리케이션은 응답 메시지를 송신 애플리케이션으로 돌려 보낼 수 있습니다. 이 메시지를 큐(응답 대상 큐라고 함)에 넣고 이 큐는 일반적으로 송신 애플리케이션에 대한 로컬 큐입니다. 응답 대상 큐의 이름은 메시지 디스크립터의 일부로 송신 애플리케이션에 의해 지정됩니다.

이벤트 큐
도구 이벤트는 MQI 애플리케이션과는 독립적으로 큐 관리자를 모니터하는 데 사용할 수 있습니다.

도구 이벤트가 발생하면 큐 관리자는 이벤트 메시지를 이벤트 큐에 넣습니다. 이 메시지는 모니터링 애플리케이션이 읽을 수 있으며 이벤트가 문제점을 표시할 때 관리자에게 알리거나 일부 보수 조치를 시작할 수 있습니다.

참고: 트리거 이벤트는 도구 이벤트와 다릅니다. 트리거 이벤트는 동일한 조건에 의해 발생하지 않으며 이벤트 메시지를 생성하지 않습니다.

도구 이벤트에 대한 자세한 정보는 도구 이벤트를 참조하십시오.

큐 관리자

애플리케이션에 제공하는 큐 관리자 및 큐잉 서비스 소개입니다.

프로그램이 해당 큐 관리자의 서비스를 사용할 수 있으려면 큐 관리자에 대한 연결이 있어야 합니다. 프로그램은 이 연결을 MQCONN 또는 MQCONNX 호출을 사용하여 명시적으로 수행할 수 있거나 암묵적으로 수행할 수도 있습니다 (이는 프로그램이 실행되는 플랫폼 및 환경에 따라 다름).

IBM MQ 큐 관리자는 다음 조치를 확인합니다.

- 오브젝트 속성이 수신된 명령에 따라 변경됩니다.
- 적절한 조건에 부합되는 경우 트리거 이벤트 또는 도구 이벤트와 같은 특수 이벤트가 생성됩니다.
- MQPUT 호출을 작성하는 애플리케이션이 요청한 대로 올바른 큐에 메시지를 넣습니다. 이러한 조치가 수행될 수 없는 경우에는 애플리케이션에 알려지며 적절한 이유 코드가 제공됩니다.

각 큐는 단일 큐 관리자에 속하며 해당 큐 관리자에 대한 로컬 큐라고 합니다. 애플리케이션이 연결되는 큐 관리자는 해당 애플리케이션에 대한 로컬 큐 관리자라고 합니다. 애플리케이션의 경우 해당 로컬 큐 관리자에 속하는 큐는 로컬 큐입니다.

리모트 큐는 다른 큐 관리자에 속하는 큐입니다. 리모트 큐 관리자는 로컬 큐 관리자를 제외한 다른 모든 큐 관리자입니다. 리모트 큐 관리자는 네트워크 전체의 원격 시스템에 존재하거나 로컬 큐 관리자와 동일한 시스템에 존재할 수 있습니다. IBM MQ는 동일한 시스템에서 다중 큐 관리자를 지원합니다.

큐 관리자 오브젝트는 일부 MQI 호출에서 사용할 수 있습니다. 예를 들어, MQI 호출 MQINQ를 사용하여 큐 관리자 오브젝트의 속성에 대해 조회할 수 있습니다.

큐 관리자의 속성

각 큐 관리자에 연관된 속성(또는 특성) 세트는 큐 관리자의 특성을 정의합니다.큐 관리자의 일부 속성은 작성될 때 수정되며 그 외의 속성은 IBM MQ 명령을 사용하여 변경할 수 있습니다. TLS(Transport Layer Security) 암호화에 사용된 속성을 제외한 모든 속성 값에 대해서는 MQINQ 호출을 사용하여 조회할 수 있습니다.

고정된 속성에는 다음이 포함됩니다.

- 큐 관리자의 이름
- 큐 관리자가 실행되는 플랫폼(예: Windows)
- 큐 관리자가 지원하는 시스템 제어 명령의 레벨
- 큐 관리자에 의해 처리되는 메시지에 지정할 수 있는 최고 우선순위
- 프로그램이 IBM MQ 명령을 송신할 수 있는 큐의 이름
- 큐 관리자가 처리할 수 있는 최대 메시지 길이입니다 z/OS (IBM MQ for z/OS에서만 수정됨).
- 프로그램이 메시지를 넣고 가져올 때 큐 관리자가 동기점을 지원하는지 여부

변경 가능한 속성에는 다음이 포함됩니다.

- 큐 관리자에 대한 텍스트 설명
- 큐 관리자가 MQI 호출을 처리할 때 문자열에 사용하는 문자 세트의 ID
- 큐 관리자가 트리거 메시지의 수를 제한하는 데 사용하는 시간 간격
- z/OS 큐 관리자가 만료된 메시지에 대해 큐를 스캔하는 빈도를 판별하는 데 사용하는 시간 간격(IBM MQ for z/OS에만 해당됨)
- 큐 관리자의 데드-레터(미전달 메시지) 큐 이름
- 큐 관리자의 기본 전송 큐 이름
- 하나의 연결에 대한 최대 열린 핸들 수
- 다양한 범주의 이벤트 보고 사용 및 사용 안함
- 작업 단위 내에서 커미트되지 않은 메시지의 최대 수

큐 관리자 및 워크로드 관리

동일한 큐에 대해 둘 이상의 정의를 가지고 있는 큐 관리자의 클러스터를 설정할 수 있습니다(예를 들어, 클러스터에 있는 큐 관리자는 서로의 복제본일 수 있음). 특정 큐에 대한 메시지는 큐의 인스턴스를 호스팅하는 큐 관리자로 핸들링할 수 있습니다. 워크로드 관리 알고리즘은 메시지를 핸들링할 큐 관리자를 결정하여 큐 관리자 간에 워크로드를 분배합니다. 자세한 정보는 클러스터 워크로드 관리 알고리즘을 참조하십시오.

채널

분산 큐 관리자에서 사용되는 채널은 IBM MQ MQI client와 IBM MQ 서버 간 또는 두 IBM MQ 서버 간의 논리 통신 링크입니다.

채널은 한 큐 관리자에서 다른 큐 관리자로 메시지를 이동시키는 데 사용되며 기본 통신 프로토콜로부터 애플리케이션을 보호합니다. 큐 관리자들은 동일하거나 서로 다른 플랫폼에서 동일한 시스템 또는 다른 시스템에 존재할 수 있습니다. 전송되는 메시지는 여러 위치에서 생성할 수 있습니다.

- 하나의 노드에서 다른 노드로 데이터를 전송하는 사용자 작성 애플리케이션 프로그램.
- PCF 명령 또는 MQAI를 사용하는 사용자 작성 관리 애플리케이션.
- IBM MQ Explorer입니다.
- 도구 이벤트 메시지를 다른 큐 관리자에 전송하는 큐 관리자.
- 원격 관리 명령을 다른 큐 관리자에 전송하는 큐 관리자(예: MQSC 명령 또는 administrative REST API).

채널에는 두 가지 정의가 있으며 연결의 양 끝에 하나씩 있습니다. 큐 관리자가 서로 통신하기 위해서는 다른 큐 관리자로 메시지를 보낼 큐 관리자에서 하나의 채널 오브젝트를 정의하고 메시지를 받을 큐 관리자에서 보충 채널 오브젝트를 정의해야 합니다. 동일한 채널 이름이 연결의 양 끝에서 사용되어야 하며 사용된 채널 유형은 호환 가능해야 합니다.

IBM MQ에는 세 개의 채널 범주가 있으며, 이러한 범주 내에는 다른 채널 유형이 포함됩니다.

- 단방향이며, 한 큐 관리자에서 다른 큐 관리자로 메시지를 전송하는 메시지 채널입니다.
- IBM MQ MQI client에서 큐 관리자로 MQI 호출을 전송하고 큐 관리자에서 IBM MQ 클라이언트로 응답하는 양방향 MQI 채널입니다.
- AMQP 채널. 이는 양방향이며, AMQP 클라이언트를 큐 관리자 또는 서버 시스템에 연결합니다. IBM MQ에서는 AMQP 채널을 사용하여 AMQP 애플리케이션 및 큐 관리자 사이에서 AMQP 호출 및 응답을 전송합니다.

메시지 채널

메시지 채널의 용도는 하나의 큐 관리자에서 다른 큐 관리자로 메시지를 전송하는 것입니다. 클라이언트 서버 환경에서는 메시지 채널이 필요하지 않습니다.

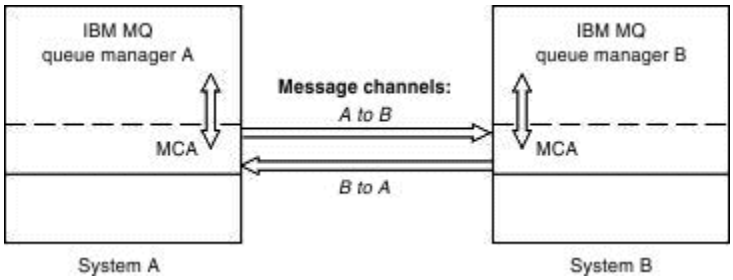


그림 2. 두 큐 관리자 간의 메시지 채널

메시지 채널은 단방향 링크입니다. 리모트 큐 관리자가 로컬 큐 관리자가 전송한 메시지에 응답하도록 하려는 경우 응답을 로컬 큐 관리자에 다시 전송하도록 두 번째 채널을 설정해야 합니다.

메시지 채널은 메시지 채널 에이전트(MCA)를 사용하여 두 큐 관리자에 연결됩니다. 채널의 각 끝에는 메시지 채널 에이전트가 있습니다. MCA가 다중 스레드를 사용하여 메시지를 전송하도록 할 수 있습니다. 이 프로세스는 파이프라이닝이라고 합니다. 파이프라이닝을 사용하면 MCA가 메시지를 좀 더 효율적으로 전송할 수 있으며 채널 성능도 향상됩니다. 파이프라이닝에 대한 자세한 정보는 채널 속성을 참조하십시오.

채널에 대한 자세한 정보는 채널 엑시트 호출 및 데이터 구조 및 41 페이지의 『분산 큐잉 컴포넌트』의 내용을 참조하십시오.

MQI 채널

MQI (Message Queue Interface) 채널은 IBM MQ MQI client 를 서버 시스템의 큐 관리자에 연결하며 IBM MQ MQI client 애플리케이션에서 MQCONN 또는 MQCONNX 호출을 발행할 때 설정됩니다.

이는 양방향 링크이고 MQI 호출 및 응답의 전송에만 사용되며, 여기에는 메시지 데이터가 포함된 MQPUT 호출 및 메시지 데이터의 리턴으로 귀결되는 MQGET 호출이 포함됩니다. 채널 정의를 작성하고 사용하는 다양한 방법이 있습니다(MQI 채널 정의 참조).

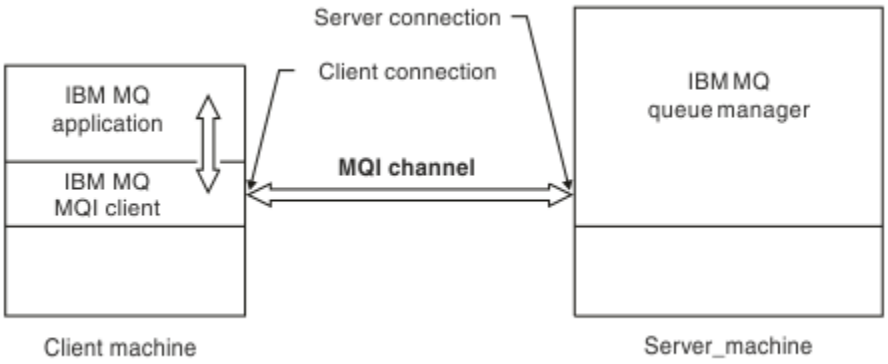


그림 *3. MQI* 채널에서의 클라이언트 연결 및 서버 연결

z/OS MQI 채널은 클라이언트를 단일 큐 관리자에 연결하거나 큐 공유 그룹의 일부인 큐 관리자에 연결하는 데 사용될 수 있습니다(큐 공유 그룹에 클라이언트 연결 참조).

MQI 채널 정의에는 두 가지 채널 유형이 있습니다. 양방향 MQI 채널을 정의합니다.

클라이언트 연결 채널
이 유형은 IBM MQ MQI client용입니다.

서버 연결 채널
이 유형은 IBM MQ MQI client 환경에서 실행 중인 IBM MQ 애플리케이션이 통신하는 큐 관리자를 실행하는 서버를 위한 것입니다.

AMQP 채널

Multi

AMQP 채널의 유형은 한 개뿐입니다.

이 채널을 사용하여 AMQP 메시징 애플리케이션을 큐 관리자와 연결하면 애플리케이션이 IBM MQ 애플리케이션과 메시지를 교환할 수 있습니다. AMQP 채널을 사용할 경우 MQ Light에서 애플리케이션을 개발한 다음 엔터프라이즈 애플리케이션으로 배치할 수 있으므로 IBM MQ에서 제공하는 엔터프라이즈 레벨 기능을 이용할 수 있습니다.

클라이언트 연결 채널

클라이언트 연결 채널은 IBM MQ MQI client에서 큐 관리자로의 통신 경로를 제공하는 오브젝트입니다.

클라이언트 연결 채널은 분산 큐잉에 사용되어 큐 관리자와 클라이언트 사이에 메시지를 이동시킵니다. 이 채널은 근본적인 통신 프로토콜로부터 애플리케이션을 숨깁니다. 클라이언트는 큐 관리자와 동일한 또는 다른 플랫폼에 존재할 수 있습니다.

채널 정의

각 채널 유형에 대한 설명은 29 페이지의 『채널 정의』의 내용을 참조하십시오.

관련 개념
38 페이지의 『분산 큐잉 및 클러스터』
분산 큐잉은 한 큐 관리자에서 다른 큐 관리자로 메시지를 송신하는 것을 의미합니다. 수신 큐 관리자는 동일 시스템 또는 다른 쪽 세계 또는 인접한 곳의 다른 시스템에 있을 수 있습니다. 로컬 큐 관리자와 동일한 플랫폼에서 실행 중이거나 IBM MQ에서 지원되는 플랫폼 중 하나에서 실행 중일 수 있습니다. 분산 큐잉 환경에서 모든 연결을 수동으로 정의하거나 클러스터를 작성하고 IBM MQ에서 자동으로 연결 세부사항 대부분을 정의하도록 설정할 수 있습니다.

MQI(Message Queue Interface) 개요
관련 태스크
원격 IBM MQ 오브젝트 관리

MQI 채널 중지
서버와 클라이언트 간의 연결 구성

관련 참조
채널 엑시트 호출 및 데이터 구조
31 페이지의 『통신』
IBM MQ MQI clients는 MQI 채널을 사용하여 서버와 통신합니다.

채널 정의

IBM MQ에서 사용하는 다양한 유형의 메시지 채널 및 MQI 채널을 설명하는 표입니다.

메시지 채널을 참조할 때 채널은 종종 채널 정의와 동의어로 사용됩니다. 일반적으로 문맥 상에서 양측이 있는 완전한 채널을 말하는지 한 측만 있는 채널 정의를 말하는지 알 수 있습니다.

메시지 채널

메시지 채널은 다음 유형 중 하나일 수 있습니다.

메시지 채널 정의 유형	설명
송신자	송신자 채널은 큐 관리자가 다른 큐 관리자에 메시지를 송신하는 데 사용하는 메시지 채널입니다. 이 송신자 채널을 사용하여 메시지를 송신하려면 다른 큐 관리자에 이 송신자 채널과 동일한 이름의 수신자 채널도 작성해야 합니다. "콜백" 메커니즘을 구현 중인 경우 또한 송신자 채널을 요청자 채널과 함께 사용할 수도 있습니다.
서버	서버 채널은 큐 관리자가 다른 큐 관리자에 메시지를 송신하는 데 사용하는 메시지 채널입니다. 이 서버 채널을 사용하여 메시지를 송신하려면 다른 큐 관리자에 이 서버 채널과 동일한 이름의 수신자 채널도 작성해야 합니다. 또한 요청자 채널과 함께 서버 채널을 사용할 수도 있습니다. 이 경우에 채널의 다른 쪽의 요청자 채널 정의는 서버 채널 정의를 시작하도록 요청합니다. 서버는 요청자에 메시지를 송신합니다. 서버가 파트너 채널의 연결 이름을 알고 있으면 통신을 시작할 수도 있습니다.
수신자	수신자 채널은 큐 관리자가 다른 큐 관리자로부터 메시지를 수신하는 데 사용하는 메시지 채널입니다. 수신자 채널을 사용하여 메시지를 수신하려면 다른 큐 관리자에 이 수신자 채널과 동일한 이름의 송신자 또는 서버 채널도 작성해야 합니다.
요청자	요청자 채널은 큐 관리자가 다른 큐 관리자로부터 메시지를 수신하는 데 사용하는 메시지 채널입니다. 요청자 채널은 리모트 측에 정의된 파트너 채널을 시작하도록 요청할 수 있습니다. 파트너 채널이 서버 채널인 경우 서버 채널은 시작 요청을 승인하고 서버 채널 정의에서 식별된 전송 큐에서 요청자 채널로 메시지를 전송하기 시작합니다. 파트너 채널이 송신자 채널인 경우 송신자 채널은 시작 요청을 승인하지만 요청자와의 연결을 닫습니다. 그런 다음, 송신자 채널이 시작되고 파트너 요청자 채널과의 세션을 협상하며, 송신자 채널 정의에서 식별된 전송 큐에서 메시지를 전송하기 시작합니다. 후자의 경우 특히 요청자 채널이 송신자 채널에 콜백을 요청하는 콜백 메커니즘에 대해 제공됩니다.

메시지 채널 정의 유형	설명
클러스터 송신자	클러스터 송신자(CLUSSDR) 채널 정의는 클러스터 큐 관리자가 전체 저장소 중 하나에 클러스터 정보를 송신할 수 있는 채널의 송신측을 정의합니다. 클러스터 송신자 채널은 큐 관리자의 상태에 대한 모든 변경사항을 저장소에 알리기 위해 사용됩니다(예: 큐의 추가 또는 제거). 메시지 전송에도 사용됩니다. 전체 저장소 큐 관리자 자체에는 서로 가리키는 클러스터-송신자 채널이 있습니다. 이를 사용하여 서로 클러스터 상태 변경사항을 통신할 수 있습니다. 큐 관리자의 CLUSSDR 채널 정의가 어떠한 전체 저장소를 가리키는지는 별로중요하지 않습니다. 초기 접속이 이루어진 후에는 추가적인 클러스터 큐 관리자 오브젝트가 필요에 따라 자동으로 정의됩니다. 따라서 큐 관리자가 클러스터 정보를 모든 전체 저장소로 송신하고 메시지를 모든 큐 관리자로 송신할 수 있습니다.
클러스터 수신자	클러스터 수신자(CLUSRCVR) 채널 정의는 클러스터 큐 관리자가 클러스터의 기타 큐 관리자로부터 메시지를 수신할 수 있는 채널의 수신측을 정의합니다. 클러스터 수신자 채널은 저장소를 목적지로 하는 클러스터-정보에 대한 정보를 전달할 수도 있습니다. 큐 관리자는 클러스터 수신자 채널을 정의하여 다른 클러스터 큐 관리자에게 자신이 메시지를 수신할 수 있음을 표시합니다. 클러스터 큐 관리자마다 최소 하나의 클러스터-수신자 채널이 필요합니다.

각 채널에 대해 채널의 각 측에 대한 채널 정의가 있도록 양측 모두를 정의해야 합니다. 채널의 양측은 호환 가능한 유형이어야 합니다.

다음과 같은 채널 정의의 결합이 있을 수 있습니다.

- 송신자-수신자
- 서버-수신자
- 요청자-서버
- 요청자-송신자(콜백)
- 클러스터-송신자-클러스터-수신자

메시지 채널 에이전트

작성한 각 채널 정의는 특정 큐 관리자에 속합니다. 큐 관리자는 같거나 다른 유형의 채널을 몇 개 가질 수 있습니다. 각 채널의 끝에는 메시지 채널 에이전트(MCA) 프로그램이 있습니다. 채널의 한쪽 끝에서 호출자 MCA는 트랜스미션 큐에서 메시지를 가져와 채널을 통해 송신합니다. 채널의 다른 쪽 끝에서 응답자 MCA는 메시지를 수신하여 리모트 큐 관리자에 전달합니다.

호출자 MCA는 송신자, 서버 또는 요청자 채널과 연관될 수 있습니다. 응답자 MCA는 메시지 채널의 모든 유형과 연관될 수 있습니다.

IBM MQ는 연결의 양측에서 다음 채널 유형 조합을 지원합니다.

호출자		메시지 플로우의 방향	응답자	
채널 유형	리스너가 필요합니까?		리스너가 필요합니까?	채널 유형
송신자	아니오	호출자에서 응답자로	예	수신자
서버	아니오	호출자에서 응답자로	예	수신자
서버	아니오	호출자에서 응답자로	예	요청자
요청자	아니오	응답자에서 호출자로	예	서버

호출자		메시지 플로우의 방향	응답자	
채널 유형	**리스너가 필요합니까?**		**리스너가 필요합니까?**	**채널 유형**
요청자	예	응답자에서 호출자로	예	송신자

MQI 채널

MQI 채널은 다음 중 하나의 유형일 수 있습니다.

MQI 채널 유형	설명
서버 연결	서버 연결 채널은 IBM MQ 서버에 IBM MQ 클라이언트를 연결하는 데 사용되는 양방향 MQI 채널입니다. 서버 연결 채널은 채널의 서버 측입니다.
클라이언트 연결	클라이언트 연결 채널은 IBM MQ 서버에 IBM MQ 클라이언트를 연결하는 데 사용되는 양방향 MQI 채널입니다. IBM MQ Explorer는 또한 클라이언트 연결을 사용하여 리모트 큐 관리자에 연결합니다. 클라이언트 연결 채널은 채널의 클라이언트 측입니다. 클라이언트 연결 채널을 작성하면 큐 관리자를 호스팅하는 컴퓨터에서 파일이 작성됩니다. IBM MQ 클라이언트 컴퓨터로 클라이언트-연결 파일을 복사해야 합니다.

Multi 다중 스레드 지원 - 파이프라이닝

선택적으로, 메시지 채널 에이전트(MCA)가 다중 스레드를 사용하여 메시지를 전송하도록 허용할 수 있습니다. 파이프라이닝이라고 하는 이 프로세스는 MCA가 보다 적은 대기 상태에서 보다 효율적으로 메시지를 전송하도록 함으로써 채널 성능을 향상시킵니다. 각 MCA는 최대 2개의 스레드로 제한됩니다.

qm.ini 파일에서 *PipeLineLength* 매개변수로 파이프라이닝을 제어합니다. 이 매개변수는 채널 스탠자에 추가됩니다.

참고: 파이프라이닝은 TCP/IP 채널에만 적용됩니다.

파이프라이닝을 사용할 때, 채널 양 측의 큐 관리자는 1보다 큰 *PipeLineLength*를 갖도록 구성되어야 합니다.

채널 엑시트 고려사항

파이프라이닝은 다음 이유로 일부 엑시트 프로그램이 실패하도록 할 수 있습니다.

- 엑시트가 직렬로 호출되지 않았을 수 있습니다.
- 엑시트를 다른 스레드에서 대신 호출했을 수 있습니다.

파이프라이닝을 사용하기 전에 엑시트 프로그램의 디자인을 검사하십시오.

- 엑시트는 실행의 모든 단계에서 다시 진입해야 합니다.
- MQI 호출 사용 시에는 다른 스레드로부터 엑시트가 호출될 때 동일한 MQI 핸들을 사용할 수 없음을 기억하십시오.

큐를 열고 엑시트의 모든 후속 호출에서 MQPUT 호출을 위해 이의 핸들을 사용하는 메시지 엑시트를 고려하십시오. 엑시트가 상이한 스레드에서 호출되므로 이는 파이프라이닝 모드에서 실패합니다. 이러한 실패를 피하려면, 각 스레드마다 큐 핸들을 유지하고 엑시트가 호출될 때마다 스레드 ID를 확인하십시오.

통신

IBM MQ MQI clients는 MQI 채널을 사용하여 서버와 통신합니다.

채널 정의는 IBM MQ MQI client 및 서버의 연결 종료 시에 둘 다 작성되어야 합니다. 채널 정의를 작성하는 방법은 MQI 채널 정의에 설명되어 있습니다.

가능한 전송 프로토콜은 다음 표에 표시됩니다.

표 *1. MQI* 채널용 전송 프로토콜				
클라이언트 플랫폼	**LU 6.2**	**TCP/IP**	**NetBIOS**	**SPX**
IBM i		예		
Linux AIX and Linux 시스템	예[1]	예		
Windows	예	예	예	예

참고:

1. Linux LU6.2는 다음 플랫폼에서 지원되지 않습니다.
   - Linux (POWER 플랫폼)
   - Linux (x86-64 플랫폼)
   - Linux (zSeries s390x 플랫폼)

전송 프로토콜 - IBM MQ MQI client 및 서버 플랫폼의 조합은 이러한 전송 프로토콜을 사용하는 IBM MQ MQI client 및 서버 플랫폼의 가능한 조합을 보여줍니다.

IBM MQ MQI client 의 IBM MQ 애플리케이션은 큐 관리자가 로컬인 경우와 동일한 방식으로 모든 MQI 호출을 사용할 수 있습니다. **MQCONN** 또는 **MQCONNX** 는 연결 핸들을 작성하여 IBM MQ 애플리케이션을 선택된 큐 관리자와 연관시킵니다. 해당 연결 핸들을 사용하는 다른 호출은 연결된 큐 관리자가 처리합니다. IBM MQ MQI client 통신은 연결 독립 및 시간 독립인 큐 관리자 간의 통신과는 반대로 클라이언트와 서버 간의 활성 연결을 요구합니다.

채널 정의를 사용하여 전송 프로토콜이 지정되며 애플리케이션에는 영향을 주지 않습니다. 예를 들어, Windows 애플리케이션은 TCP/IP로 하나의 큐 관리자에 연결되고 NetBIOS로 다른 큐 관리자에 연결할 수 있습니다.

성능 고려사항

사용하는 전송 프로토콜은 IBM MQ 클라이언트 및 서버 시스템에 영향을 미칠 수도 있습니다. 전송 속도가 느린 특정 상황에서는 IBM MQ 채널 압축을 사용할 수 있습니다.

IBM MQ 오브젝트 이름 지정

IBM MQ 오브젝트에 채택된 이름 지정 규칙은 오브젝트에 따라 달라집니다. IBM MQ에 대해 사용하는 시스템 및 사용자 ID의 이름 또한 일부 이름 지정 제한의 대상이 됩니다.

큐 관리자의 각 인스턴스는 그 이름으로 알려집니다. 한 큐 관리자가 제공된 메시지를 전송할 대상 큐 관리자를 분명하게 식별할 수 있도록 이 이름은 상호 연결된 큐 관리자의 네트워크 내에서 고유해야 합니다.

다른 유형 오브젝트의 경우 각 오브젝트는 오브젝트에 연관된 이름을 가지며 해당 이름으로 참조될 수 있습니다. 이러한 이름은 하나의 큐 관리자 및 오브젝트 유형에서 고유해야 합니다. 예를 들어, 동일한 이름을 가진 큐 및 프로세스를 가질 수 있지만 동일한 이름을 가진 두 개의 큐를 가질 수는 없습니다.

IBM MQ에서 이름은 최대 20자까지 가능한 채널의 경우를 제외하고 최대 48자까지 가질 수 있습니다. IBM MQ 오브젝트 이름 지정에 대한 자세한 정보는 33 페이지의 『IBM MQ 오브젝트 이름 지정 규칙』의 내용을 참조하십시오.

IBM MQ에 대해 사용하는 시스템 및 사용자 ID의 이름 또한 다음의 일부 이름 지정 제한의 대상이 됩니다.

- 시스템 이름에 공백이 포함되지 않도록 하십시오. IBM MQ는 공백을 포함하는 시스템 이름을 지원하지 않습니다. 이러한 시스템에서 IBM MQ를 설치할 경우 큐 관리자를 작성할 수 없습니다.
- IBM MQ 권한 부여의 경우, 사용자 ID 및 그룹의 이름은 20자를 초과할 수 없습니다(공백 사용 불가).
- Windows 클라이언트가 @ 문자가 포함된 사용자 ID(예: abc@d)로 실행 중인 경우 IBM MQ for Windows 서버는 IBM MQ MQI client의 연결을 지원하지 않습니다.

관련 개념

36 페이지의 『IBM MQ 파일 이름』

각 IBM MQ 큐 관리자, 큐, 프로세스 정의, 이름 목록, 채널, 클라이언트 연결 채널, 리스너, 서비스 및 인증 정보 오브젝트는 파일별로 표시됩니다. 오브젝트 이름은 꼭 올바른 파일 이름일 필요는 없기 때문에 큐 관리자는 필요한 위치에서 오브젝트 이름을 올바른 파일 이름으로 변환합니다.

관련 참조

33 페이지의 『IBM MQ 오브젝트 이름 지정 규칙』

IBM MQ 오브젝트 이름은 최대 길이를 갖고 있으며 대소문자가 구분됩니다. 모든 문자가 모든 오브젝트 유형에 대해 지원되지는 않으며, 많은 오브젝트가 이름의 고유성에 관한 규칙을 갖고 있습니다.

IBM MQ 오브젝트 이름 지정 규칙

IBM MQ 오브젝트 이름은 최대 길이를 갖고 있으며 대소문자가 구분됩니다. 모든 문자가 모든 오브젝트 유형에 대해 지원되지는 않으며, 많은 오브젝트가 이름의 고유성에 관한 규칙을 갖고 있습니다.

여러 가지 유형의 IBM MQ 오브젝트가 있으며, 각 유형의 오브젝트는 별도의 오브젝트 네임스페이스에 존재하기 때문에 모두가 동일한 이름을 가질 수 있습니다. 예를 들어 로컬 큐와 송신자 채널은 둘 다 동일한 이름을 가질 수 있습니다. 그러나 동일한 네임스페이스에서는 한 오브젝트가 다른 오브젝트와 동일한 이름을 가질 수 없습니다. 예를 들어 로컬 큐는 모델 큐와 동일한 이름을 가질 수 없으며, 송신자 채널은 수신자 채널과 동일한 이름을 가질 수 없습니다.

다음 IBM MQ 오브젝트는 별도의 오브젝트 네임스페이스에 존재합니다.

- 인증 정보
- 채널
- 클라이언트 채널
- 리스너
- 이름 목록
- 프로세스
- 큐
- 서비스
- 스토리지 클래스
- 구독
- 토픽

오브젝트 이름의 문자 길이

일반적으로 IBM MQ 오브젝트 이름의 길이는 최대 48자까지 가능합니다. 이 규칙은 다음 오브젝트에 적용됩니다.

- 인증 정보
- 클러스터
- 리스너
- 이름 목록
- 프로세스 정의
- 큐
- 큐 관리자
- 서비스
- 구독
- 토픽

제한사항은 다음과 같습니다.

1. z/OS z/OS 시스템에서 큐 관리자는 최대 4자여야 하며 대문자와 숫자 문자만으로 구성되어야 합니다.
2. 채널 오브젝트 이름 및 클라이언트 연결 채널 이름의 최대 길이는 20문자입니다. 채널에 대한 자세한 정보는 채널 정의를 참조하십시오.
3. 토픽 문자열은 최대 10240바이트일 수 있습니다. 모든 IBM MQ 오브젝트 이름은 대소문자가 구분됩니다.
4. 구독 이름은 최대 10240바이트일 수 있으며 공백을 포함할 수 있습니다.
5. 스토리지 클래스 이름의 최대 길이는 8자입니다.
6. CF 구조 이름의 최대 길이는 12자입니다.

오브젝트 이름의 문자

IBM MQ 오브젝트 이름에 대해 올바른 문자는 다음과 같습니다.

문자	제한사항
대문자 A - Z	• 없음
소문자 a - z	• MQSC 스크립트에서 소문자가 포함된 이름은 작은따옴표로 묶어야 합니다. 이렇게 하면 소문자가 대문자로 변경되지 않습니다. • EBCDIC 카타카나를 사용하는 시스템은 오브젝트 이름에서 소문자 a- z 문자를 사용할 수 없습니다. • z/OS z/OS 시스템에서 소문자를 사용할 때 제한이 있을 수 있습니다. 예를 들어 큐 관리자 이름은 소문자를 포함할 수 없습니다. • IBM i IBM i 시스템에서 CL 명령을 사용할 때 소문자가 포함된 이름은 작은따옴표로 묶어야 합니다. 이렇게 하면 소문자가 대문자로 변경되지 않습니다.
숫자 0 - 9	• 없음
마침표(.)	• 없음
밑줄(_)	• Multi 없음 • z/OS 선두 또는 후미 문자 밑줄을 갖는 이름은 IBM MQ for z/OS 조작 및 제어판에 의해 처리될 수 없기 때문에 사용을 피하십시오.
정방향 슬래시(/)	• Windows Windows 시스템에서는 큐 관리자 이름의 첫 번째 문자가 슬래시일 수 없습니다. • IBM i IBM i 시스템에서 CL 명령을 사용할 때 슬래시가 포함된 이름은 작은따옴표로 묶어야 합니다. • z/OS 없음

문자	제한사항
퍼센트 부호(%)	• ALW 없음 • z/OS RACF®를 IBM MQ for z/OS에 대한 외부 보안 관리자로 사용 중인 경우, 오브젝트 이름에서 %를 사용하지 마십시오. 이런 이름은 RACF 일반 프로파일이 사용될 때 보안 검사에 포함되지 않기 때문입니다. • IBM i IBM i 시스템에서 CL 명령을 사용할 때 퍼센트 기호가 포함된 이름은 작은따옴표로 묶어야 합니다.

오브젝트 이름의 문자에 관한 몇 가지 규칙이 있습니다.

1. 선두 문자 또는 임베드된 공백은 허용되지 않습니다.
2. 자국어(NL) 문자는 허용되지 않습니다.
3. 전체 필드 길이보다 작은 모든 이름은 오른쪽에 공백을 채울 수 있습니다. 큐 관리자가 리턴하는 모든 짧은 이름은 항상 오른쪽이 공백으로 채워집니다.

큐 이름

큐 이름은 다음 두 파트를 갖습니다.

- 큐 관리자의 이름
- 해당 큐 관리자에게 알려진 큐의 로컬 이름

큐 이름의 각 파트는 길이가 48자입니다.

로컬 큐를 참조하기 위해(공백 문자로 바꾸거나 선행 널 문자를 사용하여) 큐 관리자의 이름을 생략할 수 있습니다. 그러나 IBM MQ에 의해 프로그램으로 리턴되는 모든 큐 이름에 큐 관리자의 이름이 들어있습니다.

z/OS 큐 공유 그룹의 모든 큐 관리자에게 액세스할 수 있는 공유 큐는 동일한 큐 공유 그룹에 있는 임의의 공유되지 않는 로컬 큐와 동일한 이름을 가질 수 없습니다. 이 제한은 애플리케이션이 로컬 큐를 열려고 할 때 실수로 공유 큐를 열거나 반대 상황이 발생할 가능성을 피하게 합니다. 공유 큐 및 큐 공유 그룹은 IBM MQ for z/OS에서만 사용할 수 있습니다.

리모트 큐를 참조하려면 프로그램이 전체 큐 이름에 큐 관리자의 이름을 포함시켜야 하거나 리모트 큐의 로컬 정의가 있어야 합니다.

애플리케이션이 큐 이름을 사용할 때 해당 이름은 로컬 큐의 이름(또는 로컬 큐에 대한 알리어스) 또는 리모트 큐의 로컬 정의의 이름일 수 있지만, 큐에서 메시지를 가져와야 하는 경우가 아니면(큐가 로컬이어야 함) 애플리케이션이 어느 것이든 알 필요는 없습니다. 애플리케이션이 큐 오브젝트를 열 때 MQOPEN 호출이 이름 해석 기능을 수행하여 후속 조작을 수행할 큐를 판별합니다. 이것의 중요성은 애플리케이션이 큐 관리자 네트워크의 특정 위치에서 정의되는 특정 큐에 대한 내장된 종속성을 갖지 않는다는 점입니다. 그러므로 시스템 관리자가 네트워크에서 큐의 위치를 변경하고 해당 정의를 변경하는 경우, 해당 큐를 사용하는 애플리케이션이 변경될 필요가 없습니다.

예약 오브젝트 이름

SYSTEM. 로 시작하는 오브젝트 이름은 큐 관리자가 정의한 오브젝트에 예약되어 있습니다. **Alter**, **Define**및 **Replace** 명령을 사용하여 설치에 맞게 이러한 오브젝트 정의를 변경할 수 있습니다. IBM MQ에 사용하도록 정의된 이름은 큐 이름에 모두 나열되어 있습니다.

z/OS IBM MQ for z/OS에서 커플링 기능 애플리케이션 구조 이름 CSQSYSAPPL은 예약되어 있습니다.

관련 개념

AIX, Linux, and Windows에서 설치 이름

IBM MQ 파일 이름

각 IBM MQ 큐 관리자, 큐, 프로세스 정의, 이름 목록, 채널, 클라이언트 연결 채널, 리스너, 서비스 및 인증 정보 오브젝트는 파일별로 표시됩니다. 오브젝트 이름은 꼭 올바른 파일 이름일 필요는 없기 때문에 큐 관리자는 필요한 위치에서 오브젝트 이름을 올바른 파일 이름으로 변환합니다.

큐 관리자 디렉토리에 대한 기본 경로는 다음과 같습니다.

- 접두부(IBM MQ 구성 정보에 정의되어 있음)
  - Linux AIX AIX and Linux에서 기본 접두부는 `/var/mqm`입니다. 기본 접두부는 mqs.ini 구성 파일의 `DefaultPrefix` 스탠자에서 구성됩니다.
  - Windows Windows 32비트 시스템에서 기본 접두부는 `C:\Program Files (x86)\IBM\WebSphere MQ`입니다. Windows 64비트 시스템에서 기본 접두부는 `C:\Program Files\IBM\MQ`입니다. 32비트와 64비트 설치 모두 데이터 디렉토리는 씨:`\ProgramData \IBM \MQ`에 설치됩니다. 기본 접두부는 mqs.ini 구성 파일의 `DefaultPrefix` 스탠자에서 구성됩니다.

  사용 가능한 경우 접두부는 IBM MQ 탐색기의 IBM MQ 특성 페이지를 사용하여 변경할 수 있으며, 그렇지 않은 경우 `mqs.ini` 구성 파일을 수동으로 편집하십시오.
- 큐 관리자 이름이 올바른 디렉토리 이름으로 변환됩니다. 예를 들어, 다음 큐 관리자의 경우

```
queue.manager
```

  다음과 같이 표시됩니다.

```
queue!manager
```

  이 프로세스를 이름 변환이라고 합니다.

IBM MQ에서 최대 48자를 포함하는 이름을 큐 관리자에 지정할 수 있습니다.

예를 들어, 큐 관리자의 이름을 다음과 같이 지정할 수 있습니다.

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

하지만 각 큐 관리자는 파일로 표시되며 파일 이름의 최대 길이 및 이름에 사용될 수 있는 문자 수에 대한 제한이 있습니다. 결과적으로 오브젝트를 나타내는 파일의 이름이 자동으로 변환되어 파일 시스템의 요구사항을 충족시킵니다.

큐 관리자 이름의 변환을 제어하는 규칙은 다음과 같습니다.

1. 개별 문자 변환:
   - .에서 !
   - /에서 &로
2. 이름이 여전히 올바르지 않은 경우에는 다음을 수행합니다.
   a. 여덟개 문자로 이름을 자릅니다.
   b. 세 숫자 접미어를 첨부합니다.

예를 들어, 기본 접두부 및 큐 관리자의 이름을 `queue.manager`로 가정합니다.

- Windows NTFS 또는 FAT32를 사용하는 Windows에서 큐 관리자 이름은 다음이 됩니다.

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

- Windows FAT를 사용하는 Windows에서 큐 관리자 이름은 다음이 됩니다.

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

- Linux AIX AIX and Linux에서 큐 관리자 이름은 다음이 됩니다.

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

변환 알고리즘은 또한 대소문자를 구분하지 않는 파일 시스템에서만 달라지는 이름을 구별합니다.

오브젝트 이름 변환

오브젝트 이름이 올바른 파일 시스템 이름일 필요는 없습니다. 오브젝트 이름을 변환해야 합니다. 사용된 메소드는 큐 관리자 이름의 메소드와 다릅니다. 각 시스템에 소수의 큐 관리자 이름만 있더라도 각 큐 관리자에 대해 많은 수의 다른 오브젝트가 있을 수 있기 때문입니다. 큐, 프로세스 정의, 이름 목록, 채널, 클라이언트 연결 채널, 리스너, 서비스 및 인증 정보 오브젝트는 파일 시스템에 표시됩니다.

변환 프로세스가 새 이름을 생성할 때 원래 오브젝트 이름에 대한 단순 관계가 없습니다. **dspmqfls** 명령을 사용하여 실제와 변환된 오브젝트 이름 사이에서 변환할 수 있습니다.

관련 참조

dspmqfls(파일 이름 표시)

관련 정보

mqs.ini 파일의 AllQueueManagers 스탠자

IBM i IBM i의 오브젝트 이름

큐 관리자는 고유한 이름이 있는 연관된 큐 관리자 라이브러리를 가집니다. IBM i IFS(Integrated File System)의 요구사항을 충족하기 위해 큐 관리자 이름 및 오브젝트 이름을 변환해야 할 수도 있습니다.

큐 관리자를 작성할 때, IBM MQ는 큐 관리자 라이브러리를 큐 관리자와 연관시킵니다. 이 큐 관리자 라이브러리는 사용자 정의 큐 관리자 이름을 기반으로 길이가 10자 미만인 고유 이름이 제공됩니다. 큐 관리자 및 큐 관리자 라이브러리는 접두부가 /QIBM/UserData/mqm인 큐 관리자 이름을 기반으로 하는 디렉토리에 놓입니다. 큐 관리자, 큐 관리자 라이브러리 및 디렉토리의 예제는 다음과 같습니다.

큐 관리자 이름	ORANGE
큐 관리자 라이브러리 이름	QMORANGE
디렉토리	/QIBM/UserData/mqm/ORANGE

모든 큐 관리자 이름 및 큐 관리자 라이브러리 이름은 /QIBM/UserData/mqm/mqs.ini 파일의 스탠자에 작성됩니다.

IBM MQ IFS 디렉토리 및 파일

IBM i IFS(Integrated File System)는 데이터를 저장하기 위해 IBM MQ에서 광범위하게 사용됩니다. IFS에 관한 자세한 정보는 *IFS(Integrated File System)* 소개를 참조하십시오.

각 IBM MQ 오브젝트(예: 채널 또는 큐 관리자)는 파일로 표시됩니다. 오브젝트 이름은 꼭 올바른 파일 이름일 필요는 없기 때문에 큐 관리자는 필요한 위치에서 오브젝트 이름을 올바른 파일 이름으로 변환합니다.

큐 관리자 디렉토리 경로의 형식은 다음과 같습니다.

- 큐 관리자 구성 파일 qm.ini에 정의된 접두부. 기본 접두부는 /QIBM/UserData/mqm입니다.
- 리터럴, qmgrs.
- 코드화된 큐 관리자 이름, 올바른 디렉토리 이름으로 변환된 큐 관리자 이름입니다. 예를 들어, 큐 관리자 queue/manager는 queue&manager로 표시됩니다.

이 프로세스를 이름 변환이라고 합니다.

IFS 큐 관리자 이름 변환

IBM MQ에서 최대 48자를 포함하는 이름을 큐 관리자에 지정할 수 있습니다.

예를 들어, 큐 관리자 QUEUE/MANAGER/ACCOUNTING/SERVICES의 이름을 지정할 수 있습니다. 각 큐 관리자에 대해 라이브러리가 작성되는 것과 같은 방식으로, 각 큐 관리자도 파일로 표시됩니다. EBCDIC의 변형 코드 포인트 때문에, 이름에서 사용될 수 있는 문자에 제한사항이 있습니다. 결과적으로, 오브젝트를 표시하는 IFS 파일의 이름은 파일 시스템의 요구사항을 충족하도록 자동으로 변환됩니다.

이름이 queue/manager인 큐 관리자의 예제를 사용하고, / 문자를 &로 변환하고, 기본 접두부를 가정하여 IBM MQ for IBM i의 큐 관리자 이름은 /QIBM/UserData/mqm/qmgrs/queue&manager가 됩니다.

오브젝트 이름 변환

오브젝트 이름은 필수적으로 올바른 파일 시스템 이름이 아니므로 오브젝트 이름을 변환해야 할 수 있습니다. 각 시스템에 대해서는 큐 관리자 이름이 몇 개 없지만 각 큐 관리자에 대해서는 다른 오브젝트가 많을 수 있기 때문에 사용된 메소드는 큐 관리자 이름의 메소드와 다릅니다. 프로세스 정의, 큐 및 이름 목록만이 파일 시스템에 표시됩니다. 채널은 이 고려사항에 영향을 받지 않습니다.

변환 프로세스가 새 이름을 생성할 때 원래 오브젝트 이름에 대한 단순 관계가 없습니다. DSPMQMOBJN 명령을 사용하여 IBM MQ 오브젝트로 변환된 이름을 볼 수 있습니다.

분산 큐잉 및 클러스터

분산 큐잉은 한 큐 관리자에서 다른 큐 관리자로 메시지를 송신하는 것을 의미합니다. 수신 큐 관리자는 동일 시스템 또는 다른 쪽 세계 또는 인접한 곳의 다른 시스템에 있을 수 있습니다. 로컬 큐 관리자와 동일한 플랫폼에서 실행 중이거나 IBM MQ에서 지원되는 플랫폼 중 하나에서 실행 중일 수 있습니다. 분산 큐잉 환경에서 모든 연결을 수동으로 정의하거나 클러스터를 작성하고 IBM MQ에서 자동으로 연결 세부사항 대부분을 정의하도록 설정할 수 있습니다.

분산 큐잉

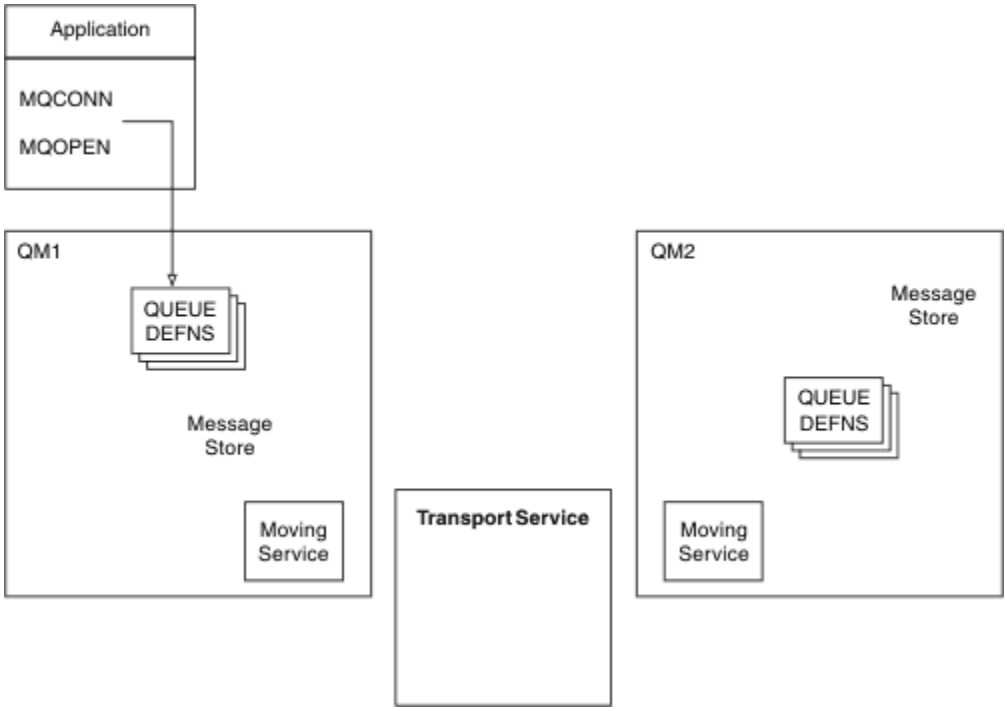


그림 *4.* 분산 큐잉의 컴포넌트 개요

이전 그림에서:

- 애플리케이션은 MQCONN 호출을 사용하여 큐 관리자에 연결합니다. 애플리케이션은 MQOPEN 호출을 사용하여 큐를 열어 메시지를 큐에 넣을 수 있습니다.
- 각 큐 관리자에는 각 큐에 대한 정의가 있습니다. 로컬 큐(이 큐 관리자에 의해 호스트됨)의 정의, 리모트 큐(다른 큐 관리자에 의해 호스트됨)의 정의가 있을 수 있습니다.
- 메시지가 리모트 큐를 대상으로 하면, 리모트 큐 관리자에 메시지를 전달할 준비가 될 때까지 로컬 큐 관리자를 메시지 저장소에서 지속되는 전송 큐에 보유합니다.
- 각 큐 관리자는 큐 관리자가 다른 큐 관리자와 통신하는 데 사용하는 이동 서비스라는 통신 소프트웨어를 포함합니다.
- 전송 서비스는 큐 관리자와 무관하며 다음 중 하나일 수 있습니다(플랫폼에 따라).
  - SNA APPC(Systems Network Architecture Advanced Program-to Program Communication)
  - TCP/IP(Transmission Control Protocol/Internet Protocol)
  - NetBIOS(Network Basic Input/Output System)
  - SPX(Sequenced Packet Exchange)

메시지를 송신해야 하는 컴포넌트

메시지를 리모트 큐 관리자에 전송하려는 경우, 로컬 큐 관리자는 전송 큐 및 채널에 대한 정의가 필요합니다. 채널은 두 큐 관리자 간의 단방향 통신 링크입니다. 리모트 큐 관리자의 임의의 수의 큐를 대상으로 하는 메시지를 전달할 수 있습니다.

각 채널 끝에는 송신 끝 또는 수신 끝과 같이 이를 정의하는 별도의 정의가 있습니다. 단순 채널은 로컬 큐 관리자의 송신자 채널 정의 및 리모트 큐 관리자의 수신자 채널 정의로 구성됩니다. 이 두 정의는 이름이 동일해야 하며 함께 하나의 채널을 구성해야 합니다.

메시지 송신 및 수신을 핸들링하는 소프트웨어는 메시지 채널 에이전트(MCA)라고 합니다. 채널의 각 끝에 메시지 채널 에이전트(MCA)가 있습니다.

각 큐 관리자에 데드-레터 큐(미전달 메시지 큐라고도 하는)가 있어야 합니다. 목적지에 메시지를 전달할 수 없는 경우 이 큐에 메시지를 넣습니다.

다음 그림은 큐 관리자, 전송 큐, 채널 및 MCA 간의 관계를 표시합니다.

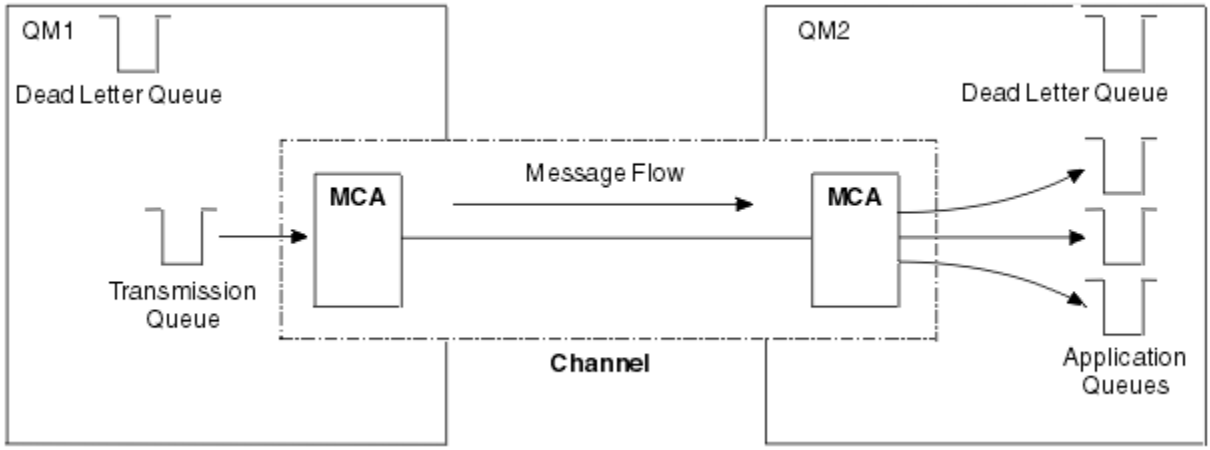


그림 *5.* 메시지 송신

메시지를 리턴해야 하는 컴포넌트

애플리케이션이 메시지를 리모트 큐 관리자에서 리턴해야 하는 경우, 다음 그림에 표시된 대로 큐 관리자 간에 반대 방향으로 실행하도록 다른 채널을 정의해야 합니다.

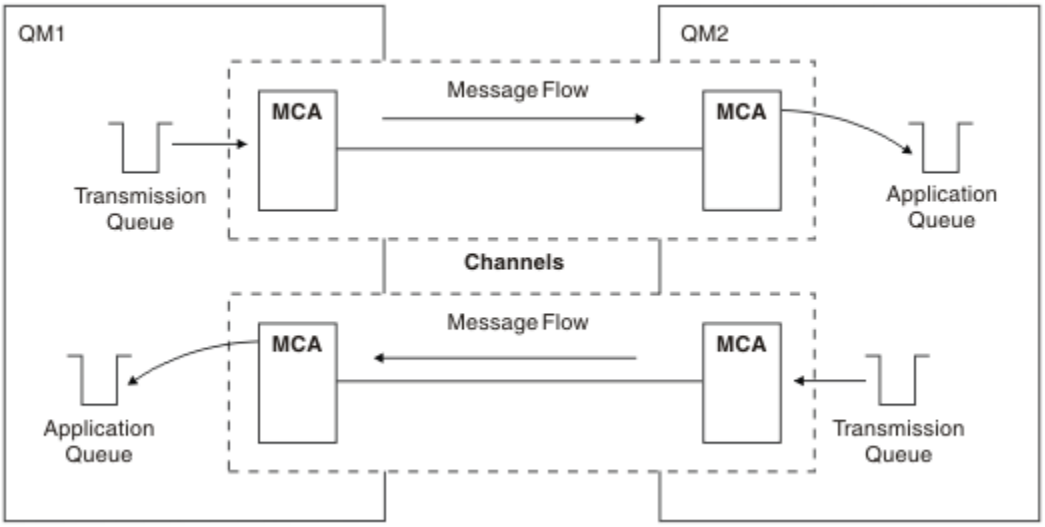


그림 6. 양방향으로 메시지 송신

클러스터

분산 큐잉 환경에서 모든 연결을 수동으로 정의하는 대신 클러스터의 큐 관리자 세트를 그룹화할 수 있습니다. 이를 수행하는 경우, 큐 관리자는 명확한 채널 정의, 리모트 큐 정의 또는 각 목적지에 대한 전송 큐가 필요없이 호스팅하는 큐를 클러스터의 다른 큐 관리자에서 사용할 수 있도록 설정할 수 있습니다. 클러스터의 모든 큐 관리자에는 클러스터의 다른 큐 관리자에 메시지를 전송하는 단일 전송 큐가 있습니다. 각 큐 관리자에 대해 하나의 클러스터-수신자 채널 및 하나의 클러스터-송신자 채널만 정의해야 합니다. 추가 채널은 클러스터가 자동으로 관리합니다.

IBM MQ 클라이언트는 임의의 다른 큐 관리자에게 연결할 수 있는 것처럼 클러스터의 일부인 큐 관리자에 연결할 수 있습니다. 수동으로 구성된 분산 큐잉과 마찬가지로 MQPUT 호출을 사용하여 메시지를 큐 관리자에 있는 큐에 넣습니다. MQGET 호출을 사용하여 로컬 큐에서 메시지를 검색합니다.

클러스터를 지원하는 플랫폼의 큐 관리자는 클러스터의 파트일 필요가 없습니다. 클러스터를 사용하는 대신 분산 큐잉도 수동으로 계속 구성할 수 있습니다.

클러스터 사용의 이점

클러스터링은 다음과 같은 두 가지 주요 이점을 제공합니다.

- 클러스터는 보통 구성할 채널, 전송 큐, 리모트 큐에 대한 많은 오브젝트 정의를 필요로 하는 IBM MQ 네트워크 관리를 단순화합니다. 이 상황은 특히 많은 큐 관리자를 상호 연결해야 하는, 잠재적으로 계속 변화하는 대규모 네트워크에서 적용됩니다. 이 아키텍처는 특히 구성 및 활동적인 유지보수가 어렵습니다.
- 클러스터는 큐 및 클러스터의 큐 관리자에서 메시지 트래픽의 워크로드를 분산하는 데 사용할 수 있습니다. 이러한 분산을 통해 단일 큐의 메시지 워크로드를 여러 큐 관리자에 있는 해당 큐의 동등한 인스턴스에 분산할 수 있습니다. 이러한 워크로드 분산을 사용하여 시스템에서 특히 활성 메시지 플로우의 확장 성능을 향상시키고 시스템 장애에 대해 더 많은 회복 기능을 달성할 수 있습니다. 이러한 환경에서 분산된 큐의 각 인스턴스는 메시지를 처리하는 이용 애플리케이션을 보유합니다. 자세한 정보는 워크로드 관리를 위한 클러스터 사용을 참조하십시오.

메시지가 클러스터에서 라우팅되는 방식

클러스터를 성실한 시스템 관리자가 유지보수하는 큐 관리자의 네트워크로 생각할 수 있습니다. 사용자가 클러스터 큐를 정의할 때마다 시스템 관리자가 자동으로 다른 큐 관리자에서 필요한 대로 대응하는 리모트 큐 정의를 작성합니다.

IBM MQ가 클러스터의 각 큐 관리자에 대한 전송 큐를 제공하기 때문에 사용자가 전송 큐 정의를 작성할 필요가 없습니다. 이 단일 전송 큐를 사용하여 클러스터의 다른 모든 큐 관리자로 메시지를 전달할 수 있습니다. 단일 전송 큐를 사용하도록 제한되지는 않습니다. 큐 관리자는 다중 전송 큐를 사용하여 클러스터의 각 큐 관리자로 전달되는 메시지를 구분할 수 있습니다. 일반적으로 큐 관리자는 단일 클러스터 전송 큐를 사용합니다. 큐 관리자에서

클러스터의 각 큐 관리자마다 각기 다른 클러스터 전송 큐를 사용할 수 있도록 큐 관리자 속성 DEFCLXQ를 변경할 수 있습니다. 또한 클러스터 전송 큐를 수동으로 정의할 수도 있습니다.

클러스터에 조인하는 모든 큐 관리자는 이 방식으로 작업하는 데 동의합니다. 자기 자신 및 자신이 호스팅하는 큐에 대한 정보를 송신하고, 클러스터의 다른 멤버에 관한 정보를 수신합니다.

큐 관리자를 사용할 수 없게 되면 정보를 잃게 되도록 하기 위해 클러스터에서 두 개의 큐 관리자를 전체 저장소의 역할을 하도록 지정합니다. 이러한 큐 관리자는 클러스터의 모든 큐 관리자 및 큐에 대한 전체 정보 세트를 저장합니다. 클러스터의 다른 모든 큐 관리자는 이러한 큐 관리자 및 메시지를 교환하는 큐에 대한 정보만 저장합니다. 이러한 큐 관리자를 부분 저장소라고 합니다. 자세한 정보는 50 페이지의 『클러스터 저장소』의 내용을 참조하십시오.

클러스터의 파트가 되기 위해서는 큐 관리자가 클러스터-송신자 채널과 클러스터-수신자 채널이라는 두 채널을 가져야 합니다.

- 클러스터-송신자 채널은 송신자 채널 같은 통신 채널입니다. 이미 클러스터의 멤버인 전체 저장소에 연결하기 위해 수동으로 큐 관리자에 하나의 클러스터-송신자 채널을 작성해야 합니다.
- 클러스터-수신자 채널은 수신자 채널 같은 통신 채널입니다. 하나의 클러스터-수신자 채널을 수동으로 작성해야 합니다. 클러스터 통신을 수신하기 위해 채널이 큐 관리자에 대한 메커니즘으로 작용합니다.

그러면 이 큐 관리자와 클러스터의 다른 멤버 사이에서 통신에 필요한 다른 모든 채널이 자동으로 작성됩니다.

다음 그림은 CLUSTER라는 클러스터의 컴포넌트를 표시합니다.

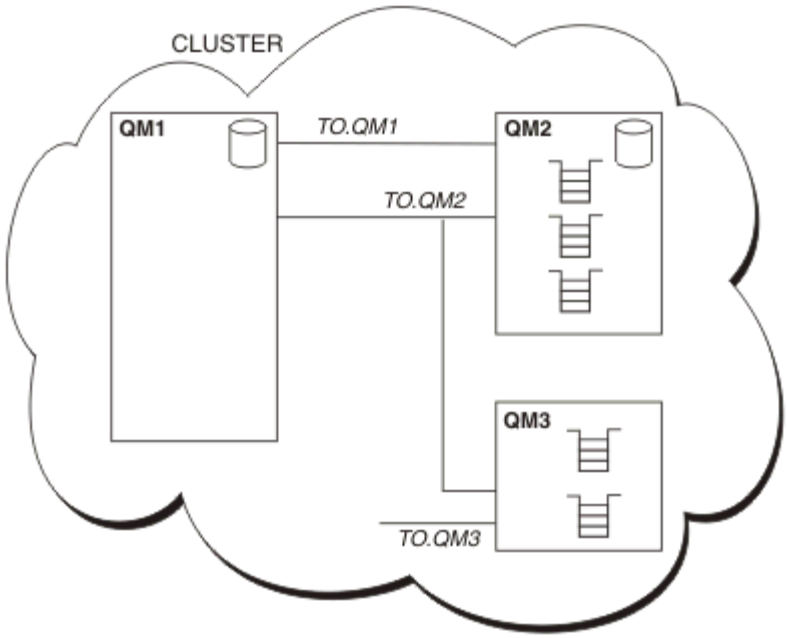


그림 7. 큐 관리자의 클러스터

- CLUSTER에는 세 개의 큐 관리자(QM1, QM2 및 QM3)가 포함되어 있습니다.
- QM1 및 QM2는 클러스터의 큐 및 큐 관리자에 관한 정보의 전체 저장소를 호스트합니다.
- QM2 및 QM3은 일부 클러스터 큐, 즉 클러스터의 다른 큐 관리자에 액세스 가능한 큐를 호스트합니다.
- 각 큐 관리자에는 메시지를 수신할 수 있는 TO.qmgr이라는 클러스터-수신자 채널이 있습니다.
- 각 큐 관리자에는 또한 정보를 저장소 큐 관리자 중의 하나로 송신할 수 있는 클러스터-송신자 채널이 있습니다.
- QM1 및 QM3은 QM2에 있는 저장소에 송신하며 QM2는 QM1에 있는 저장소에 송신합니다.

분산 큐잉 컴포넌트

분산 큐잉의 컴포넌트로는 메시지 채널, 메시지 채널 에이전트, 전송 큐, 채널 시작기 및 리스너, 채널 엑시트 프로그램이 있습니다. 메시지 채널의 각 끝에 대한 정의는 다음 유형 중 하나일 수 있습니다.

메시지 채널은 하나의 큐 관리자에서 다른 큐 관리자로 메시지를 이동시키는 채널입니다. MQI 채널과 메시지 채널을 혼동하지 마십시오. 두 가지 유형의 MQI 채널(서버 연결(SVRCONN) 및 클라이언트 연결(CLNTCONN))이 있습니다. 자세한 정보는 채널을 참조하십시오.

메시지 채널의 각 끝의 정의는 다음 유형 중 하나일 수 있습니다.

- 송신자(SDR)
- 수신자(RCVR)
- 서버(SVR)
- 요청자(RQSTR)
- 클러스터 송신자(CLUSSDR)
- 클러스터 수신자(CLUSRCVR)

메시지 채널은 한 끝에 정의된 다음 유형 중 하나 및 다른 끝의 호환 가능한 유형을 사용하여 정의됩니다. 가능은 결합은 다음과 같습니다.

- 송신자-수신자
- 요청자-서버
- 요청자-송신자(콜백)
- 서버-수신자
- 클러스터 송신자-클러스터 수신자

송신자-수신자 채널의 작성에 대한 자세한 지시사항은 채널 정의에 포함되어 있습니다. 송신자-수신자 채널을 설정하기 위해 필요한 매개변수의 예는 사용자 플랫폼에 적용 가능한 구성 정보 예를 참조하십시오. 모든 유형의 채널을 정의하는 데 필요한 매개변수는 DEFINE CHANNEL을 참조하십시오.

송신자-수신자 채널

한 시스템의 송신자는 채널을 시작하여 메시지를 다른 시스템에 송신할 수 있습니다 송신자는 채널의 다른 끝에 있는 수신자가 시작되기를 요청합니다. 송신자는 메시지를 전송 큐에서 수신자에게 송신합니다. 수신자는 메시지를 목적지 큐에 넣습니다. 42 페이지의 그림 8은 이 내용을 설명합니다.

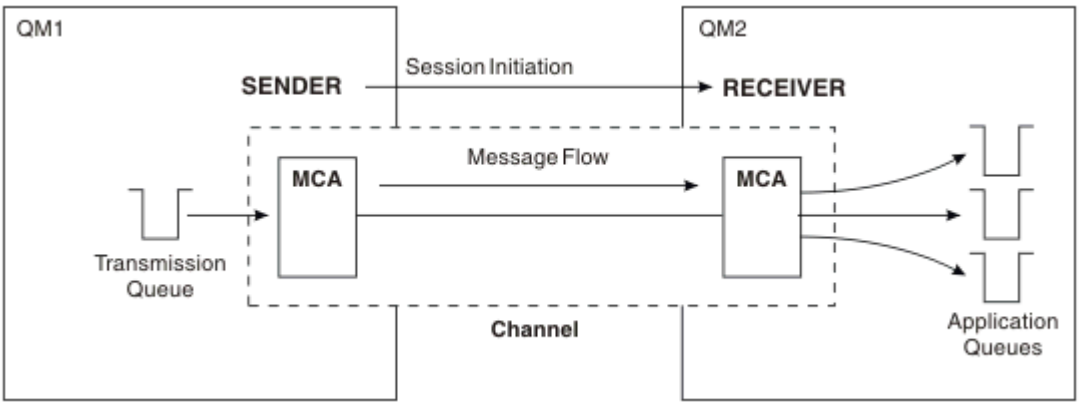


그림 *8.* 송신자-수신자 채널

요청자-서버 채널

한 시스템의 요청자는 채널을 시작하여 메시지를 다른 시스템에 수신할 수 있습니다 요청자는 채널의 다른 끝에 있는 서버가 시작되기를 요청합니다. 서버는 채널 정의에서 정의된 전송 큐에서 요청자에게 메시지를 송신합니다.

서버 채널은 통신을 시작하여 메시지를 요청자에게도 송신할 수 있습니다. 채널 정의에서 지정된 파트너의 연결 이름이 있는 서버 채널인 완전한 서버에만 적용됩니다. 완전한 서버는 요청자에서 시작되거나 요청자와의 통신을 시작할 수 있습니다.

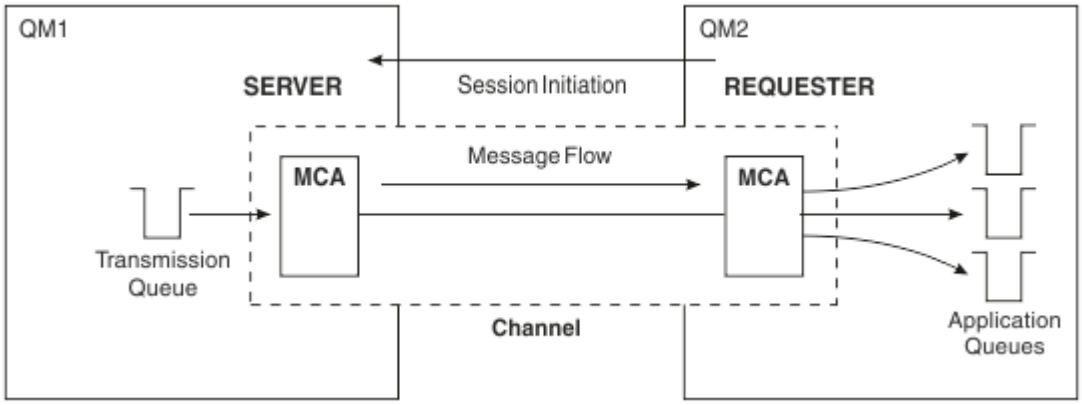


그림 *9.* 요청자-서버 채널

요청자-송신자 채널

요청자는 채널을 시작하고 송신자는 호출을 종료합니다. 그런 다음 송신자는 채널 정의(콜백이라는)의 정보에 따라 통신을 다시 시작합니다. 전송 큐에서 요청자에게 메시지를 송신합니다.

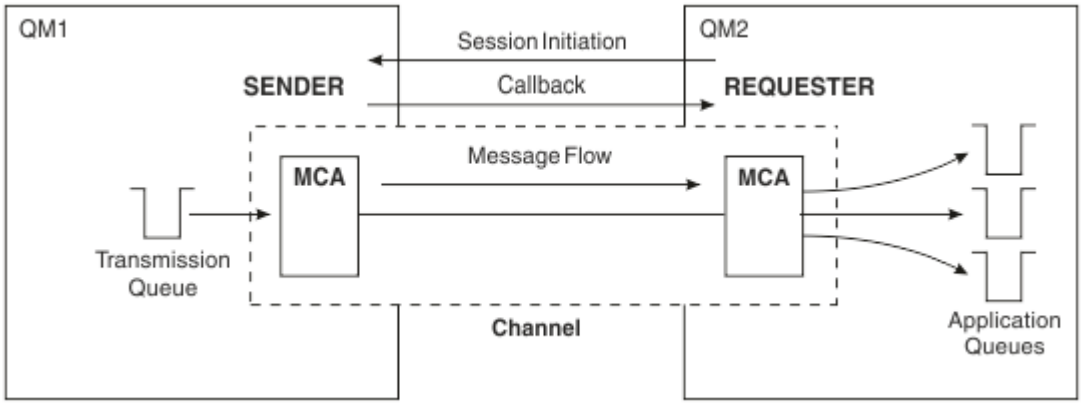


그림 *10.* 요청자-송신자 채널

서버-수신자 채널

송신자-수신자와 같지만 채널 정의에서 지정된 파트너의 연결 이름이 있는 서버 채널인 완전한 서버에만 적용됩니다. 채널 시동은 링크의 서버 끝에서 시작되어야 합니다. 이 그림은 42 페이지의 그림 8과 같습니다.

클러스터-송신자 채널

클러스터에서, 각 큐 관리자에는 클러스터 정보를 전체 저장소 큐 관리자 중 하나로 송신할 수 있는 클러스터-송신자 채널이 있습니다. 큐 관리자는 메시지를 클러스터-송신자 채널의 다른 큐 관리자에게도 송신할 수 있습니다.

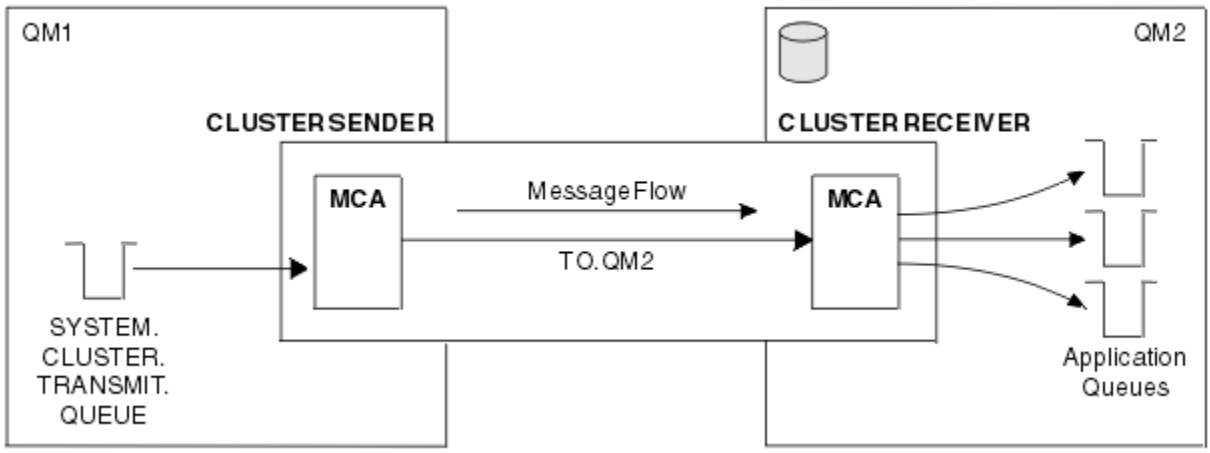


그림 *11.* 클러스터 송신자 채널

클러스터-수신자 채널

클러스터에는, 각 큐 관리자에 클러스터에 관한 정보 및 메시지를 수신할 수 있는 클러스터-수신자 채널이 있습니다. 이 그림은 44 페이지의 그림 11과 같습니다.

데드-레터 큐

데드-레터 큐(또는 미전달 메시지)는 올바른 목적지로 라우트할 수 없는 경우 메시지를 송신하는 큐입니다. 일반적으로, 각 큐 관리자에는 데드-레터 큐가 있습니다.

미전달 메시지 큐라고도 하는 데드-레터 큐(DLQ)는 예를 들어 큐가 존재하지 않거나 가득 차서 목적지 큐로 전달할 수 없는 메시지에 대한 보유 큐입니다. 데드-레터 큐가 데이터 변환 오류에 대한 채널 종료 송신에도 사용됩니다. 네트워크의 모든 큐 관리자는 일반적으로 데드-레터 큐로 사용할 로컬 큐를 가지므로 올바른 목적지로 전달할 수 없는 메시지를 추후 검색을 위해 저장할 수 있습니다.

메시지는 큐 관리자, 메시지 채널 에이전트(MCA) 및 애플리케이션에 의해 DLQ에 넣어질 수 있습니다. DLQ에 대한 모든 메시지는 데드 레터 헤더 구조, MQDLH를 접두부로 사용합니다. MQDLH 구조의 *Reason* 필드에는 메시지가 DLQ에 있는 이유를 식별하는 이유 코드가 포함됩니다.

일반적으로, 각 큐 관리자에 대해 데드-레터 큐를 정의해야 합니다. 이를 고려하지 않고 MCA가 메시지를 넣을 수 없으면, 전송 큐에 남게 되며 채널이 중지됩니다. 또한 빠른 비지속 메시지(빠른 비지속 메시지 참조)를 전달할 수 없으며 데드-레터 큐가 대상 시스템에 없으면 이 메시지는 제거됩니다.

그러나, 데드-레터 큐 사용은 메시지를 전달한 순서에 영향을 줄 수 있어 이를 사용하지 않도록 선택할 수 있습니다.

관련 태스크

데드-레터 큐에 대한 작업

미배달 메시지 문제점 해결

관련 참조

runmqdlq(데드-레터 큐 핸들러 실행)

리모트 큐 정의

리모트 큐 정의는 다른 큐 관리자에서 소유하는 큐에 대한 정의입니다.

애플리케이션이 로컬 큐에서만 메시지를 검색할 수 있는 반면, 로컬 큐나 리모트 큐에 메시지를 넣을 수 있습니다. 그러므로, 각 로컬 큐에 대한 정의뿐 아니라 큐 관리자에는 리모트 큐 정의가 있을 수 있습니다. 리모트 큐 정의의 장점은 리모트 큐나 리모트 큐 관리자의 이름, 또는 전송 큐의 이름을 지정하지 않고 애플리케이션을 사용하여 메시지를 리모트 큐에 넣을 수 있다는 것입니다. 리모트 큐 정의는 위치 독립성을 제공합니다.

리모트 큐 정의를 위한 다른 사용이 있으며, 이는 나중에 설명됩니다.

리모트 큐 관리자로 가져오는 방법

각 소스 및 대상 큐 관리자 간에 하나의 채널이 항상 없을 수 있습니다. 두 관리자 간의 링크 방법으로 멀티호핑, 채널 공유, 다른 채널 사용 및 클러스터링을 포함한 여러 다른 방법이 있습니다.

멀티홉

소스 큐 관리자와 대상 큐 관리자 간에 직접적인 통신 링크가 없는 경우, 대상 큐 관리자로 가는 중에 하나 이상의 중간 큐 관리자를 통해 전달할 수 있습니다. 이를 멀티홉이라고 합니다.

모든 큐 관리자와 중간 큐 관리자의 전송 큐 사이에 채널을 정의해야 합니다. 45 페이지의 그림 12에 설명되어 있습니다.

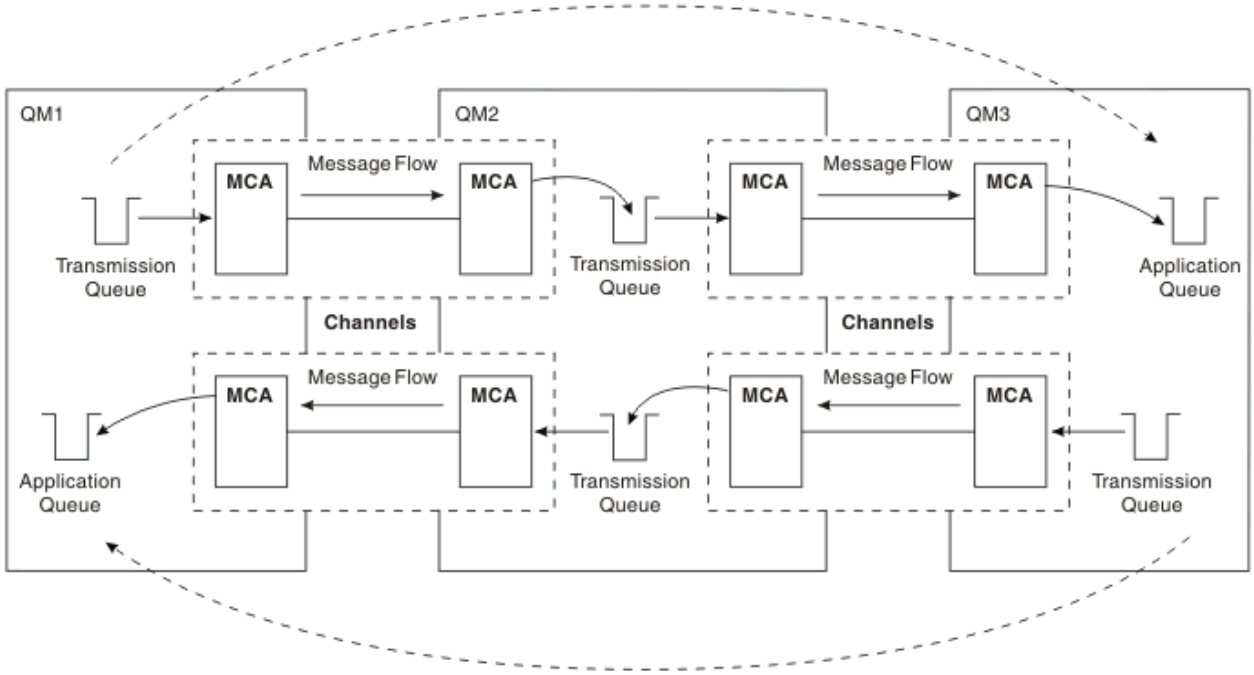


그림 *12.* 중간 큐 관리자를 통해 전달

채널 공유

애플리케이션 설계자로서 애플리케이션이 큐 이름과 함께 리모트 큐 관리자 이름을 지정하도록 강제 실행하거나 각 리모트 큐에 대해 리모트 큐 정의를 작성하도록 선택할 수 있습니다. 이 정의에는 리모트 큐 관리자 이름, 큐 이름 및 전송 큐의 이름이 있습니다. 이런 방식으로, 동일한 리모트 위치에 큐의 주소를 지정하는 모든 애플리케이션의 모든 메시지에는 동일한 전송 큐를 통해 전송된 메시지가 있습니다. 45 페이지의 그림 13에 설명되어 있습니다.

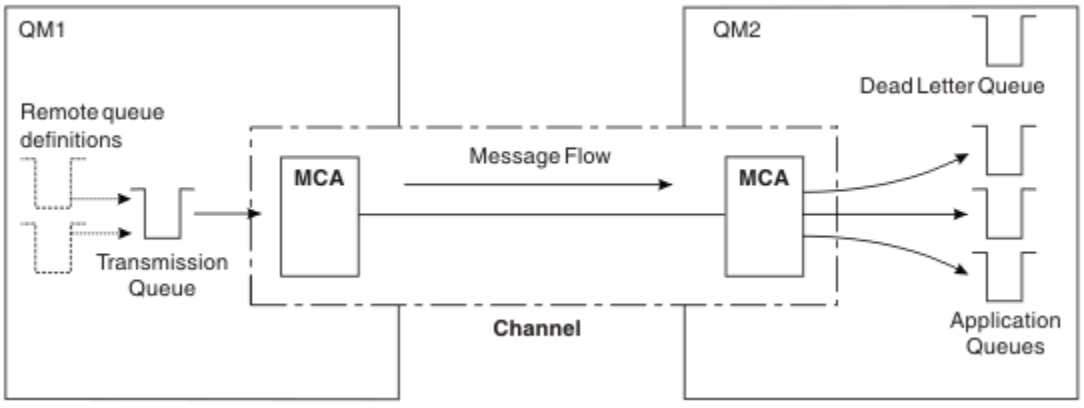


그림 *13.* 전송 큐 공유

45 페이지의 그림 13는 메시지가 다중 애플리케이션에서 다중 리모트 큐까지 동일 채널을 사용할 수 있음을 설명합니다.

다른 채널 사용

두 큐 관리자 간에 송신될 다른 유형의 메시지가 있는 경우 두 관리자 간에 둘 이상의 채널을 정의할 수 있습니다. 아마도 보안 용도로, 대량의 메시지 트래픽에 대한 전달 속도의 균형을 유지하기 위해 대체 채널이 필요한 경우가 있습니다.

다른 채널 및 다른 전송 큐를 정의하기 위해 필요한 두 번째 채널을 설정하려면 위치를 지정하는 리모트 큐 정의 및 전송 큐 이름을 작성하십시오. 그런 다음 애플리케이션은 채널을 사용할 수 있지만 메시지는 여전히 동일한 대상 큐에 전달됩니다. 46 페이지의 그림 14에 설명되어 있습니다.

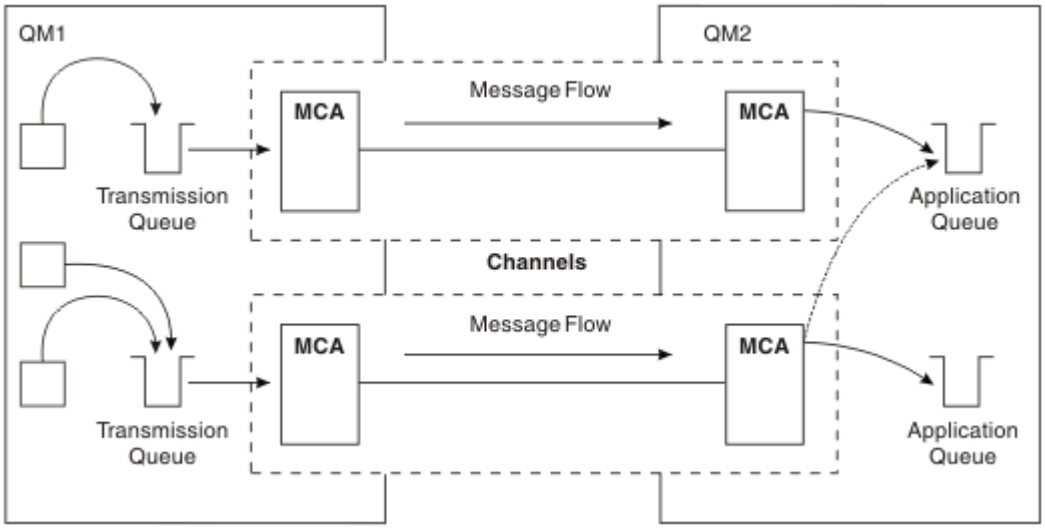


그림 *14.* 다중 채널 사용

리모트 큐 정의를 사용하여 전송 큐를 지정하는 경우 애플리케이션은 위치(즉, 목적지 큐 관리자) 자체를 지정하지 **않아야** 합니다. 이렇게 하는 경우, 큐 관리자는 리모트 큐 정의를 사용하지 않습니다. 리모트 큐 정의는 위치 독립성을 제공합니다. 애플리케이션은 큐가 있는 위치를 알지 않아도 메시지를 논리 큐에 넣을 수 있으며 애플리케이션을 변경하지 않고도 물리적 큐를 변경할 수 있습니다.

클러스터링 사용

클러스터 내의 모든 큐 관리자는 클러스터 수신자 채널을 정의합니다. 다른 큐 관리자가 메시지를 큐 관리자에 보내려는 경우, 해당 클러스터 송신자 채널을 자동으로 정의합니다. 예를 들어, 클러스터에 둘 이상의 큐 인스턴스가 있는 경우 클러스터 송신자 채널은 큐를 호스팅하는 큐 관리자에 정의될 수 있습니다. IBM MQ는 라운드 로빈 루틴을 사용하여 메시지를 라우트할 사용 가능한 큐 관리자를 선택하는 워크로드 관리 알고리즘을 사용합니다. 자세한 정보는 클러스터를 참조하십시오.

주소 지정 정보

애플리케이션이 리모트 큐 관리자에 대해 목적지인 메시지를 넣으면, 로컬 큐 관리자는 전송 큐에 넣기 전에 전송 헤더를 추가합니다. 이 헤더에는 목적지 큐와 큐 관리자의 이름, 즉 주소 지정 정보가 포함됩니다.

단일 큐 관리자 환경에서 애플리케이션이 메시지를 넣을 큐를 열 때 목적지 큐의 주소가 설정됩니다. 목적지 큐가 동일한 큐 관리자에 있기 때문에, 주소 정보에는 필요하지 않습니다.

분산 큐잉 환경에서 큐 관리자는 목적지 큐 이름뿐만 아니라 해당 큐의 위치(즉, 큐 관리자 이름) 및 원격 위치에 대한 라우트(즉, 전송 큐)를 알아야 합니다. 이 주소 지정 정보는 전송 헤더에 포함되어 있습니다. 수신 채널은 전송 헤더를 제거하며 전송 헤더의 정보를 사용하여 목적지 큐를 찾습니다.

리모트 큐 정의를 사용하는 경우 애플리케이션이 목적지 큐 관리자의 이름을 지정하지 않아도 됩니다. 이 정의는 리모트 큐의 이름, 메시지의 대상인 리모트 큐 관리자의 이름 및 메시지를 전송하는 데 사용되는 전송 큐의 이름을 지정합니다.

알리어스의 개념

알리어스는 메시지 서비스 품질을 제공하는 데 사용됩니다. 애플리케이션을 변경하지 않고도 큐 관리자 알리어스를 사용하여 시스템 관리자가 대상 큐 관리자의 이름을 변경할 수 있습니다. 또한 시스템 관리자가 라우트를 목적지 큐 관리자로 변경하거나 다른 여러 큐 관리자를 통한 전달(멀티호핑)과 관련된 라우트를 설정할 수 있습니다. 응답 대상 큐 알리어스는 응답을 위한 서비스 품질을 제공합니다.

큐 관리자 알리어스 및 응답 대상 큐 알리어스는 공백 RNAME이 있는 리모트 큐 정의를 사용하여 작성됩니다. 이 정의는 실제 큐를 정의하지 않습니다. 큐 관리자가 이 정의를 사용하여 물리적인 큐 이름, 큐 관리자 이름 및 전송 큐를 해석합니다.

알리어스 정의는 공백 RNAME이 있다는 특징을 가집니다.

큐 이름 해석

큐가 열릴 때마다 모든 큐 관리자에서 큐 이름 해석이 이루어집니다. 대상 큐, 대상 큐 관리자(로컬일 수 있음) 및 큐 관리자에 대한 라우트(널일 수 있음)를 식별하는 것이 목적입니다. 해석된 이름에는 큐 관리자 이름, 큐 이름 및 큐 관리자가 리모트인 경우 전송 큐의 세 부분이 있습니다.

리모트 큐 정의가 존재하면 알리어스 정의를 참조하지 않습니다. 애플리케이션에서 제공되는 큐 이름은 리모트 큐 정의에서 지정된 전송 큐, 리모트 큐 관리자 및 목적지 큐의 이름으로 해석됩니다. 큐 이름 해석에 대한 자세한 정보는 큐 이름 해석을 참조하십시오.

리모트 큐 정의가 없고 큐 관리자 이름이 지정되어 있거나 이름 서비스로 해석된 경우, 큐 관리자는 제공된 큐 관리자 이름과 일치하는 큐 관리자 알리어스 정의가 있는지 보고 확인합니다. 있는 경우, 정의 내의 정보를 사용하여 큐 관리자 이름을 목적지 큐 관리자의 이름으로 해석합니다. 큐 관리자 알리어스 정의는 목적지 큐 관리자에 대한 전송 큐를 판별하는 데도 사용될 수 있습니다.

해석된 큐 이름이 로컬 큐가 아닌 경우, 큐 관리자 이름 및 큐 이름 모두 애플리케이션이 전송 큐에 넣은 각 메시지의 전송 헤더에 포함됩니다.

리모트 큐 정의 또는 큐 관리자 알리어스 정의가 변경되지 않는 한 일반적으로 사용된 전송 큐의 이름은 해석된 큐 관리자와 동일합니다. 이러한 전송 큐를 정의하지 않았지만 기본 전송 큐를 정의한 경우 이 이름이 사용됩니다.

z/OS z/OS에서 실행 중인 큐 관리자의 이름은 4자로 제한됩니다.

큐 관리자 알리어스 정의

큐를 열어 메시지를 넣은 애플리케이션이 큐 이름 **및** 큐 관리자 이름을 지정하는 경우 큐 관리자 알리어스 정의가 적용됩니다.

큐 관리자 알리어스 정의는 다음과 같이 3가지로 사용됩니다.

- 메시지 송신 시, 큐 관리자 이름 다시 맵핑
- 메시지 송신 시, 전송 큐 변경 또는 지정
- 메시지 수신 시, 로컬 큐 관리자가 해당 메시지에 대해 의도된 목적지인지 여부 판별

아웃바운드 메시지 - 큐 관리자 이름 다시 맵핑

큐 관리자 알리어스 정의를 사용하여 MQOPEN 호출에서 지정된 큐 관리자 이름을 다시 맵핑할 수 있습니다. 예를 들어, MQOPEN 호출은 THISQ의 큐 이름 및 YOURQM의 큐 관리자 이름을 지정합니다. 로컬 큐 관리자에 다음 예제와 같은 큐 관리자 알리어스 정의가 있습니다.

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

애플리케이션이 메시지를 큐 관리자 YOURQM에 넣을 때 사용할 실제 큐 관리자가 REALQM임을 표시합니다. 로컬 큐 관리자가 REALQM인 경우, 메시지를 로컬 큐인 THISQ 큐에 넣습니다. 로컬 큐 관리자가 REALQM이 아닌 경우, REALQM이라는 전송 큐로 메시지를 라우트합니다. 큐 관리자는 YOURQM 대신 REALQM을 알리도록 전송 헤더를 변경합니다.

아웃바운드 메시지 - 전송 큐 대체 또는 지정

48 페이지의 그림 15에서는 큐 관리자 QM3에 큐 이름을 표시하는 전송 헤더가 있는 큐 관리자 QM1에 메시지가 도달하는 시나리오를 보여줍니다. 이 시나리오에서 QM3는 QM2를 통해 멀티호핑하여 도달할 수 있습니다.

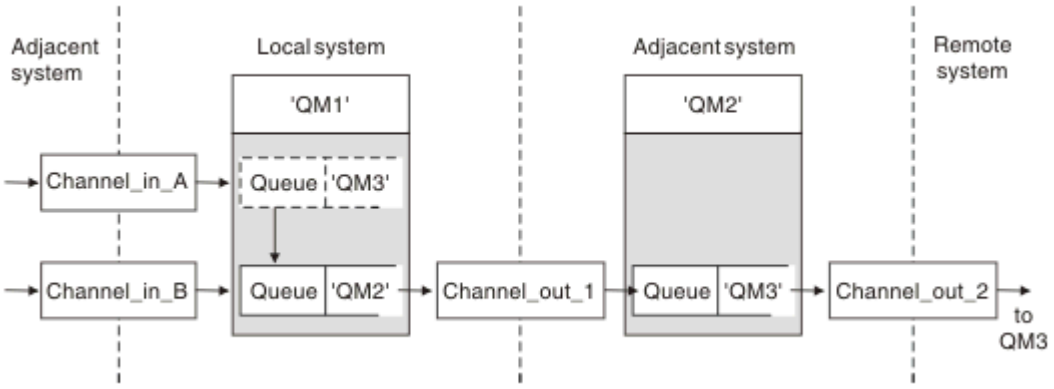


그림 *15.* 큐 관리자 알리어스

QM3의 모든 메시지는 큐 관리자 알리어스가 있는 QM1에 캡처됩니다. 큐 관리자 알리어스의 이름은 QM3로 지정되며 전송 큐 QM2를 통한 정의 QM3를 포함합니다. 정의는 다음 예제와 같습니다.

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

큐 관리자는 메시지를 전송 큐 QM2에 넣지만 목적지 큐 관리자 QM3의 이름이 변경되지 않아 전송 큐 헤더를 변경하지 않습니다.

QM1에 도달하여 QM2의 큐 이름을 포함한 전송 헤더를 표시하는 모든 메시지도 QM2 전송 큐에 넣습니다. 이런 방식으로 목적지가 다른 메시지는 적절한 인접 시스템에 대한 공용 전송 큐로 수집되어 계속 목적지로 전송됩니다.

인바운드 메시지 - 목적지 판별

수신 MCA는 전송 헤더에서 참조된 큐를 엽니다. 큐 관리자 알리어스 정의가 참조된 큐 관리자와 동일한 이름으로 존재하면 전송 헤더에서 수신된 큐 관리자 이름은 해당 목적지의 RQMNAME으로 바뀝니다.

이 프로세스는 다음과 같이 2가지로 사용됩니다.

- 메시지를 다른 큐 관리자에게 전달
- 큐 관리자 이름을 로컬 큐 관리자와 동일하도록 변경

응답 대상 큐 알리어스 정의

응답 대상 큐 알리어스 정의는 메시지 디스크립터에서 응답 정보에 대한 대체 이름을 지정합니다. 애플리케이션을 변경하지 않고 큐나 큐 관리자의 이름을 변경할 수 있다는 이점이 있습니다.

큐 이름 해석

애플리케이션이 메시지에 응답할 때 수신된 메시지의 메시지 디스크립터에 있는 데이터를 사용하여 응답할 큐의 이름을 찾습니다. 송신 애플리케이션은 응답이 송신되어 이 정보를 메시지에 첨부하는 위치를 표시합니다. 이 개념은 애플리케이션 디자인의 일부로 통합될 수 있습니다.

큐 이름 해석은 메시지를 큐에 넣기 전에 애플리케이션의 송신 끝에서 발생합니다. 따라서 큐 이름 해석은 원격 애플리케이션과 상호작용하기 전에 발생합니다. 이 상황에서만 큐가 열리지 않을 때 이름 해석이 이루어집니다.

큐 관리자 알리어스를 사용하여 큐 이름 해석

보통 애플리케이션은 응답 대상 큐를 지정하며 응답 대상 큐 관리자 이름을 공백으로 둡니다. 큐 관리자는 넣기 시 자신의 이름을 완성합니다. 예를 들어, 이 방법은 전송 큐 QM1을 사용하는 기본 리턴 채널 대신 전송 큐

QM1_relief를 사용하는 채널의 경우와 같이 응답에 사용할 대체 채널을 원하는 경우를 제외하고는 잘 작동합니다. 이런 상황에서는 전송 큐 헤더에 지정된 큐 관리자 이름이 "실제" 큐 관리자 이름과 일치하지 않지만 큐 관리자 알리어스 정의를 사용하여 다시 지정됩니다. 대체 라우트와 함께 응답을 리턴하려면 응답 대상 큐 알리어스 정의를 사용하여 응답 대상 큐 데이터에도 맵핑해야 합니다.

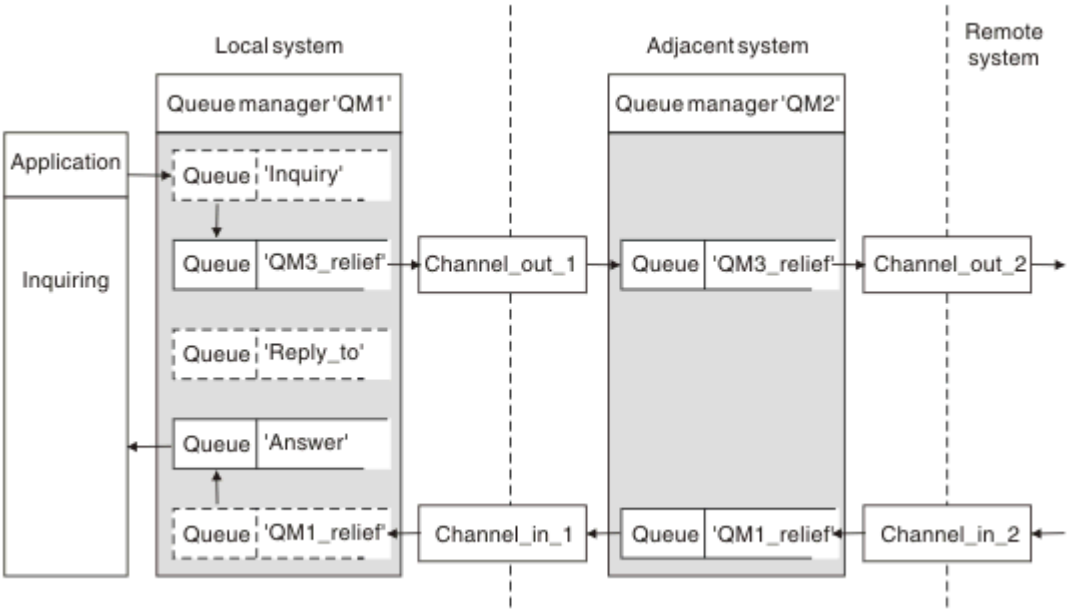


그림 *16.* 응답 위치 변경에 사용되는 응답 대상 큐 알리어스

49 페이지의 그림 16에서의 예제:

1. 애플리케이션은 MQPUT 호출을 사용하고 메시지 디스크립터에서 다음 정보를 지정하여 메시지를 넣습니다.

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

   응답 대상 큐 알리어스가 사용되려면 ReplyToQMgr는 공백이어야 합니다.
2. 이름 Answer 및 큐 관리자 이름 QM1_relief가 포함된 Reply_to라는 응답 대상 큐 알리어스 정의를 작성합니다.

   ```
   DEFINE QREMOTE ('Reply_to') RNAME ('Answer')
   RQMNAME ('QM1_relief')
   ```
3. 메시지는 ReplyToQ='Answer' 및 ReplyToQMgr='QM1_relief'를 표시하는 메시지 디스크립터와 함께 전송됩니다.
4. 애플리케이션 스펙에는 응답을 Reply_to가 아닌 큐 Answer에서 찾을 수 있는 정보를 포함해야 합니다.

다음을 정의하는 병렬 리턴 채널을 작성해야 하는 응답을 준비하려면:

- QM2에서, QM1_relief라는 전송 큐

  ```
  DEFINE QLOCAL ('QM1_relief') USAGE(XMITQ)
  ```
- QM1에서, 큐 관리자 알리어스 QM1_relief

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

이 큐 관리자 알리어스는 병렬 리턴 채널의 체인을 종료하며 QM1의 메시지를 캡처합니다.

나중에 가끔 이를 수행하기를 원한다고 생각되면, 애플리케이션이 시작부터 알리어스 이름을 사용하는지 확인하십시오. 현재까지는 응답 대상 큐에 대한 정상 큐 알리어스이지만 나중에는 큐 관리자 알리어스로 변경될 수 있습니다.

응답 대상 큐 이름

응답 대상 큐 이름 지정에 주의가 필요합니다. 애플리케이션이 응답 대상 큐 이름을 메시지에 넣은 이유는 응답이 전송되는 큐를 지정할 수 있기 때문입니다. 이 이름으로 응답 대상 큐 알리어스 정의를 작성하면 실제 응답 대상 큐(즉, 로컬 큐 정의)를 동일한 이름으로 지정할 수 없습니다. 그러므로 응답 대상 큐 알리어스 정의가 큐 관리자 이름뿐만 아니라 새 큐 이름을 포함해야 하며 애플리케이션 스펙에는 다른 큐에서 응답을 찾을 수 있는 정보를 포함해야 합니다.

이제 애플리케이션은 원본 메시지를 넣을 때의 응답 대상 큐로 이름 지정된 큐와 다른 큐에서 메시지를 검색해야 합니다.

클러스터 컴포넌트

클러스터는 큐 관리자, 클러스터 저장소, 클러스터 채널 및 클러스터 큐로 구성됩니다.

각 클러스터 컴포넌트에 대한 정보는 다음 하위 주제를 참조하십시오.

관련 개념
클러스터링과 분산 큐잉의 비교
관련 태스크
큐 관리자 클러스터 구성
새 클러스터 설정

클러스터 저장소

저장소는 클러스터의 멤버인 큐 관리자에 대한 정보의 콜렉션입니다.

저장소 정보에는 큐 관리자 이름, 해당 위치, 해당 채널, 호스팅하고 있는 큐 및 기타 정보가 포함됩니다. 이 정보는 SYSTEM.CLUSTER.REPOSITORY.QUEUE라고 하는 큐에 메시지의 양식으로 저장됩니다. 이 큐는 기본 오브젝트 중 하나입니다. Multi 멀티플랫폼에서는 IBM MQ 큐 관리자를 작성할 때 정의됩니다. z/OS IBM MQ for z/OS에서 큐 관리자 사용자 정의의 일부로 정의됩니다.

전체 저장소와 부분 저장소

일반적으로, 클러스터에서 두 개의 큐 관리자가 전체 저장소를 보유합니다. 나머지 큐 관리자 모두 부분 저장소를 보유합니다.

클러스터에서 모든 큐 관리자에 대한 전체 정보 세트를 호스팅하는 큐 관리자에 전체 저장소가 있습니다. 클러스터의 다른 큐 관리자에는 전체 저장소에 있는 정보의 서브세트를 포함하는 부분 저장소가 있습니다.

부분 저장소에는 큐 관리자가 메시지를 교환해야 하는 큐 관리자에 대한 정보만 포함됩니다. 큐 관리자는 필요한 정보에 대한 업데이트를 요청합니다. 그러면 필요한 경우, 전체 저장소 큐 관리자가 큐 관리자에 새 정보를 보낼 수 있습니다. 많은 시간 동안, 부분 저장소에는 큐 관리자가 클러스터 내에서 수행하는 데 필요한 모든 정보가 포함됩니다. 큐 관리자에서 일부 추가 정보가 필요한 경우, 큐 관리자는 전체 저장소를 조회하여 자신의 부분 저장소를 업데이트합니다. 큐 관리자는 SYSTEM.CLUSTER.COMMAND.QUEUE라는 큐를 사용하여 저장소에 대한 업데이트를 요청 및 수신합니다.

클러스터의 멤버인 큐 관리자를 마이그레이션하는 경우 부분 저장소를 마이그레이션하기 전에 전체 저장소를 마이그레이션하십시오. 이는 이전 저장소가 신규 릴리스에서 도입된 새 속성을 저장하지 못하기 때문입니다. 이전 저장소는 이를 허용하지만 저장하지는 않습니다.

클러스터 큐 관리자

클러스터 큐 관리자는 클러스터의 멤버인 큐 관리자입니다.

큐 관리자는 둘 이상의 클러스터의 멤버가 될 수 있습니다. 각 클러스터 큐 관리자의 이름은 멤버인 모든 클러스터에서 고유해야 합니다.

클러스터 큐 관리자는 큐를 호스팅할 수 있습니다. 이 큐는 클러스터에서 다른 큐 관리자에 광고합니다. 그러나 이 작업을 수행할 필요는 없습니다. 이 큐 관리자는 클러스터의 다른 위치에서 호스트된 큐에 메시지를 대신 공급하고 해당 클러스터에 명시적으로 전달되는 응답만 수신할 수 있습니다.

z/OS IBM MQ for z/OS에서 클러스터 큐 관리자는 큐 공유 그룹의 멤버일 수 있습니다. 이러한 경우, 클러스터 큐 관리자는 샘플 큐 공유 그룹의 다른 큐 관리자와 해당되는 큐 정의를 공유합니다.

클러스터 큐 관리자는 자율적입니다. 클러스터 큐 관리자는 정의하는 채널 및 큐에 대해 전체 제어를 가지고 있습니다. 해당되는 정의는 다른 큐 관리자(동일한 큐 공유 그룹에 있는 큐 관리자가 아닌 다른)에서 수행할 수 없습니다. 저장소 큐 관리자는 클러스터의 다른 큐 관리자에서 정의를 제어하지 않습니다. 필요할 때 사용하기 위해 모든 전체 정의 세트를 보유합니다. 클러스터는 큐 관리자의 연합입니다.

클러스터 큐 관리자에서 정의를 작성하거나 변경한 후, 정보는 전체 저장소 큐 관리자로 보내집니다. 클러스터의 다른 저장소는 나중에 업데이트됩니다.

전체 저장소 큐 관리자

전체 저장소 큐 관리자는 클러스터의 자원에 대한 전체 표시를 보유하는 클러스터 큐 관리자입니다. 사용 가능성을 보증하기 위해, 각 클러스터에서 두 개 이상의 전체 저장소 큐 관리자를 설정하십시오. 전체 저장소 큐 관리자는 클러스터의 다른 큐 관리자가 보낸 정보를 수신하고 해당되는 저장소를 업데이트합니다. 둘 다 클러스터에 대한 새 정보로 최신 상태를 유지하도록 하기 위해 서로 메시지를 보냅니다.

큐 관리자 및 저장소

모든 클러스터에는 클러스터에 있는 큐 관리자, 큐 및 채널에 대한 정보의 전체 저장소를 보유하는 최소 하나의 (두 개가 선호됨) 큐 관리자가 있습니다. 이 저장소는 또한 클러스터의 다른 큐 관리자로부터의 정보에 대한 업데이트 요청을 포함합니다.

다른 큐 관리자는 각각 통신해야 하는 큐 관리자와 큐 서브세트에 대한 정보를 포함하는 부분 저장소를 보유합니다. 큐 관리자는 먼저 다른 큐 또는 큐 관리자에 액세스해야 하는 경우 조회를 작성하여 부분 저장소를 빌드합니다. 큐 관리자는 해당 큐 또는 큐 관리자에 관한 새 정보를 알리도록 요청합니다.

각 큐 관리자는 `SYSTEM.CLUSTER.REPOSITORY.QUEUE`라고 하는 큐에 메시지의 저장소 정보를 저장합니다. 큐 관리자는 `SYSTEM.CLUSTER.COMMAND.QUEUE`라고 하는 큐에서 메시지의 저장소 정보를 교환합니다.

클러스터에 조인하는 각 큐 관리자는 클러스터 송신자 `CLUSSDR` 채널을 저장소 중 하나에 대해 정의합니다. 클러스터의 다른 큐 관리자가 전체 저장소를 보유한다고 즉시 학습합니다. 그때부터, 큐 관리자는 저장소에서 정보를 요청할 수 있습니다. 큐 관리자가 정보를 선택된 저장소로 보낼 때, 하나의 다른 저장소(있는 경우)로도 정보를 보냅니다.

전체 저장소는 호스팅하는 큐 관리자가 링크되는 큐 관리자 중 하나에서 새 정보를 수신할 때 업데이트됩니다. 새 정보는 또한 저장소 큐 관리자가 서비스 외부에 있는 경우 지연되는 위험을 줄이기 위해 다른 저장소로 보냅니다. 모든 정보는 두 번 보내므로, 저장소는 중복을 버려야 합니다. 정보의 각 항목은 저장소가 중복을 식별하기 위해 사용하는 순서 번호를 전달합니다. 모든 저장소는 메시지를 교환하여 서로 맞춥니다.

클러스터 큐

클러스터 큐는 클러스터 큐 관리자에 의해 호스팅되며 클러스터의 다른 큐 관리자가 사용할 수 있는 큐입니다.

클러스터 큐 정의는 클러스터의 다른 큐 관리자에 통지됩니다. 다른 큐 관리자는 해당하는 리모트 큐 정의 없이도 클러스터 큐에 메시지를 넣을 수 있습니다. 클러스터 큐는 클러스터 이름 목록을 사용하여 둘 이상의 클러스터에 통지될 수 있습니다.

큐가 통지되면 클러스터의 큐 관리자가 해당 큐에 메시지를 넣을 수 있습니다. 메시지를 넣으려면 큐 관리자가 전체 저장소에서 큐가 호스팅되고 있는 위치를 찾아야 합니다. 그런 다음 메시지에 몇 가지 라우팅 정보를 추가하고 클러스터 전송 큐에 메시지를 넣습니다.

클러스터 큐는 IBM MQ for z/OS에서 큐 공유 그룹의 멤버가 공유하는 큐일 수 있습니다.

관련 태스크
클러스터 큐 정의

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.

클러스터 채널

모든 전체 저장소에서 클러스터-수신기 채널과 클러스터에 있는 기타 모든 전체 저장소에 연결하는 클러스터-송신기 채널 세트를 수동으로 정의합니다. 부분 저장소를 추가할 때 클러스터-수신기 채널과 전체 저장소 중 하나에 연결하는 단일 클러스터-송신기 채널을 수동으로 정의합니다. 추가 클러스터-송신기 채널은 필요할 때 클러스터에서 자동으로 정의합니다. 자동 정의된 클러스터-송신기 채널은 수신 큐 관리자에서 대응하는 클러스터-수신기 채널 정의의 속성을 사용합니다.

클러스터-수신자 채널: CLUSRCVR

CLUSRCVR 채널 정의는 클러스터 큐 관리자가 클러스터에서 다른 큐 관리자로부터 메시지를 수신할 수 있는 채널의 끝을 정의합니다.

클러스터 큐 관리자마다 하나 이상의 CLUSRCVR 채널을 정의해야 합니다. CLUSRCVR 채널을 정의하여, 큐 관리자는 메시지를 수신할 수 있음을 다른 클러스터 큐 관리자에 표시합니다.

CLUSRCVR 채널 정의를 사용하여 다른 큐 관리자가 해당 클러스터-송신자 채널 정의를 자동으로 정의할 수 있습니다. 이 문서의 53 페이지의 『자동 정의된 클러스터-송신자 채널』 절을 참조하십시오.

클러스터-송신자 채널: CLUSSDR

전체 저장소 큐 관리자 모두에서 클러스터에 있는 다른 모든 전체 저장소 큐 관리자까지 CLUSSDR 채널을 수동으로 정의합니다. 전체 저장소에 의해 교환되는 모든 업데이트는 이 채널에서 독점적으로 플로우됩니다. 이러한 채널을 수동으로 정의하여 전체 저장소의 네트워크를 명시적으로 제어합니다.

부분 저장소 큐 관리자를 클러스터에 추가할 때 전체 저장소 중 하나에 연결할 단일 CLUSSDR 채널을 수동으로 정의합니다. 처음에 접속한 후 CLUSSDR 채널을 포함하는 큐 관리자의 추가 클러스터 큐 관리자 오브젝트가 필요에 따라 자동으로 정의되므로 어떤 전체 저장소를 선택하느냐는 관계가 거의 없습니다. 따라서 큐 관리자가 모든 전체 저장소에 클러스터 정보를 보내고, 클러스터에 있는 모든 큐 관리자에 메시지를 보낼 수 있습니다.

이 문서의 절에 설명되어 있는 것처럼 자동 정의된 송신자 채널은 클러스터-수신자 채널의 구성을 기반으로 합니다. 따라서 클러스터 채널에 설정하는 채널 특성은 일치하는 CLUSSDR 및 클러스터-수신자 채널에서 동일하게 설정되거나 클러스터-수신자 채널에만 설정되어야 합니다.

이전에 설명한 이유에 한해서만 CLUSSDR 채널을 수동으로 정의해야 합니다. 즉, 부분 저장소를 전체 저장소에 처음 연결하거나 두 개의 전체 저장소를 함께 연결하려는 경우입니다. 부분 저장소에 연결하거나 클러스터에 없는 큐 관리자에 연결하는 CLUSSDR 채널을 수동으로 구성하면 AMQ9427 및 AMQ9428과 같은 오류 메시지가 발생됩니다. 예를 들어, 전체 저장소의 위치를 수정하는 경우와 같이 일시적으로 이러한 상황이 불가피할 수는 있지만 수동 정의는 가능한 빨리 삭제해야 합니다.

자동 정의된 클러스터-송신자 채널

일반적으로 부분 저장소 큐 관리자를 클러스터에 추가할 때 다음과 같이 큐 관리자에서 두 개의 클러스터 채널만 수동으로 정의합니다.

- 클러스터의 전체 저장소 큐 관리자에 대한 클러스터-송신자(CLUSSDR) 채널.
- 클러스터 수신자(CLUSRCVR) 채널.

사용자가 정의하는 CLUSSDR 채널을 사용하여 큐 관리자가 클러스터에 처음 접속할 수 있습니다. 처음 접속한 후 클러스터가 필요에 따라 추가 CLUSSDR 채널을 자동으로 정의합니다.

자동 정의된 CLUSSDR 채널은 수신 큐 관리자에 있는 대응하는 CLUSRCVR 채널 정의의 속성을 사용합니다. 수동으로 정의된 CLUSSDR 채널이 있어도 자동 정의된 CLUSSDR 채널의 속성이 사용됩니다. 예를 들어, **CONNAME** 매개변수에 포트 번호를 지정하지 않고 CLUSRCVR 채널을 정의하고 포트 번호를 지정하는 CLUSSDR 채널을 수동으로 정의한다고 가정하십시오. 자동 정의된 CLUSSDR 채널이 수동으로 정의된 채널을 대체하면 포트 번호(CLUSRCVR 채널에서 가져옴)가 비어 있게 됩니다. 기본 포트 번호를 사용하고 채널이 실패합니다.

수동으로 정의된 CLUSSDR 채널과 대응하는 CLUSRCVR 채널 정의의 구성이 서로 다른 경우, 일부 차이점은 바로 적용되며(예: 워크로드 밸런싱 매개변수) 다른 일부는 채널이 재시작될 때 적용됩니다(예: TLS 구성).

혼동을 피하기 위해 가능하면 다음 지침을 준수하십시오.

- 전체 저장소를 가리키는 CLUSSDR 채널만 수동으로 정의하십시오.
- 수동으로 정의한 CLUSSDR 채널이 있는 경우 수신 큐 관리자에 있는 대응하는 CLUSRCVR 채널 정의와 동일하게 구성하십시오.

자동 정의된 채널에 대한 작업도 참조하십시오.

관련 개념
자동 정의된 채널에 대한 작업

클러스터 전송 큐 및 클러스터 송신자 채널에 대한 작업

관련 태스크

새 클러스터 설정

클러스터에 큐 관리자 추가

클러스터 토픽

클러스터 토픽은 **cluster** 속성이 정의된 관리 토픽입니다. 클러스터 토픽에 대한 정보는 클러스터의 모든 멤버에게 푸시된 다음 로컬 토픽과 결합되어 여러 큐 관리자에 걸쳐 있는 토픽 공간의 일부분을 구성합니다. 따라서 한 큐 관리자의 토픽에 발행된 메시지를 클러스터에 있는 다른 큐 관리자의 구독으로 전달할 수 있습니다.

큐 관리자에 대한 클러스터 토픽을 정의하면 클러스터 토픽 정의가 전체 저장소 큐 관리자로 송신됩니다. 그러면 전체 저장소에서 클러스터 토픽 정의가 클러스터 내의 모든 큐 관리자로 전파되므로 클러스터의 모든 큐 관리자에 있는 발행자와 구독자가 동일한 클러스터 토픽을 사용할 수 있습니다. 클러스터 토픽이 작성된 큐 관리자를 클러스터 토픽 호스트라고 합니다. 클러스터 토픽은 클러스터의 모든 큐 관리자에서 사용할 수 있지만, 해당 토픽이 정의된 큐 관리자(호스트)에서 수정해야 합니다. 이 경우 수정사항이 전체 저장소를 통해 클러스터의 모든 멤버에게 전파됩니다.

직접 라우팅 또는 토픽 호스트 라우팅을 사용하도록 클러스터 토픽을 구성하는 데 대한 정보 및 클러스터된 토픽 상속 및 와일드카드 구독에 대한 정보는 클러스터 토픽 정의를 참조하십시오.

클러스터 토픽을 표시하는 데 사용할 명령에 대한 정보는 관련 정보를 참조하십시오.

관련 개념

관리 토픽에 대한 작업

구독에 대한 작업

관련 참조

표시 주제

표시 TP상태

표시 등록

기본 클러스터 오브젝트

Multi 멀티플랫폼에서 기본 클러스터 오브젝트는 큐 관리자를 정의할 때 자동으로 작성되는 기본 오브젝트 세트에 포함됩니다. z/OS z/OS의 사용자 정의 샘플에서 기본 클러스터 오브젝트 정의를 찾을 수 있습니다.

참고: 다른 채널 정의와 동일한 방법으로, MQSC 또는 PCF 명령을 실행하여 기본 채널 정의를 변경할 수 있습니다. SYSTEM.CLUSTER.HISTORY.QUEUE의 경우를 제외하고는 기본 큐 정의를 변경하지 마십시오.

SYSTEM.CLUSTER.COMMAND.QUEUE

클러스터의 각 큐 관리자는 메시지를 전체 저장소로 전송하는 데 사용되는 SYSTEM.CLUSTER.COMMAND.QUEUE라고 하는 로컬 큐입니다. 메시지에는 큐 관리자에 대한 새 정보나 변경된 정보, 또는 다른 큐 관리자에 관한 정보에 대한 요청이 포함됩니다. SYSTEM.CLUSTER.COMMAND.QUEUE는 보통 비어 있습니다.

SYSTEM.CLUSTER.HISTORY.QUEUE

클러스터의 각 큐 관리자에는 SYSTEM.CLUSTER.HISTORY.QUEUE(이)라는 로컬 큐가 있습니다. SYSTEM.CLUSTER.HISTORY.QUEUE은(는) 서비스 목적으로 클러스터 상태 정보의 히스토리를 저장하는 데 사용됩니다.

기본 오브젝트 설정에서 SYSTEM.CLUSTER.HISTORY.QUEUE은(는) PUT(ENABLED)으로 설정됩니다. 히스토리 콜렉션을 억제하려면 설정을 PUT(DISABLED)으로 변경하십시오.

SYSTEM.CLUSTER.REPOSITORY.QUEUE

클러스터의 각 큐 관리자에는 SYSTEM.CLUSTER.REPOSITORY.QUEUE(이)라는 로컬 큐가 있습니다. 이 큐는 모든 전체 저장소 정보를 저장하는 데 사용됩니다. 이 큐는 보통 비어있지 않습니다.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
각 큐 관리자에는 로컬 큐 SYSTEM.CLUSTER.TRANSMIT.QUEUE에 대한 정의가 있습니다. SYSTEM.CLUSTER.TRANSMIT.QUEUE은(는) 클러스터 내에 있는 모든 큐 및 큐 관리자에 대한 모든 메시지의 기본 전송 큐입니다. 큐 관리자 속성 DEFCLXQ 을 변경하여 각 클러스터-송신자 채널의 기본 전송 큐를 SYSTEM.CLUSTER.TRANSMIT. *ChannelName*로 변경할 수 있습니다.
SYSTEM.CLUSTER.TRANSMIT.QUEUE를 삭제할 수 없습니다. 또한 사용되는 기본 전송 큐가 SYSTEM.CLUSTER.TRANSMIT.QUEUE 또는 SYSTEM.CLUSTER.TRANSMIT. *ChannelName*인지 여부를 확인하는 권한 검사를 정의하는 데 사용됩니다.

SYSTEM.DEF.CLUSRCVR
각 클러스터에는 SYSTEM.DEF.CLUSRCVR이라고 하는 기본 CLUSRCVR 채널 정의가 있습니다. SYSTEM.DEF.CLUSRCVR은 클러스터의 큐 관리자에서 클러스터 수신자 채널을 작성할 때 지정하지 않는 속성에 대한 기본값을 제공하기 위해 사용됩니다.

SYSTEM.DEF.CLUSSDR
각 클러스터에는 SYSTEM.DEF.CLUSSDR이라고 하는 기본 CLUSSDR 채널 정의가 있습니다. SYSTEM.DEF.CLUSSDR은 클러스터의 큐 관리자에서 클러스터 송신자 채널을 작성할 때 지정하지 않는 속성에 대한 기본값을 제공하기 위해 사용됩니다.

관련 개념
기본 클러스터 오브젝트에 대한 작업

발행/구독 메시징

발행/구독 메시징을 사용하면 해당 정보의 이용자로부터 정보의 제공자를 분리시킬 수 있습니다. 정보가 송수신되기 위해 송신 애플리케이션과 수신 애플리케이션이 서로에 대해 어느 것도 알 필요가 없습니다.

포인트-투-포인트 IBM MQ 애플리케이션은 메시지를 다른 애플리케이션으로 송신하기 전에 해당 애플리케이션에 대한 일부 정보를 알아야 합니다. 예를 들어 정보를 송신할 큐의 이름을 알아야 하며 큐 관리자 이름을 지정할 수도 있습니다.

IBM MQ 발행/구독에서는 애플리케이션이 대상 애플리케이션을 알지 않아도 됩니다. 모든 송신 애플리케이션은 다음을 수행해야 합니다.

- 애플리케이션이 원하는 정보를 포함하는 IBM MQ 메시지를 넣으십시오 .
- 정보의 주제를 나타내는 메시지를 토픽에 지정합니다.
- IBM MQ에서 해당 정보 분배를 처리하도록 허용합니다.

유사하게 대상 애플리케이션은 수신한 정보의 소스에 대해 알 필요가 없습니다.

다음 그림은 가장 간단한 발행/구독 시스템을 표시합니다. 하나의 발행자, 하나의 큐 관리자 및 하나의 구독자가 있습니다. 큐 관리자의 구독자가 구독을 작성하고, 발행이 발행자로부터 큐 관리자로 송신된 후 발행이 큐 관리자에 의해 구독자에게 전달됩니다.

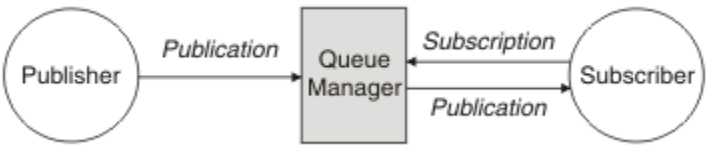


그림 *17.* 간단한 발행/구독 구성

전형적인 발행/구독 시스템에는 많은 다른 토픽에서 둘 이상의 발행자와 둘 이상의 구독자 및 종종 둘 이상의 큐 관리자가 있습니다. 한 애플리케이션이 발행자이면서 구독자일 수 있습니다.

발행/구독 메시징과 포인트-투-포인트 간의 다른 큰 차이점은 포인트-투-포인트 큐로 보내진 메시지는 단일 사용 애플리케이션에 의해서만 처리된다는 점입니다. 발행/구독 토픽에 발행된 메시지는(둘 이상의 구독자가 관심을 등록함) 모든 관심 있는 구독자에 의해 처리됩니다.

발행/구독 컴포넌트

발행/구독은 구독자가 발행자로부터 메시지 양식으로 정보를 수신할 수 있는 메커니즘입니다. 발행자와 구독자 사이의 상호작용은 표준 IBM MQ 기능을 사용하여 큐 관리자에 의해 제어됩니다.

전형적인 발행/구독 시스템에는 많은 다른 토픽에서 둘 이상의 발행자와 둘 이상의 구독자 및 종종 둘 이상의 큐 관리자가 있습니다. 한 애플리케이션이 발행자이면서 구독자일 수 있습니다.

정보 제공자는 발행자라고 합니다. 발행자는 해당 정보에 관심이 있는 애플리케이션에 대해 알지 않고도 주제에 대한 정보를 제공합니다. 발행자는 이 메시지의 토픽을 발행하고 정의하려는 발행이라고 하는 메시지 양식으로 이 정보를 생성합니다.

정보의 이용자는 구독자라고 합니다. 구독자는 구독자가 관심이 있는 토픽을 설명하는 구독을 작성합니다. 따라서 구독은 구독자에게 전달되는 발행을 판별합니다. 구독자는 여러 구독을 작성하고 많은 다양한 발행자로부터 정보를 수신할 수 있습니다.

발행된 정보는 IBM MQ 메시지에서 전송되며, 정보의 주제는 해당 토픽으로 식별됩니다. 발행자는 정보를 발행할 때 토픽을 지정하고 구독자는 발행을 수신하려는 토픽을 지정합니다. 구독자는 구독하는 토픽에 대한 정보만 받습니다.

토픽을 통해 포인트-투-포인트 메시징에 필요한 경우 각 메시지에서 특정 목적지를 포함해야 하는 필요를 없애 정보의 제공자와 이용자를 발행/구독 메시징에서 분리할 수 있습니다.

발행자와 구독자 사이의 상호작용은 큐 관리자에 의해 모두 제어됩니다. 큐 관리자는 발행자로부터 메시지를 수신하고 토픽 범위에 따라 구독자로부터 구독을 수신합니다. 큐 관리자의 작업은 메시지 토픽에서 관심사를 등록한 구독자에게 발행된 메시지를 라우팅하는 것입니다.

표준 IBM MQ 기능은 메시지 분산에 사용되므로 애플리케이션은 기존 IBM MQ 애플리케이션에서 사용 가능한 모든 기능을 사용할 수 있습니다. 즉, 지속 메시지를 사용하여 한 번만 보장된 전달을 받을 수 있으며, 발행자가 커미트한 경우에만 구독자에게 메시지를 전달하도록 메시지를 트랜잭션 작업 단위로 포함할 수 있음을 의미합니다.

발행자 및 발행물

IBM MQ 발행/구독에서 발행자는 발행물이라고 하는 표준 IBM MQ 메시지의 양식으로 큐 관리자가 사용할 수 있는 특정 토픽에 대한 정보를 작성하는 애플리케이션입니다. 발행자는 둘 이상의 토픽에 대한 정보를 발행할 수 있습니다.

발행자는 MQPUT 동사를 사용하여 이전에 열린 토픽에 메시지를 넣으며, 이 메시지는 발행물입니다. 그리고 로컬 큐 관리자는 발행물의 토픽을 구독하는 구독자에게 해당 발행물을 라우팅합니다. 발행된 메시지를 둘 이상의 구독자가 이용할 수 있습니다.

해당 구독을 보유하는 모든 로컬 구독자에게 발행물을 배포함은 물론, 큐 관리자는 이에 연결된 기타 큐 관리자에게 직접적으로 또는 토픽에 대한 구독자가 있는 큐 관리자의 네트워크를 통해 발행물을 배포할 수도 있습니다.

IBM MQ 발행/구독 네트워크에서 발행 애플리케이션은 구독자일 수도 있습니다.

동기점 하의 발행물

발행자는 작업 단위로 구독자에게 전달된 모든 메시지를 포함하도록 동기점에서 MQPUT 또는 MQPUT1 호출을 실행할 수 있습니다. MQPMO_RETAIN 옵션 또는 토픽 전달 옵션 NPMSGDLV 또는 PMSGDLV(ALL 또는 ALLDUR 값의)가 지정되면, 큐 관리자는 발행자 MQPUT 또는 MQPUT1 호출 범위 내의 동기점에서 내부 MQPUT 또는 MQPUT1 호출을 사용합니다.

상태 및 이벤트 정보

발행은 주식의 현재 가격과 같은 상태 발행이나, 해당 주식의 거래와 같은 이벤트 발행으로 분류할 수 있습니다.

상태 발행

상태 발행은 축구 경기의 현재 점수 또는 주가와 같이 어떤 항목의 현재 상태에 대한 정보를 포함합니다. 주가가 변동되거나 축구 경기의 점수가 변경되는 것과 같이 무슨 일이 발생하면 이전 상태 정보가 새 정보로 대체되므로 더 이상 필요하지 않게 됩니다.

구독자는 시작할 때 상태 정보의 최신 버전을 수신하려고 하며, 상태가 변경되면 새 정보가 전송되기를 원합니다.

발행에 상태 정보가 포함되는 경우, 흔히 보유된 발행으로 발행됩니다. 일반적으로 새 구독자는 즉시 현재 상태 정보를 원하며, 구독자는 정보가 다시 발행되는 이벤트를 기다리기를 원하지 않습니다. 구독자가 MQSO_PUBLICATIONS_ON_REQUEST 또는 MQSO_NEW_PUBLICATIONS_ONLY 옵션을 사용하지 않는 한, 구독할 때 구독자는 자동으로 토픽의 보유된 발행을 수신합니다.

이벤트 발행

이벤트 발행은 일부 주식의 거래나 특정 골의 득점과 같이 발생하는 개별 이벤트에 대한 정보를 포함합니다. 각 이벤트는 다른 이벤트와 서로 독립적입니다.

구독자는 이벤트가 발생하면 해당 이벤트에 대한 정보를 수신하려고 합니다.

보유된 발행물

기본적으로 발행은 관련된 모든 구독자에게 전송된 후에 제거됩니다. 그러나 발행자는 발행 사본을 보유하도록 지정할 수 있습니다. 그러면 토픽에 관심사를 등록한 향후 구독자에게 이를 전송할 수 있습니다.

이벤트 정보의 경우 관련된 모든 구독자에게 전송된 후에 발행을 삭제하는 것이 적절하지만, 상태 정보에서는 항상 적절하지 않을 수도 있습니다. 메시지를 보유하면 새 구독자가 초기 상태 정보를 수신하기 전에 정보 발행을 다시 기다리지 않아도 됩니다. 예를 들어 주가에 대해 구독하는 구독자는 주가가 변경되고 이후에 다시 발행되기를 기다리지 않고도 현재 가격을 직접 수신할 수 있습니다.

큐 관리자는 각 토픽에 대해 하나의 발행만 보유할 수 있으므로 새 보유된 발행이 큐 관리자에 도달하면 토픽의 기존 보유된 발행은 삭제됩니다. 그러나 기존 발행의 삭제가 새 보유된 발행의 도착과 동시에 수행되지 않을 수도 있습니다. 따라서 토픽에서 보유된 발행을 송신하는 발행자가 하나만 존재할 수 있습니다.

구독자는 MQSO_NEW_PUBLICATIONS_ONLY 구독 옵션을 사용하여 보유된 발행을 수신하지 않도록 지정할 수 있습니다. 기존 구독자는 보유된 발행의 중복 사본을 전송해줄 것을 요청할 수 있습니다.

상태 정보라도 발행을 보유하지 않으려는 경우가 있습니다.

- 토픽에 대한 모든 구독이 해당 토픽에 대한 발행을 수행하기 전에 이루어진 경우 새 구독을 예상하지 않았거나 이를 허용하지 않으면 처음 발행될 때 전체 구독자 세트에 전달되므로 발행을 보유하지 않아도 됩니다.
- 발행이 자주(가령 매초마다) 나타나면 새 구독자 또는 장애에서 복구한 구독자는 초기 구독 이후 거의 즉시 현재 상태를 수신하므로 이러한 발행을 보유하지 않아도 됩니다.
- 발행이 큰 경우 각 토픽에 대한 보유된 발행을 저장하는 데 상당한 스토리지 공간이 필요하게 될 수도 있습니다. 다중 큐 관리자 환경에서 보유된 발행은 일치하는 구독이 있는 네트워크의 모든 관리자에서 저장합니다.

보유된 발행의 사용 여부를 결정할 때 구독 애플리케이션이 실패에서 복구하는 방법을 고려하십시오. 발행자가 보유된 발행을 사용하지 않을 경우, 구독자 애플리케이션은 현재 상태를 로컬로 저장해야 할 수도 있습니다.

발행을 보유하려면 MQPMO_RETAIN 메시지 넣기 옵션을 사용하십시오. 이 옵션이 사용되고 발행물을 보유할 수 없는 경우에는 메시지가 발행되지 않고 호출은 MQRC_PUT_NOT_RETAINED가 발생하여 실패합니다.

메시지가 보유된 발행이면 MQIsRetained 메시지 특성으로 표시됩니다. 메시지의 지속성은 원래 발행될 때 속성 그대로입니다.

관련 개념

발행/구독 클러스터에서 보유된 발행에 대한 디자인 고려사항

동기점 하의 발행물

IBM MQ 발행/구독에서 동기점은 발행자가 사용하거나 큐 관리자가 내부적으로 사용할 수 있습니다.

발행자는 MQPMO_SYNCPOINT 옵션을 사용하여 MQPUT/MQPUT1 호출을 발행할 때 동기점을 사용합니다. 구독자에 전달된 모든 메시지는 작업 단위에서 커미트되지 않은 최대 메시지 수로 계산됩니다. MAXUMSGS 큐 관리자 속성이 이 한계를 지정합니다. 한계에 도달하면 발행자가 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED 이유 코드를 수신합니다.

발행자가 MQPMO_RETAIN 옵션 또는 값이 ALL이나 ALLDUR인 토픽 전달 옵션 NPMSGDLV/PMSGDLV와 함께 MQPMO_NO_SYNCPOINT를 사용하여 MQPUT/MQPUT1 호출을 발행하는 경우 큐 관리자는 메시지가 요청된

대로 전달될 수 있도록 내부 동기점을 사용합니다. 한계가 발행자 MQPUT/MQPUT1 호출의 범위에 도달하면 발행자가 2024 (07E8) (RC2024): MQRC_SYNCPOINT_LIMIT_REACHED 이유 코드를 수신할 수 있습니다.

구독자 및 구독

IBM MQ 발행/구독에서 구독자는 발행/구독 네트워크에 있는 큐 관리자로부터 특정 토픽에 대한 정보를 요청하는 애플리케이션입니다. 구독자는 둘 이상의 발행자로부터 동일하거나 다른 토픽에 대한 메시지를 수신할 수 있습니다.

구독은 애플리케이션 또는 MQSC 명령을 사용하여 수동으로 작성할 수 있습니다. 이러한 구독은 로컬 큐 관리자로 발행되고, 구독자가 수신하려는 발행에 대한 정보를 포함합니다.

- 구독자가 관심이 있는 토픽. 이는 와일드카드를 사용하면 여러 토픽으로 해석될 수 있습니다.
- 발행된 메시지에 적용할 선택적 선택 문자열.
- 선택한 발행을 배치해야 하는 큐(구독자 큐라고 함)에 대한 핸들 및 선택적 CorrelId.

로컬 큐 관리자는 구독 정보를 저장하고 발행을 수신하면 구독의 토픽 및 선택 문자열과 일치하는 구독이 있는지 여부를 판별하도록 정보를 스캔합니다. 일치하는 각 구독에 대해 큐 관리자는 구독자의 구독자 큐로 발행을 지정합니다. 큐 관리자가 구독에 대해 저장하는 정보는 DIS SUB 및 DIS SBSTATUS 명령을 사용하여 볼 수 있습니다.

다음과 같은 이벤트 중 하나가 발생하는 경우에만 구독이 삭제됩니다.

- 구독자는 MQCLOSE 호출을 사용하여 구독을 해제합니다(구독이 비지속으로 작성된 경우).
- 구독이 만료됩니다.
- 구독은 시스템 관리자가 DELETE SUB 명령을 사용하여 삭제합니다.
- 구독자 애플리케이션이 종료됩니다(구독이 비지속으로 작성된 경우).
- 큐 관리자가 중지되거나 재시작됩니다(구독이 비지속으로 작성된 경우).

메시지를 가져올 때 MQGET 호출에서 적절한 옵션을 사용하십시오. 애플리케이션이 한 구독의 메시지만 처리하는 경우에는 최소한 `get-by-correlid`를 C 샘플 프로그램 `amqssbxa.c` 및 비관리 MQ 구독자에 시연되어 있는 바와 같이 사용해야 합니다. 사용할 **CorrelId** 는 MQSD의 MQSUB에서 리턴됩니다.**SubCorrelId** 필드.

관련 개념

복제 및 공유된 구독

관련 참조

sharedSubscription 특성을 정의하는 방법의 예

관리 큐 및 발행/구독

구독을 작성하면 관리 큐잉을 사용하도록 선택할 수 있습니다. 관리 큐잉을 사용하는 경우 구독을 작성할 때 구독 큐는 자동으로 작성됩니다. 관리 큐는 구독의 지속성에 따라 자동으로 정리됩니다. 관리 큐의 사용은 발행을 수신하는 큐를 작성하지 않아도 되며, 지속 불가능 구독 연결이 닫히면 이용하지 않은 발행은 구독자 큐에서 자동으로 제거됨을 의미합니다.

애플리케이션이 구독자 큐로 특정 큐를 사용하지 않아도 되면 이를 수신하는 발행의 목적지는 MQSO_MANAGED 구독 옵션을 사용하여 관리 구독을 사용할 수 있습니다. 관리 구독을 작성하면 큐 관리자는 발행이 수신되는 큐 관리자가 작성한 구독자 큐의 구독자에게 오브젝트 핸들을 리턴합니다. 이는 관리 구독이 IBM MQ에서 구독을 처리하는 구독이기 때문입니다. 큐 오브젝트 핸들을 리턴하여 큐를 찾아보거나 가져오거나 조회할 수 있습니다(임시 동적 큐에 대한 액세스 권한을 명시적으로 부여받지 않는 한, 관리 큐의 속성을 설정하거나 여기에 배치할 수 없음).

구독의 지속성은 구독 애플리케이션에서 큐 관리자로의 연결이 끊어진 경우 관리 큐가 남아 있는지 여부를 판별합니다.

관리 구독은 애플리케이션 연결이 종료될 때 (그렇지 않은 경우) 이용되지 않은 메시지가 구독 큐에 계속 남아 있어서 큐 관리자에서 무한으로 공간을 차지하므로 지속 불가능 구독과 함께 사용할 때 특히 유용합니다. 관리 구독을 사용하는 경우 관리 큐는 임시 동적 큐이며, 다음과 같은 이유로 연결이 끊어질 때 이용되지 않은 메시지와 함께 삭제됩니다.

- MQCO_REMOVE_SUB를 포함하는 MQCLOSE가 사용되고 관리 Hobj가 닫힙니다.
- 지속 불가능 구독(MQSO_NON_DURABLE)을 사용하는 애플리케이션과의 연결이 끊어집니다.

- 구독이 만료되었고 관리 Hobj가 닫혔으므로 구독이 제거됩니다.

관리 구독을 지속 가능 구독과 함께 사용할 수도 있지만 연결을 다시 열 때 검색할 수 있도록 이용되지 않은 메시지를 구독자 큐에 남겨둘 수 있습니다. 이러한 이유로 지속 가능 구독에 대한 관리 큐는 영구적 동적 큐 양식을 취하고 구독 애플리케이션과 큐 관리자의 연결이 끊어진 경우에도 그대로 남아 있습니다.

연결이 끊어진 후에도 큐가 그대로 존재하도록 영구적 동적 관리 큐를 사용하려는 경우 구독에서 만료를 설정할 수 있으며, 무한으로 계속 존재하지 않습니다.

관리 큐를 삭제하면 오류 메시지가 수신됩니다.

작성된 관리 큐의 이름은 각각 고유하도록 끝에 숫자(시간 소인)가 추가됩니다.

구독 지속성

구독은 지속 가능 또는 지속 불가능으로 구성될 수 있습니다. 구독 지속성은 구독 애플리케이션과 큐 관리자의 연결이 끊어질 때 구독에서 나타나는 상황을 판별합니다.

지속 가능 구독

지속 가능 구독의 경우 구독 애플리케이션과 큐 관리자의 연결이 닫힌 경우에도 계속 존재합니다. 구독이 지속 가능하면 구독 애플리케이션의 연결이 끊어질 때 구독은 그대로 남아 있으며, 구독을 작성할 때 리턴된 **SubName**을 사용하여 다시 구독을 요청하며 다시 연결할 때 구독 애플리케이션에서 이를 사용할 수 있습니다.

지속적으로 구독하는 경우 구독 이름(**SubName**)이 필요합니다. 구독을 식별하는 데 사용할 수 있도록 구독 이름은 큐 관리자에서 고유해야 합니다. 이러한 식별의 방법은 큐 관리자에서 연결이 끊어졌거나 MQCO_KEEP_SUB 옵션을 사용하여 구독에 대한 연결을 고의적으로 닫은 경우 재개하려는 구독을 지정할 때 필요합니다. MQSO_RESUME 옵션과 함께 MQSUB 호출을 사용하여 기존 구독을 계속할 수 있습니다. SUBTYPE ALL 또는 ADMIN과 함께 DISPLAY SBSTATUS 명령을 사용하는 경우에도 subscription 이름이 표시됩니다.

애플리케이션에 지속 가능 구독이 더 이상 필요하지 않으면 MQCO_REMOVE_SUB 옵션과 함께 MQCLOSE 함수를 사용하여 제거할 수 있거나 수동으로 MQSC 명령 DELETE SUB를 사용하여 삭제할 수 있습니다.

DURSUB 토픽 속성을 사용하여 토픽에 대해 지속 가능 구독을 작성할 수 있는지 여부를 지정할 수 있습니다.

MQSO_RESUME 옵션을 사용하여 MQSUB 호출에서 리턴할 때 구독 만료는 남은 만료 시간이 아닌, 원래 구독의 만료로 설정됩니다.

큐 관리자는 계속해서 발행을 송신하여 구독자 애플리케이션이 연결되지 않아도 지속 가능 구독을 만족시킬 수 있습니다. 이로 인해 구독자 큐에 메시지가 누적될 수 있습니다. 이 문제점을 피하는 가장 쉬운 방법은 해당되는 경우에만 지속 불가능 구독을 사용하는 것입니다. 그러나 지속 가능 구독을 사용해야 하는 경우 구독자가 보유된 발행 옵션을 사용하여 구독하면 메시지 누적을 방지할 수 있습니다. 그러면 구독자는 MQSUBRQ 호출을 사용하여 발행을 수신하는 시기를 제어할 수 있습니다.

지속 불가능 구독

지속 불가능 구독은 구독 애플리케이션과 큐 관리자의 연결이 열려 있는 경우에만 존재합니다. 이 구독은 고의적으로 또는 연결 유실에 의해 구독 애플리케이션과 큐 관리자의 연결이 끊길 때 제거됩니다. 연결이 닫히면 구독에 대한 정보가 큐 관리자에서 제거되고 DISPLAY SBSTATUS 명령을 사용하여 구독을 표시하는 경우 더 이상 표시되지 않습니다. 더 이상 구독자 큐에 메시지를 넣지 않습니다.

지속 불가능 구독에 대한 구독자 큐에서 이용되지 않은 발행의 처리 방법은 다음과 같이 판별됩니다.

- 구독 애플리케이션이 관리 목적지를 사용하는 경우 이용되지 않은 발행물이 자동으로 제거됩니다.
- 구독 애플리케이션이 구독 시 고유한 구독자 큐로 핸들을 제공하는 경우 이용되지 않은 메시지는 자동으로 제거되지 않습니다. 적절한 경우 큐를 지우는 것은 애플리케이션의 몫입니다. 큐가 둘 이상의 구독자 또는 다른 포인트-투-포인트 애플리케이션에서 공유되는 경우 큐를 완전히 지우는 것은 적절하지 않을 수도 있습니다.

지속 불가능 구독에 필요하지는 않지만 제공되는 경우 구독 이름은 큐 관리자에서 사용됩니다. 구독을 식별하는 데 사용할 수 있도록 구독 이름은 큐 관리자에서 고유해야 합니다.

관련 개념

복제 및 공유된 구독

관련 태스크
JMS 2.0 공유 구독 사용
관련 참조
sharedSubscription 특성을 정의하는 방법의 예

선택 문자열
선택 문자열은 구독과 일치하는지 여부를 판별하기 위해 발행에 적용되는 표현식입니다. 선택 문자열은 와일드카드 문자를 포함할 수 있습니다.

구독하는 경우 토픽 지정 외에도 선택 문자열을 지정하여 메시지 특성에 따라 발행을 선택할 수 있습니다.

선택 문자열은 각 구독자에 전달하도록 수정하기 전에 발행자가 입력한 메시지에 대해 평가됩니다. 선택 문자열에서 발행 조작의 일부로 수정될 수 있는 필드를 사용할 때는 주의하십시오. 예를 들어 MQMD 필드 `UserIdentifier`, `MsgId`, `CorrelId`가 이에 해당합니다.

선택 문자열은 발행 조작의 일부로 큐 관리자가 추가한 메시지 특성 필드를 참조하지 않습니다(발행/구독 메시지 특성 참조). 단, 발행을 위해 토픽 문자열을 포함하는 메시지 특성 `MQTopicString`은 예외입니다.

관련 개념
선택 문자열 규칙 및 제한사항

토픽
토픽은 발행/구독 메시지에서 발행된 정보의 제목입니다.

포인트-투-포인트 시스템의 메시지는 특정 목적지 주소로 송신됩니다. 주제 기반 발행/구독 시스템의 메시지는 메시지의 컨텐츠에 대해 설명하는 주제에 따른 구독자로 송신됩니다. 컨텐츠 기반 시스템에서, 메시지는 메시지의 컨텐츠를 기반으로 한 구독자로 송신됩니다.

IBM MQ 발행/구독 시스템은 주제 기반 발행/구독 시스템입니다. 발행자는 메시지를 작성하고 발행 주제에 맞는 토픽 문자열을 사용하여 해당 메시지를 발행합니다. 구독자는 발행을 수신하기 위해 발행 토픽을 선택하도록 토픽 문자열과 일치하는 패턴의 구독을 작성합니다. 큐 관리자는 발행 토픽과 일치하는 구독을 가진 구독자에게 발행을 전달하며 이 큐 관리자에게 발행을 수신할 수 있는 권한이 부여됩니다. 61 페이지의 『토픽 문자열』 글에서는 발행 주제를 식별하는 토픽 문자열의 구문에 대해 설명합니다. 또한 구독자는 수신할 토픽을 선택하도록 토픽 문자열을 작성합니다. 구독자가 작성하는 토픽 문자열에는 발행에 있는 토픽 문자열의 패턴 일치에 대한 두 가지 대체 와일드카드 설계 중 하나가 있습니다. 패턴 일치는 62 페이지의 『와일드카드 설계』에 설명되어 있습니다.

주제 기반 발행/구독에서는, 발행자 또는 관리자가 주제를 토픽으로 분류해야 합니다. 일반적으로 주제는 '/' 문자로 토픽 문자열에서 하위 토픽을 작성하여 계층적으로 토픽 트리로 구성됩니다. 토픽 트리 예는 67 페이지의 『토픽 트리』의 내용을 참조하십시오. 토픽은 토픽 트리의 노드입니다. 토픽은 추가 하위 토픽이 없는 리프 노드 또는 하위 토픽이 있는 중간 노드입니다.

주제를 계층적 토픽 트리로 구성하면서, 토픽을 관리 토픽 오브젝트와 연관시킬 수 있습니다. 토픽을 관리 토픽 오브젝트와 연관시켜 토픽을 클러스터에 분배할지의 여부와 같은 속성을 토픽에 지정합니다. 관리 토픽 오브젝트의 `TOPICSTR` 속성을 사용하여 토픽의 이름을 지정하면 연관이 이루어집니다. 명시적으로 관리 토픽 오브젝트를 토픽에 연관시키지 않으면, 토픽이 관리 토픽 오브젝트와 연관시킨 토픽 트리의 가장 가까운 상위의 속성을 상속합니다. 어떤 상위 토픽도 정의하지 않은 경우, `SYSTEM.BASE.TOPIC`에서 상속합니다. 관리 토픽 오브젝트는 68 페이지의 『관리 토픽 오브젝트』에 설명되어 있습니다.

참고: `SYSTEM.BASE.TOPIC`에서 토픽의 속성을 모두 상속하는 경우에도, `SYSTEM.BASE.TOPIC`에서 직접 상속하는 토픽의 루트 토픽을 정의하십시오. 예를 들어, 토픽 공간인 미국의 주 `USA/Alabama`, `USA/Alaska` 등에서는 `USA`가 루트 토픽입니다. 루트 토픽은 주로 잘못된 구독과 발행이 일치하는 것을 피하기 위해 겹치지 않는 개별 토픽 공간을 작성하는 데 사용됩니다. 또한 이 루트 토픽을 사용하면 전체 토픽 공간에 영향을 미치도록 루트 토픽의 속성을 변경할 수 있습니다. 예를 들어, **CLUSTER** 속성에 이름을 설정할 수 있습니다.

발행자 또는 구독자로 토픽을 참조하는 경우, 토픽 문자열을 제공하거나 토픽 오브젝트를 참조할 수 있습니다. 또한 제공하는 토픽 문자열에서 토픽 오브젝트의 하위 토픽을 정의하는 경우에는 둘 다 수행할 수 있습니다. 큐 관리자는 토픽 오브젝트에서 이름 지정된 토픽 문자열 접두부에 토픽 문자열을 추가하고 두 토픽 문자열 사이에 '/'를 삽입하여 토픽을 식별합니다(예: *topic string/object string*). 65 페이지의 『토픽 문자열 결합』에서는

이에 대해 자세히 설명합니다. 이렇게 생성된 토픽 문자열은 토픽을 식별하고 이 토픽을 관리 토픽 오브젝트와 연관시키는 데 사용됩니다. 관리 토픽 오브젝트는 마스터 토픽에 해당하는 토픽 오브젝트와 다를 수도 있습니다.

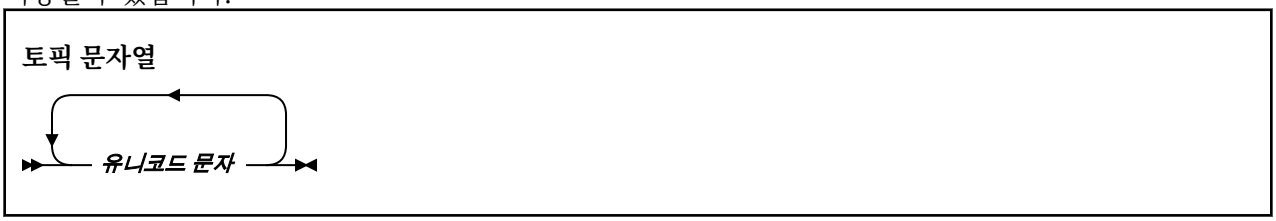
컨텐츠 기반 발행/구독에서는, 모든 메시지의 컨텐츠를 검색하는 선택 문자열을 제공하여 수신할 메시지 내용을 정의합니다. IBM MQ에서는 메시지의 전체 컨텐츠와 다른 메시지 특성을 스캔하는 메시지 선택자를 사용하여 컨텐츠 기반 발행/구독의 중간 형식을 제공합니다. 선택기를 참조하십시오. 메시지 선택자는 전형적으로 토픽을 구독한 다음 숫자 특성에 대한 선택 사항을 규정하는 데 사용됩니다. 선택자를 사용하면 특정 범위의 값에만(문자 또는 토픽 기반 와일드카드를 사용하여 수행할 수 없는 내용) 관심이 있음을 지정할 수 있습니다. 메시지의 전체 컨텐츠를 기반으로 하여 필터링해야 하는 경우, IBM Integration Bus를 사용해야 합니다.

토픽 문자열

토픽 문자열을 사용하여 토픽으로 발행하는 레이블 정보입니다. 문자 또는 토픽 기반 와일드카드 토픽 문자열을 사용하여 토픽 그룹을 구독합니다.

토픽

토픽 문자열은 발행/구독 메시지의 토픽을 식별하는 문자열입니다. 토픽 문자열을 구성할 때 원하는 모든 문자를 사용할 수 있습니다.



3자는 IBM WebSphere® MQ 7 발행/구독에서 특별한 의미를 지닙니다. 이는 토픽 문자열에서 허용되지만, 주의해서 사용해야 합니다. 특수 문자의 사용은 62 페이지의 『토픽 기반 와일드카드 설계』에서 설명됩니다.

슬래시(/)

토픽 레벨 구분 기호입니다. '/' 문자를 사용하여 토픽 트리로 토픽을 구성합니다.

가능한 경우 빈 토픽 레벨('//')은 피하십시오. 이는 토픽 문자열이 없는 토픽 계층의 노드에 대응합니다. 토픽 문자열에서 선두 또는 후미 문자 '/'는 선두 또는 후미의 빈 노드에 대응하므로 이러한 사용도 피해야 합니다.

해시 기호(#)

'/'와 함께 사용하여 구독에서 다중 레벨 와일드카드를 구성합니다. 발행된 토픽의 이름을 지정하는 데 사용되는 토픽 문자열에서 '/' 가까이에서 '#'를 사용할 때 주의하십시오. 61 페이지의 『토픽 문자열 예제』에서는 '#'의 합리적인 사용법을 보여줍니다.

'.../#/...', '#/...', '.../#' 문자열에는 구독 토픽 문자열에서 특별한 의미를 지닙니다. 문자열은 토픽 계층에서 하나 이상의 레벨에 있는 모든 토픽과 일치합니다. 따라서 이러한 순서 중 하나로 토픽을 작성한 경우 토픽 계층의 다중 레벨에서 모든 토픽을 구독하지 않는 한, 이를 구독할 수 없습니다.

더하기 기호(+)

'/'와 함께 사용하여 구독에서 단일 레벨 와일드카드를 구성합니다. 발행된 토픽의 이름을 지정하는 데 사용되는 토픽 문자열에서 '/' 가까이에서 '+'를 사용할 때 주의하십시오.

'.../+/...', '+/...', '.../+' 문자열에는 구독 토픽 문자열에서 특별한 의미를 지닙니다. 문자열은 토픽 계층에서 한 레벨에 있는 모든 토픽과 일치합니다. 따라서 이러한 순서 중 하나로 토픽을 작성한 경우 토픽 계층의 한 레벨에서 모든 토픽을 구독하지 않는 한, 이를 구독할 수 없습니다.

토픽 문자열 예제

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

관련 참조

TOPIC

와일드카드 설계
여러 토픽을 구독하는 데 사용하는 2개의 와일드카드 설계가 있습니다. 설계 선택은 구독 옵션입니다.

MQSO_WILDCARD_TOPIC
토픽 기반 와일드카드 설계를 사용하여 구독할 토픽을 선택합니다.

이는 와일드카드 스키마를 명확하게 선택하지 않은 경우 기본값입니다.

MQSO_WILDCARD_CHAR
문자 기반 와일드카드 설계를 사용하여 구독할 토픽을 선택합니다.

DEFINE SUB 명령에서 **wschema** 매개변수를 지정하여 설계를 설정합니다. 자세한 정보는 DEFINE SUB를 참조하십시오.

참고: IBM WebSphere MQ 7.0 이전에 작성된 구독은 항상 문자 기반 와일드카드 설계를 사용합니다.

예

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

토픽 기반 와일드카드 설계
토픽 기반 와일드카드를 사용하면 구독자가 한 번에 둘 이상의 토픽을 구독할 수 있습니다.

토픽 기반 와일드카드는 IBM MQ 발행/구독에서 토픽 시스템의 강력한 기능입니다. 다중 레벨 와일드카드와 단일 레벨 와일드카드는 구독에 사용될 수 있으나, 메시지 발행자가 토픽에서 사용할 수는 없습니다.

토픽 기반 와일드카드 설계에서는 토픽 레벨로 그룹화된 발행을 선택할 수 있습니다. 토픽 계층구조의 각 레벨마다 해당 토픽 레벨에 대한 구독의 문자열이 발행의 문자열과 정확히 일치해야 하는지 여부를 선택할 수 있습니다. 예를 들어, `IBM/+/Results` 구독은 모든 토픽을 선택합니다.

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

두 종류의 와일드카드가 있습니다.

다중 레벨 와일드카드

- 다중 레벨 와일드카드는 구독에서 사용됩니다. 발행에서 사용되면 리터럴로 처리됩니다.
- 다중 레벨 와일드카드 문자 `'#'`는 토픽 내 많은 레벨과 일치시키는 데 사용됩니다. 예를 들어 토픽 트리 예제를 사용하면 `'USA/Alaska/#'`를 구독할 때 토픽 `'USA/Alaska'` 및 `'USA/Alaska/Juneau'`에서 메시지를 수신합니다.
- 다중 레벨 와일드카드는 0개 이상의 레벨을 표시할 수 있습니다. 따라서 `'USA/#'`는 단일 `'USA'`와도 일치할 수 있습니다. 여기서 `'#'` 기호는 0 레벨을 나타냅니다. 토픽 레벨 구분 기호는 분리할 레벨이 없기 때문에 이 컨텍스트에서는 의미가 없습니다.
- 다중 레벨 와일드카드는 자체에 또는 토픽 레벨 분리 문자 옆에 지정된 경우에만 유효합니다. 따라서 `'#'` 및 `'USA/#'`은 `'#'` 문자가 와일드카드로 처리되는 유효한 토픽입니다. `'USA#'`도 유효한 토픽 문자열이지만 `'#'` 문자는 와일드카드로 간주되지 않으며, 특별한 의미를 지니지 않습니다. 64 페이지의 『토픽 기반 와일드카드가 와일드카드가 아닌 경우』의 내용을 참조하십시오.

단일 레벨 와일드카드

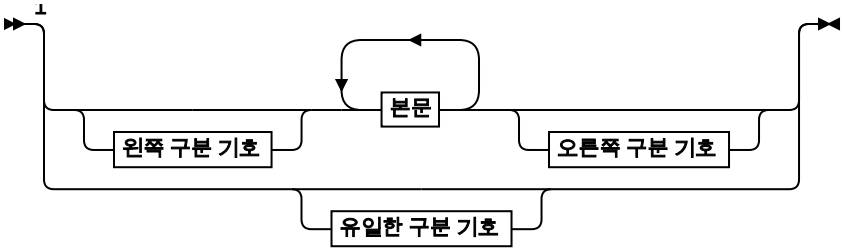
- 단일 와일드카드는 구독에서 사용됩니다. 발행에서 사용되면 리터럴로 처리됩니다.
- 단일 레벨 와일드카드 문자 `'+'`는 하나의 유일한 토픽 레벨과 일치합니다. 예를 들어 `'USA/+'`는 `'USA/Alabama/Auburn'`이 아닌 `'USA/Alabama'`와 일치합니다. 단일 레벨 와일드카드는 단일 레벨과만 일치하므로 `'USA/+'`는 `'USA'`와 일치하지 않습니다.
- 단일 레벨 와일드카드는 토픽 트리의 레벨에서 그리고 다중 레벨 와일드카드와 연결하여 사용될 수 있습니다. 단일 레벨 와일드카드는 자체에 지정된 경우를 제외하고 토픽 레벨 분리 문자 옆에 지정해야 합니다. 따

라서 '+' 및 'USA/+'은 '+' 문자가 와일드카드로 처리되는 유효한 토픽입니다. 'USA+'도 유효한 토픽 문자열이지만 '+' 문자는 와일드카드로 간주되지 않으며, 특별한 의미를 지니지 않습니다. 64 페이지의 『토픽 기반 와일드카드가 와일드카드가 아닌 경우』의 내용을 참조하십시오.

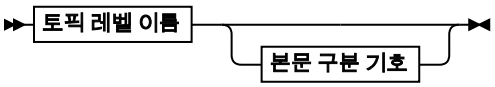
토픽 기반 와일드카드 설계의 구문에는 이스케이프 문자가 없습니다. '#' 및 '+'가 와일드카드로 처리되는지 여부는 컨텍스트에 따라 달라집니다. 자세한 정보는 64 페이지의 『토픽 기반 와일드카드가 와일드카드가 아닌 경우』의 내용을 참조하십시오.

참고: 토픽 문자열의 시작과 끝은 특별한 방식으로 처리됩니다. '$' 를 사용하여 문자열의 끝을 표시하면 '$#/...' 는 다중 레벨 와일드카드이고 '$/#/..'입니다. 루트의 빈 노드이고 다음에 다중 레벨 와일드카드가 나옵니다.

토픽 기반 와일드카드 문자열

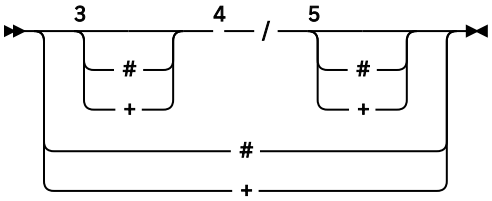


본문



토픽 레벨 이름

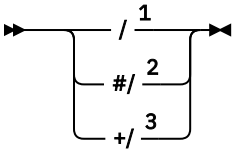
유일한 구분 기호



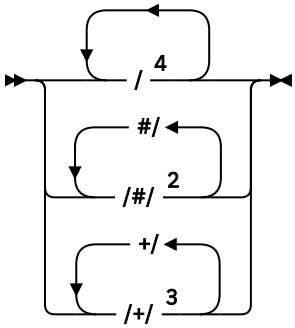
참고:

[1] 널 또는 길이가 0인 토픽 문자열은 유효하지 않음
[2] 문자 기반 및 토픽 기반 와일드카드 설계 간의 호환성을 위해 레벨 이름 문자열에서 *, ?, % 를 사용하지 않는 것이 좋습니다.
[3] 이 경우는 왼쪽 구분 기호 패턴과 동일합니다.
[4] 와일드카드가 없는 /는 단일 빈 토픽과 일치합니다.
[5] 이 경우는 오른쪽 구분 기호 패턴과 동일합니다.
[6] 모든 토픽과 일치합니다.
[7] 1개 레벨만 존재하는 모든 토픽과 일치합니다.

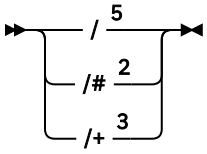
왼쪽 구분 기호



본문 구분 기호



오른쪽 구분 기호



참고:

[1] 토픽 문자열이 빈 토픽으로 시작함
[2] 0개 이상의 레벨과 일치합니다. 여러 개의 다중 레벨 일치 문자열은 하나의 다중 레벨 일치 문자열과 효과는 동일합니다.
[3] 정확히 하나의 레벨과 일치합니다.
[4] //는 빈 토픽(토픽 문자열이 없는 토픽 오브젝트)입니다.
[5] 토픽 문자열이 빈 토픽으로 끝남

토픽 기반 와일드카드가 와일드카드가 아닌 경우

와일드카드 문자 '+' 및 '#'은 토픽 레벨에서 해당 문자를 포함하여 다른 문자와 함께 혼합된 경우 특별한 의미는 없습니다.

즉 토픽 레벨에서 다른 문자와 함께 '+' 또는 '#'을 포함하는 토픽을 발행할 수 있음을 의미합니다.

예를 들어 다음 두 개의 토픽을 고려하십시오.

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

첫 번째 예제에서 '+' 및 '#' 문자는 와일드카드로 처리되므로 발행하려는 토픽 문자열에서 유효하지 않지만, 구독에서 유효합니다.

두 번째 예제에서 '+' 및 '#' 문자는 와일드카드로 처리되지 않으므로 토픽 문자열은 발행 및 구독 대상 모두일 수 있습니다.

예

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

문자 기반 와일드카드 설계
문자 기반 와일드카드 설계에서는 기존 문자 일치 방식에 기반하여 토픽을 선택할 수 있습니다.

문자열 '*'를 사용하여 토픽 계층에서 여러 레벨에 있는 모든 토픽을 선택할 수 있습니다. 문자 기반 와일드카드 설계에서 '*'를 사용하는 것은 토픽 기반 와일드카드 문자열 '#'를 사용하는 것과 같습니다.

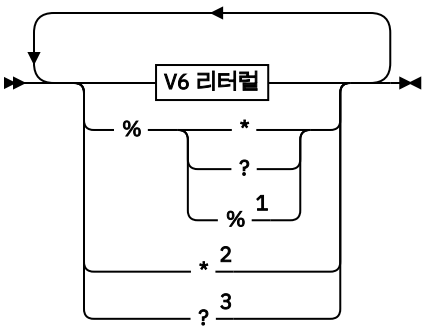
' $x/*/y$ '는 토픽 기반 스킴에서 ' $x/\#/y$ '와 동일하며 ' x 및 y '레벨 사이의 토픽 계층 구조에서 모든 토픽을 선택합니다. 여기서 ' x ' 및 ' y '은 와일드카드로 리턴되는 레벨 세트에 없는 토픽 이름입니다.

문자 기반 설계에는 토픽 기반 설계의 '/+/'와 정확히 동일한 항목이 없습니다. 'IBM/*/Results'가 'IBM/Patents/Software/Results'를 선택할 수도 있습니다. 계층의 각 레벨에서 토픽 이름 세트가 고유한 경우에만 동일한 일치를 생성하는 2개 설계를 포함하는 쿼리를 항상 구성할 수 있습니다.

일반적으로 사용되는 문자 기반 설계에서 '*' 및 '?'는 토픽 기반 설계에서 동일한 기능이 없습니다. 토픽 기반 설계는 와일드카드를 사용하는 부분 일치를 수행하지 않습니다. 문자 기반 와일드카드 구독 'IBM/*ware/Results'에는 이와 동등한 토픽 기반 항목이 없습니다.

참고: 문자 와일드카드 구독을 사용하는 일치는 토픽 기반 구독을 사용하는 일치보다 느립니다.

문자 기반 와일드카드 문자열



V6 리터럴

►► **,? 및%를 제외한 모든 유니코드 문자* ►◄

참고:

[1] 즉, 다음 문자를 이스케이프 처리하므로 리터럴로 처리됩니다. '%' 다음에는 '*', '?' 또는 '%'가 와야 합니다. 61 페이지의 『토픽 문자열 예제』를 참조하십시오.
[2] 즉, 구독에서 0개 이상의 문자와 일치합니다.
[3] 즉, 구독에서 1개 문자와 정확히 일치합니다.

예

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

토픽 문자열 결합

메시지를 구독할 수 있도록 구독을 작성하거나 토픽을 열 경우, 토픽 문자열이 두 개의 별도 하위 토픽 문자열 또는 "subtopics"를 결합하여 형식화될 수 있습니다. 한 하위 토픽은 애플리케이션 또는 관리 명령에 의해 토픽 문자열로 제공되며, 다른 하위 토픽은 토픽 오브젝트와 연관된 토픽 문자열입니다. 하위 토픽을 자체에서 토픽 문자열로 사용하거나 새 토픽 이름을 형성하도록 둘을 결합할 수 있습니다.

예를 들어, MQSC 명령 **DEFINE SUB**를 사용하여 구독을 정의할 경우 이 명령은 **TOPICSTR**(토픽 문자열) 또는 **TOPICOBJ**(토픽 오브젝트) 중 하나 또는 둘 다를 속성으로 사용할 수 있습니다. **TOPICOBJ**만 제공되면 해당 토픽 오브젝트와 연관된 토픽 문자열이 토픽 문자열로 사용됩니다. **TOPICSTR**만 제공되면 이것이 토픽 문자열로 사용됩니다. 둘 다 제공되면 **TOPICOBJ** / **TOPICSTR** 형식으로 단일 토픽 문자열을 형성하도록 병합됩니다. 여기서 **TOPICOBJ** 구성 토픽 문자열이 항상 첫 번째에 오며, 문자열의 두 부분은 항상 "/" 문자로 나뉩니다.

마찬가지로 MQI 프로그램에서 전체 토픽 이름은 MQOPEN에 의해 작성됩니다. 이는 다음에 나열된 순서대로 발행/구독 MQI 호출에서 사용되는 2개의 필드로 구성됩니다.

1. **ObjectName** 필드에 이름 지정된, 토픽 오브젝트의 **TOPICSTR** 속성.
2. 애플리케이션에서 제공하는 하위 토픽을 정의하는 **ObjectString** 매개변수.

결과로 생성되는 토픽 문자열은 **ResObjectString** 매개변수에서 리턴됩니다.

이러한 필드는 각 필드의 첫 번째 문자가 공백 또는 널 문자가 아니면 존재한다고 간주되며 필드 길이는 0보다 큽니다. 필드 중 하나만 존재하면 이는 토픽 이름으로 그대로 사용됩니다. 필드 모두에 값이 없는 경우 전체 토픽 이름이 유효하지 않으면 호출은 이유 코드 MQRC_UNKNOWN_OBJECT_NAME 또는 MQRC_TOPIC_STRING_ERROR로 실패합니다.

두 필드가 모두 있으면 "/" 문자는 결과로 생성되어 결합된 토픽 이름의 두 요소 사이에 삽입됩니다.

다음 표는 토픽 문자열 연결의 예를 표시합니다.

표 *2.* 토픽 문자열 연결 예제

토픽 오브젝트의 TOPICSTR	**애플리케이션 또는 DEFINE SUB 명령에서 제공되는 토픽 문자열**	**전체 토픽 이름**	**주석**
Football/Scores	' '	Football/Scores	토픽 오브젝트의 TOPICSTR은 단독으로 사용됩니다.
' '	Football/Scores	Football/Scores	ObjectString/TOPICSTR은 단독으로 사용됩니다.
Football	Scores	Football/Scores	"/" 문자가 연결 지점에 추가되었습니다.
Football	/Scores	Football//Scores	두 문자열 사이에 '빈 노드'가 생성됩니다. 이는 "Football/Scores"와 다릅니다.
/Football	Scores	/Football/Scores	토픽이 '빈 노드'로 시작합니다. 이는 "Football/Scores"와 다릅니다.

"/" 문자는 특수 문자로 간주되며 67 페이지의 『토픽 트리』에서 전체 토픽 이름에 대한 구조를 제공합니다. "/" 문자는 토픽 트리의 구조가 영향을 받기 때문에 다른 이유로 사용해서는 안 됩니다. 토픽 "/Football"은 토픽 "Football"과 동일하지 않습니다.

참고: 구독을 작성할 때 토픽 오브젝트를 사용하면, 토픽 오브젝트 토픽 문자열의 값이 정의 시에 구독에서 수정됩니다. 토픽 오브젝트의 후속 변경사항은 구독이 정의되는 토픽 문자열에 영향을 미치지 않습니다.

토픽 문자열의 와일드카드 문자

다음 와일드카드 문자는 특수 문자입니다.

- 더하기 부호(+)
- 숫자 부호(#)
- 별표(*)
- 물음표(?)

와일드카드 문자는 구독에서 사용될 경우에만 특별한 의미를 가집니다. 이러한 문자가 다른 곳에서 사용될 경우 적합하지 않다고 간주되지 않지만, 사용 방법을 이해하고 있어야 하며 토픽 오브젝트 발행 또는 정의 시에 토픽 문자열에 이러한 문자를 사용하지 않는 것이 좋습니다.

한 토픽 레벨 내의 다른 문자(해당 문자 포함)와 함께 # 또는 +가 결합된 토픽 문자열에서 발행할 경우, 토픽 문자열은 와일드카드 설계와 함께 구독될 수 있습니다.

두 / 문자 사이의 유일한 문자로 # 또는 +를 사용하여 토픽 문자열에서 발행할 경우, 토픽 문자열은 와일드카드 설계 MQSO_WILDCARD_TOPIC을 사용하여 애플리케이션에 의해 명시적으로 구독될 수 없습니다. 이 상황으로 인해 애플리케이션은 예상보다 많은 발행을 가져옵니다.

정의된 토픽 오브젝트의 토픽 문자열에서 와일드카드 문자를 사용해서는 안 됩니다. 이렇게 할 경우, 오브젝트가 발행자에 의해 사용될 때는 문자가 리터럴 문자로 처리되고 구독에서 사용될 경우 와일드카드 문자로 처리됩니다. 이는 혼동을 야기할 수 있습니다.

코드 스니펫 예제

예제 프로그램 예제 2: 가변 토픽에 대한 발행자에서 추출한 이 코드 스니펫은 가변 토픽 문자열에 토픽 오브젝트를 결합합니다.

```
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);
```

토픽 트리

정의한 각 토픽은 요소 또는 토픽 트리의 노드입니다. 토픽 트리는 비워 두거나 MQSC 또는 PCF 명령을 사용하여 이미 정의된 토픽으로 시작되거나 해당 토픽을 포함할 수 있습니다. 토픽 작성 명령을 사용하거나 발행 또는 구독에 토픽을 처음으로 지정하면 새 토픽을 정의할 수 있습니다.

아무 문자열이나 사용해도 토픽의 토픽 문자열을 정의할 수 있으나 계층 구조 트리에 적합한 토픽 문자열을 선택해야 합니다. 토픽 문자열 및 토픽 트리를 신중하게 디자인하면 다음과 같은 조작에 도움이 될 수 있습니다.

- 다중 토픽을 구독합니다.
- 보안 정책을 설정합니다.

토픽 트리를 평면적 선형 구조로 구성할 수도 있지만, 토픽 트리를 하나 이상의 루트 토픽이 있는 계층 구조로 빌드하는 것이 더 좋습니다. 보안 계획 및 토픽에 대한 자세한 정보는 발행/구독 보안을 참조하십시오.

67 페이지의 그림 18에서는 루트 토픽이 하나인 토픽 트리 예제를 보여줍니다.

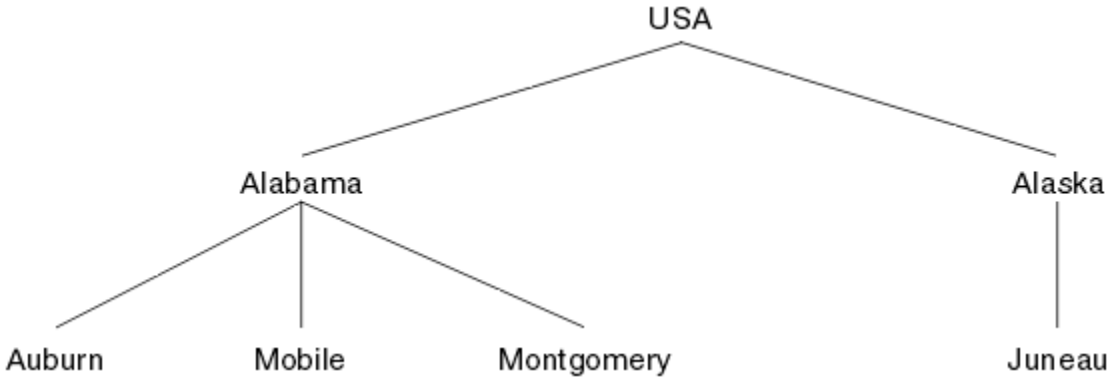


그림 *18.* 토픽 트리 예제

그림의 각 문자열은 토픽 트리의 노드를 나타냅니다. 전체 토픽 문자열은 토픽 트리에 있는 하나 이상의 레벨에서 노드를 집계하여 작성됩니다. 레벨은 "/" 문자로 구분됩니다. 지정된 전체 토픽 문자열의 형식은 "root/level2/level3"입니다.

67 페이지의 그림 18에 표시된 토픽 트리의 올바른 토픽은 다음과 같습니다.

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

"USA/Alaska/Juneau"

토픽 문자열 및 토픽 트리를 디자인할 때, 큐 관리자는 토픽 문자열의 의미를 도출하도록 시도하거나 토픽 문자열 자체를 해석하지 않습니다. 단순히 선택한 메시지를 해당 토픽의 구독자에게 보내는 데 토픽 문자열을 사용합니다.

토픽 트리의 구성 및 컨텐츠에는 다음 원칙이 적용됩니다.

- 토픽 트리의 레벨 수에 제한이 없습니다.
- 토픽 트리의 레벨 이름 길이에 제한이 없습니다.
- "루트" 노드 수에 제한이 없습니다. 즉 토픽 트리 수에 제한이 없습니다.

관련 태스크

토픽 트리에서 불필요한 토픽 수 감소

관리 토픽 오브젝트

관리 토픽 오브젝트를 사용하여 기본이 아닌 특정 속성을 토픽에 지정할 수 있습니다.

68 페이지의 그림 19에서는 서로 다른 스포츠를 포함하는 별도의 토픽으로 구분된 Sport의 상위 레벨 토픽을 토픽 트리로 시각화하는 방법을 보여줍니다.

그림 *19.* 토픽 트리의 시각화

68 페이지의 그림 20에서는 각 스포츠에 대한 여러 유형의 정보를 구분하기 위해 추가로 토픽 트리를 구분하는 방법을 보여줍니다.

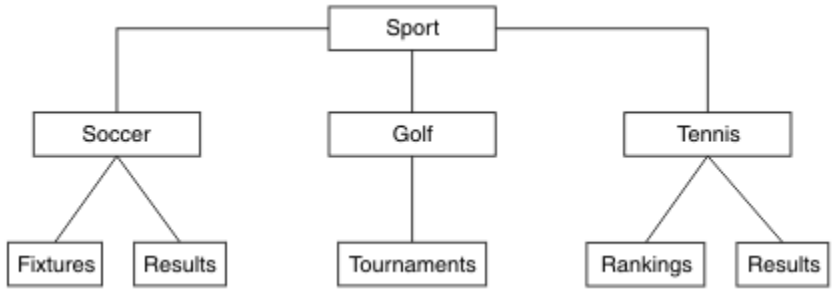


그림 *20.* 확장된 토픽 트리

표시된 토픽 트리를 작성하려는 경우 관리 토픽 오브젝트를 정의하지 않아도 됩니다. 이 트리의 각 노드는 발행 또는 구독 조작에서 작성된 토픽 문자열에 의해 정의됩니다. 트리에서 각 토픽은 해당 상위에서 속성을 상속합니다. 기본적으로 모든 속성은 ASPARENT로 설정되므로 속성은 상위 토픽 오브젝트에서 상속됩니다. 이 예제에서 모든 토픽에는 Sport 토픽과 동일한 속성이 있습니다. Sport 토픽에는 관리 토픽 오브젝트가 없으며 SYSTEM.BASE.TOPIC

토픽 트리의 루트 노드(즉, SYSTEM.BASE.TOPIC)에서 mqm이 아닌 사용자에게는 권한을 부여하지 않는 것이 좋습니다. 권한은 상속되지만 제한할 수 없기 때문입니다. 따라서 이 레벨에서 권한을 제공하면 전체 트리에 권한을 제공하게 됩니다. 따라서 계층의 더 낮은 토픽 레벨에서 권한을 제공해야 합니다.

관리 토픽 오브젝트는 토픽 트리에 있는 특정 노드에 대한 특정 속성을 정의하는 데 사용할 수 있습니다. 다음 예제에서 관리 토픽 오브젝트는 Soccer 토픽의 지속 가능 구독 특성 DURSUB를 NO 값으로 설정하도록 정의됩니다.

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

이제 토픽 트리는 다음과 같이 시각화될 수 있습니다.

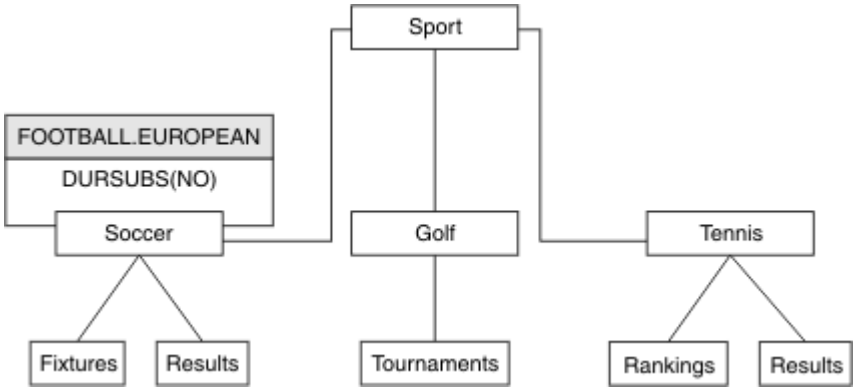


그림 *21. Sport/Soccer* 토픽과 연관된 관리 토픽 오브젝트의 시각화

트리에서 Soccer 아래 토픽을 구독하는 애플리케이션은 계속해서 관리 토픽 오브젝트를 추가하기 전에 사용하던 토픽 문자열을 사용할 수 있습니다. 그러나 이제 애플리케이션은 문자열 `/Sport/Soccer` 대신 오브젝트 이름 `FOOTBALL.EUROPEAN`을 사용하여 구독하도록 작성될 수 있습니다. 예를 들어 `/Sport/Soccer/Results`를 구독하려면 애플리케이션은 `MQSD.ObjectName`을 `FOOTBALL.EUROPEAN`으로, `MQSD.ObjectString`을 `Results`로 지정할 수 있습니다.

이 기능을 사용하면 애플리케이션 개발자로부터 토픽 트리의 일부를 숨길 수 있습니다. 토픽 트리의 특정 노드에서 관리 토픽 오브젝트를 정의하면 애플리케이션 개발자가 노드의 하위로 고유한 토픽을 정의할 수 있습니다. 개발자는 상위 토픽에 대해 알아야 하지만 상위 트리의 다른 노드에 대해서는 알지 않아도 됩니다.

속성 상속

토픽 트리에 관리 토픽 오브젝트가 많으면 기본적으로 각 관리 토픽 오브젝트는 가장 가까운 상위 관리 토픽에서 해당 속성을 상속합니다. 이전 예제는 69 페이지의 그림 22에서 확장되었습니다.

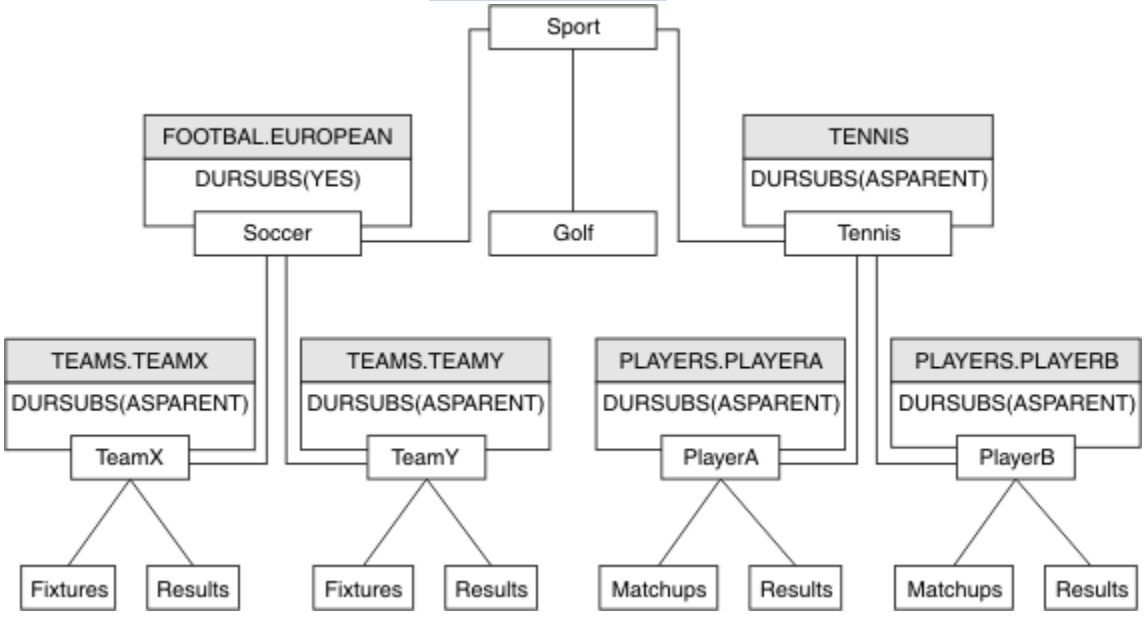


그림 *22.* 여러 관리 토픽 오브젝트를 포함하는 토픽 트리

예를 들어 상속을 사용하여 `/Sport/Soccer`의 모든 하위 토픽에 구독이 비지속인 특성을 제공합니다. `FOOTBALL.EUROPEAN` 의 `DURSUB` 속성을 `NO`로 변경하십시오.

이 속성은 다음과 같은 명령을 사용하여 설정할 수 있습니다.

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

Sport/Soccer의 하위 토픽에 대한 모든 관리 토픽 오브젝트에서는 DURSUB가 기본값, ASPARENT로 설정되어 있습니다. FOOTBALL.EUROPEAN의 DURSUB 특성 값을 NO로 변경하면 Sport/Soccer의 하위 토픽은 DURSUB 특성 값 NO를 상속합니다. Sport/Tennis의 모든 하위 토픽은 SYSTEM.BASE.TOPIC 오브젝트에서 DURSUB의 값을 상속합니다. SYSTEM.BASE.TOPIC의 값은 YES입니다.

이제 Sport/Soccer/TeamX/Results 토픽에서 지속 가능 구독을 작성하려고 하면 실패합니다. 그러나 Sport/Tennis/PlayerB/Results의 지속 가능 구독 작성에는 성공합니다.

WILDCARD 특성으로 와일드카드 사용 제어

MQSC **Topic** WILDCARD 특성 또는 동등한 PCF 토픽 WildcardOperation 특성을 사용하여 와일드카드 토픽 문자열 이름을 사용하는 구독자 애플리케이션으로 발행물의 전달을 제어할 수 있습니다. WILDCARD 특성은 다음과 같은 두 가지 가능한 값을 보유할 수 있습니다.

WILDCARD
이 토픽에 관한 와일드카드 구독의 동작입니다.

PASSTHRU
이 토픽 오브젝트의 토픽 문자열보다 덜 특정한 와일드카드 토픽에 대한 구독이 이 토픽 및 이 토픽보다 더욱 특정한 토픽 문자열에 대한 발행물을 수신합니다.

BLOCK
이 토픽 오브젝트의 토픽 문자열보다 덜 특정한 와일드카드 토픽에 대한 구독이 이 토픽 또는 이 토픽보다 더욱 특정한 토픽 문자열에 대한 발행물을 수신하지 않습니다.

이 속성의 값은 구독이 정의될 때 사용됩니다. 이 속성을 대체할 경우 기존 구독에 포함된 토픽 세트는 수정의 영향을 받지 않습니다. 이 시나리오는 토픽 오브젝트를 작성 또는 삭제할 때 토폴로지가 변경된 경우에도 적용됩니다. WILDCARD 속성을 수정한 후에 작성된 구독과 일치하는 토픽 세트가 수정된 토폴로지를 사용하여 작성됩니다. 일치하는 토픽 세트를 강제로 기존 구독에 대해 재평가하려는 경우 큐 관리자를 재시작해야 합니다.

예제, 73 페이지의 『예제: Sport 발행/구독 클러스터 작성』 에서는 70 페이지의 그림 23에 표시된 토픽 트리 구조를 작성하는 단계를 수행할 수 있습니다.

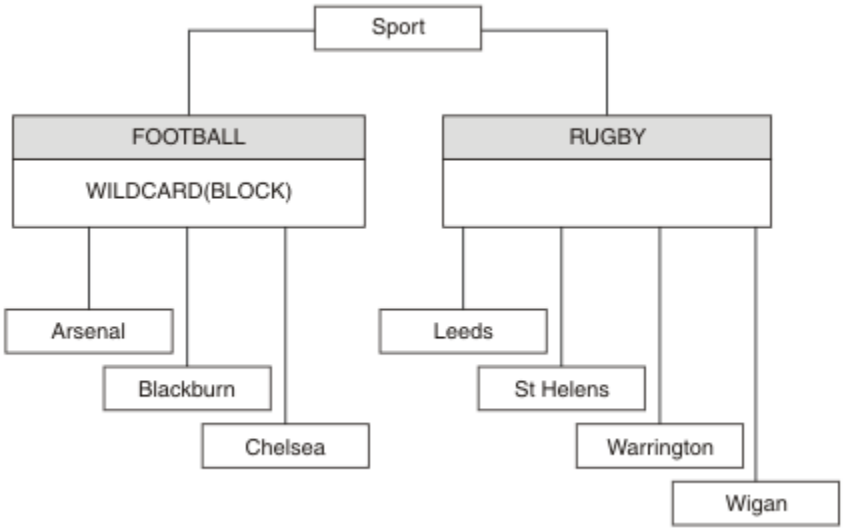


그림 *23. WILDCARD* 특성, *BLOCK*을 사용하는 토픽 트리

와일드카드 토픽 문자열 #을 사용하는 구독자는 Sport 토픽 및 Sport/Rugby 하위 트리에 대한 모든 발행을 수신합니다. 구독자는 Sport/Football 하위 트리에 대한 발행을 수신하지 않습니다. Sport/Football 토픽의 WILDCARD 특성 값은 BLOCK입니다.

PASSTHRU는 기본 설정입니다. Sport 트리의 노드에서 WILDCARD 특성 값 PASSTHRU를 설정할 수 있습니다. 노드에 WILDCARD 특성 값 BLOCK이 없으면 PASSTHRU를 설정해도 Sports 트리의 노드에서 구독자가 관찰하는 작동은 대체되지 않습니다.

예제에서 구독을 작성하여 와일드카드 설정이 전달되는 발행에 미치는 영향을 확인하십시오(75 페이지의 그림 27 참조). 일부 발행을 작성하려면 76 페이지의 그림 30에서 발행 명령을 실행하십시오.

```
pub QMA
```

그림 *24. QMA*에 발행

결과는 71 페이지의 표 3에 표시됩니다. WILDCARD 특성 값 BLOCK을 설정할 때 와일드카드를 포함하는 구독이 와일드카드 범위 내 토픽에 대한 발행을 수신하지 않도록 하는 방법에 주의하십시오.

표 *3. QMA*에서 수신된 발행

구독	토픽 문자열	수신된 발행	참고
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Sports/Football에서 WILDCARD(BLOCK)으로 차단된 Football 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby의 기본 WILDCARD는 Leeds에서 와일드카드 구독을 금지하지 않습니다.

참고:

구독에 WILDCARD 특성 값 BLOCK이 있는 토픽 오브젝트와 일치하는 와일드카드가 있다고 가정합니다. 또한 구독이 일치하는 와일드카드 오른쪽에 토픽 문자열을 포함하면 구독은 발행을 수신하지 않습니다. 차단되지 않은 발행 세트는 차단된 와일드카드 상위에 해당하는 토픽에 대한 발행입니다. BLOCK 특성 값이 설정된 토픽의 하위에 해당하는 토픽에 대한 발행은 와일드카드로 차단됩니다. 따라서 와일드카드 오른쪽에 토픽을 포함하는 구독 토픽 문자열은 일치하는 발행을 수신하지 않습니다.

WILDCARD 특성 값을 BLOCK으로 설정해도 와일드카드를 포함하는 토픽 문자열을 사용하여 구독할 수 없음을 의미하지는 않습니다. 이러한 구독은 정상적입니다. 구독이 WILDCARD 특성 값 BLOCK이 설정된 토픽 오브젝트를 포함하는 토픽과 일치하는 명시적 토픽을 보유합니다. 이는 WILDCARD 특성 값 BLOCK이 설정된 토픽의 상위 또는 하위에 해당하는 토픽에 대한 와일드카드를 사용합니다. 70 페이지의 그림 23의 예제에서 Sports/Football/#과 같은 구독은 발행을 수신할 수 있습니다.

와일드카드 및 클러스터 토픽

클러스터 토픽 정의는 클러스터의 모든 큐 관리자로 전파됩니다. 클러스터의 한 큐 관리자에서 클러스터 토픽에 대한 구독으로 인해 큐 관리자가 프록시 구독을 작성할 수 있습니다. 프록시 구독은 클러스터의 다른 모든 큐 관리자에서 작성됩니다. 클러스터 토픽과 결합해 와일드카드를 포함하는 토픽 문자열을 사용하는 구독은 작동을 예상하기 어려울 수 있습니다. 작동은 다음 예제에서 설명됩니다.

예제, 73 페이지의 『예제: Sport 발행/구독 클러스터 작성』에 설정된 클러스터의 경우 QMB에는 QMA와 동일한 구독 세트가 있지만 QMB는 발행자가 QMA에 발행한 후 발행을 수신하지 않습니다(71 페이지의 그림 24 참조). Sports/Football 및 Sports/Rugby 토픽이 클러스터 토픽이어도 fullsubs.tst에 정의된 구독은 클러스터 토픽을 참조하지 않습니다. 프록시 구독은 QMB에서 QMA로 전파되지 않습니다. 프록시 구독이 없으면 QMA에 대한 발행은 QMB로 전달되지 않습니다.

Sports/#/Leeds와 같은 일부 구독은 이 경우 Sports/Rugby에 해당하는 클러스터 토픽을 참조할 수도 있습니다. Sports/#/Leeds 구독은 실제로 토픽 오브젝트 SYSTEM.BASE.TOPIC으로 해석됩니다.

Sports/#/Leeds와 같이 구독에서 참조하는 토픽 오브젝트를 해석하는 규칙은 다음과 같습니다. 토픽 문자열을 첫 번째 와일드카드까지 자릅니다. 토픽 문자열을 왼쪽부터 스캔하여 연관된 관리 토픽 오브젝트가 있는 첫 번

째 토픽을 찾습니다. 이 토픽 오브젝트는 클러스터 이름을 지정하거나 로컬 토픽 오브젝트를 정의할 수 있습니다. 예 Sports/#/Leeds에서 잘림 이후의 토픽 문자열은 토픽 오브젝트가 없는 Sports이므로 Sports/#/Leeds 는 로컬 토픽 오브젝트인 SYSTEM.BASE.TOPIC에서 상속합니다.

클러스터 토픽을 구독하여 와일드카드 전파 방식을 변경하는 방법을 확인하려면 배치 스크립트, upsubs.bat를 실행하십시오. 스크립트는 구독 큐를 지우고 fullsubs.tst에서 클러스터 토픽 구독을 추가합니다. puba.bat를 다시 실행하여 발행 배치를 작성하십시오(71 페이지의 그림 24 참조).

72 페이지의 표 4에서는 발행이 발행된, 동일한 큐 관리자에 2개의 새 구독을 추가하는 결과를 보여줍니다. 결과는 예상한 대로입니다. 새 구독은 각각 하나의 발행을 수신하고 다른 구독에서 수신하는 발행 수는 변경되지 않습니다. 다른 클러스터 큐 관리자에서 예상치 못한 결과가 발생합니다(72 페이지의 표 5 참조).

표 *4. QMA*에서 수신된 발행

구독	토픽 문자열	수신된 발행	참고
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Sports/Football에서 WILDCARD(BLOCK)으로 차단된 Football 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby의 기본 WILDCARD는 Leeds에서 와일드카드 구독을 금지하지 않습니다.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal은 구독이 와일드카드를 포함하지 않으므로 발행을 수신합니다.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds는 모든 이벤트에서 발행을 수신합니다.

72 페이지의 표 5에서는 QMB에서 2개의 새 구독을 추가하고 QMA에서 발행하는 결과를 보여줍니다. QMB가 이러한 2개의 새 구독 없이 발행을 수신하지 않는다는 점을 상기하십시오. 예상대로 Sports/FootBall 및 Sports/Rugby 가 둘 다 클러스터 토픽이므로 두 개의 새 구독이 발행물을 수신합니다. QMB 는 Sports/Football/Arsenal 및 Sports/Rugby/Leeds 에 대한 프록시 구독을 QMA에 전달하며, 이는 QMB에 발행물을 송신합니다.

예상치 못한 결과는, 이전에 발행을 수신하지 않던 2개의 구독 Sports/# 및 Sports/#/Leeds가 이제 발행을 수신한다는 점입니다. 이유는 바로, 다른 구독에 대해 QMB로 전달되는 Sports/Football/Arsenal 및 Sports/Rugby/Leeds 발행이 이제 QMB에 연결된 모든 구독자에게 사용 가능합니다. 결과적으로 로컬 토픽 Sports/# 및 Sports/#/Leeds에 대한 구독은 Sports/Rugby/Leeds 발행을 수신합니다. Sports/#/Arsenal은 계속해서 발행을 수신하지 않습니다. Sports/Football에서 해당 WILDCARD 특성 값이 BLOCK으로 설정되었기 때문입니다.

표 *5. QMB*에서 수신된 발행

구독	토픽 문자열	수신된 발행	참고
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Sports/Football*에서 *WILDCARD(BLOCK)*에 의해 차단된 *Football* 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함

표 5. *QMB*에서 수신된 발행 *(계속)*			
구독	**토픽 문자열**	**수신된 발행**	**참고**
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Sports/Rugby*의 기본 *WILDCARD*는 *Leeds*에서 와일드카드 구독을 금지하지 않습니다.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal은 구독이 와일드카드를 포함하지 않으므로 발행을 수신합니다.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds는 모든 이벤트에서 발행을 수신합니다.

대부분의 애플리케이션에서는 한 구독이 다른 구독의 작동에 영향을 주는 것은 바람직하지 않습니다. 값이 BLOCK인 WILDCARD 특성의 한 가지 중요한 사용법은 균일하게 작동하는 와일드카드를 포함하는 동일한 토픽 문자열에 대한 구독을 작성하는 것입니다. 발행자와 동일한 큐 관리자 또는 다른 큐 관리자의 구독인지에 상관없이 구독 결과는 동일합니다.

와일드카드 및 스트림

발행/구독 API에 기록된 새 애플리케이션의 경우 * 에 대한 구독이 발행물을 수신하지 않습니다. 모든 Sport 발행을 수신하려면 Sports/* 또는 Sports/#을 구독해야 하며,하고 Business 발행도 마찬가지입니다.

큐된 기존 발행/구독 애플리케이션의 동작은 발행/구독 브로커가 IBM MQ의 이후 버전으로 마이그레이션될 때 변경되지 않습니다. **Publish**, **Register Publisher**또는 **Subscriber** 명령의 **StreamName** 특성은 스트림이 마이그레이션된 토픽의 이름에 맵핑됩니다.

와일드카드 및 구독 지점

발행/구독 API에 기록된 새 애플리케이션의 경우 마이그레이션의 효과는 * 에 대한 구독이 발행물을 수신하지 않는다는 점입니다. 모든 Sport 발행을 수신하려면 Sports/* 또는 Sports/#을 구독해야 하며,하고 Business 발행도 마찬가지입니다.

큐된 기존 발행/구독 애플리케이션의 동작은 발행/구독 브로커가 IBM MQ의 이후 버전으로 마이그레이션될 때 변경되지 않습니다. **Publish**, **Register Publisher** 또는 **Subscriber** 명령에서 **SubPoint** 특성은 구독이 마이그레이션되는 토픽 이름에 맵핑됩니다.

예제: Sport 발행/구독 클러스터 작성

다음 단계는 네 개의 큐 관리자 (두 개의 전체 저장소, CL1A 및 CL1B, 두 개의 부분 저장소, QMA 및 QMB) 가 있는 CL1클러스터를 작성합니다. 전체 저장소는 클러스터 정의만 보유하는 데 사용됩니다. QMA는 클러스터 토픽 호스트를 지정합니다. 지속 가능 구독은 QMA 및 QMB 모두에서 정의됩니다.

참고: 예제는 Windows용으로 코딩되었습니다. 다른 플랫폼에서 예제를 구성하고 테스트하도록 qmgrs.bat 작성 및 pub.bat 작성을 다시 코딩해야 합니다.

1. 스크립트 파일을 작성하십시오.
   a. topics.tst 작성
   b. wildsubs.tst 작성
   c. fullsubs.tst 작성
   d. qmgrs.bat 작성
   e. pub.bat 작성
2. qmgrs.bat 작성을 실행하여 구성을 작성하십시오.

```
qmgrs
```

70 페이지의 그림 23에서 토픽을 작성하십시오. 그림 5에서 스크립트는 클러스터 토픽 Sports/Football 및 Sports/Rugby를 작성합니다.

참고: REPLACE 옵션은 주제의 TOPICSTR 특성을 바꾸지 않습니다. TOPICSTR은(는) 다른 토픽 트리를 테스트하는 예제에서 사용되는 특성입니다. 토픽을 변경하려면 먼저 토픽을 삭제하십시오.

```
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')
```

그림 *25.* 토픽 삭제 및 작성*: topics.tst*

참고: REPLACE가 토픽 문자열을 바꾸지 않으므로 토픽을 삭제합니다.

와일드카드를 사용하여 구독을 작성하십시오. 70 페이지의 그림 23에서 토픽 오브젝트를 포함하는 토픽에 대응하는 와일드카드. 각 구독에 대한 큐를 작성하십시오. 스크립트를 실행하거나 다시 실행하면 큐가 지워지고 구독이 삭제됩니다.

참고: REPLACE 옵션은 구독의 TOPICOBJ 또는 TOPICSTR 특성을 바꾸지 않습니다. TOPICOBJ 또는 TOPICSTR은 다른 구독을 테스트하기 위해 예제에서 다양하게 사용할 수 있는 특성입니다. 변경하려면 먼저 구독을 삭제하십시오.

```
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)
```

그림 *26.* 와일드카드 구독 작성*: wildsubs.tst*

클러스터 토픽 오브젝트를 참조하는 구독을 작성하십시오.

참고:

구분 기호, /는 TOPICOBJ에서 참조하는 토픽 문자열과 TOPICSTR에서 정의하는 토픽 문자열 사이에 자동으로 삽입됩니다.

DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) 정의는 동일한 구독을 작성합니다. TOPICOBJ는 이미 정의한 토픽 문자열을 참조하는 빠른 방법으로 사용됩니다. 작성되면 구독은 더 이상 토픽 오브젝트를 참조하지 않습니다.

```
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)
```

그림 *27.* 구독 삭제 및 삭제*: fullsubs.tst*

2개의 저장소를 포함하는 클러스터를 작성하십시오. 발행 및 구독을 위해 2개의 부분 저장소를 작성하십시오. 모두 삭제하고 다시 시작하도록 스크립트를 다시 실행하십시오. 또한 스크립트는 토픽 계층 및 초기 와일드카드 구독을 작성합니다.

참고:

다른 플랫폼에 비슷한 스크립트를 작성하거나 모두 명령을 입력하십시오. 스크립트를 사용하면 모두 삭제하고 동일한 구성으로 다시 빠르게 시작할 수 있습니다.

```
@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
```

그림 *28.* 큐 관리자 작성*: qmgrs.bat*

클러스터 토픽에 구독을 추가하여 구성을 업데이트하십시오.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

그림 *29.* 구독 업데이트*: upsubs.bat*

큐 관리자에서 매개변수로 pub.bat를 실행하여 발행 토픽 문자열을 포함하는 메시지를 발행하십시오. Pub.bat 는 샘플 프로그램 **amqspub**를 사용합니다.

```
@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
```

그림 *30.* 발행*: pub.bat*

스트림 및 토픽

큐에 있는 발행/구독에는 통합된 발행/구독 모델에 존재하지 않는 발행 스트림의 개념이 내포되어 있습니다. 큐에 있는 발행/구독에서는 스트림이 여러 가지 토픽에 대한 정보 플로우를 분리하는 방법을 제공합니다. 스트림은 다른 토픽 ID에 관리적으로 맵핑될 수 있는 최상위 레벨 토픽으로 구현됩니다.

기본 스트림 SYSTEM.BROKER.DEFAULT.STREAM은 네트워크의 모든 브로커 및 큐 관리자에 대해 자동으로 설정되며 기본 스트림을 사용하기 위해 추가 구성이 필요하지 않습니다. 디폴트 스트림을 이름이 지정되지 않은 디폴트 토픽 공간으로 생각해보십시오. 기본 스트림에 발행된 토픽은 큐에 있는 발행/구독이 사용 가능한 상태에서 연결된 모든 큐 관리자가 즉시 사용할 수 있습니다. 이름 지정된 스트림은 별도의 이름 지정된 토픽 공간입니다. 이름 지정된 스트림을 사용되는 각 브로커에 정의해야 합니다.

발행자 및 구독자가 다른 큐 관리자에 있으면 브로커를 동일한 브로커 계층에 연결한 후에 발행 및 구독에서 이들 사이의 전달을 위해 추가 구성이 필요하지 않습니다. 동일한 상호 운용성이 역으로도 작용합니다.

이름 지정된 스트림

큐에 있는 발행/구독 프로그래밍 모델에서 작동하는 솔루션 설계자는 Sport로 이름 지정된 스트림에 모든 Sport 발행을 배치하도록 결정할 수 있습니다. 큐된 발행/구독이 사용 가능한 IBM MQ 에서 실행되는 큐 관리자가 스트림을 사용할 수 있도록 하려면 스트림을 수동으로 추가해야 합니다.

스트림 Sport의 Soccer/Results를 구독하는 큐에 있는 발행/구독 애플리케이션은 그대로 작동합니다. 또한 MQSUB를 사용하고 Soccer/Results 토픽 문자열을 제공하여 Sport 토픽을 구독하는 통합된 발행/구독 애플리케이션은 동일한 발행을 수신합니다.

스트림 추가 태스크는 스트림 추가 주제에서 설명됩니다. 스트림을 수동으로 추가해야 하는 두 가지 이유가 있습니다.

1. 통합된 발행/구독 MQI 인터페이스로 애플리케이션을 마이그레이션하는 대신, 이후 버전 큐 관리자에서 실행되는 큐에 있는 발행/구독 애플리케이션을 계속 개발합니다.
2. 스트림을 토픽에 디폴트 맵핑하면 토픽 공간에 "충돌"이 발생하고 스트림의 발행에 다른 곳의 발행과 동일한 토픽 문자열이 생깁니다.

권한

기본적으로 토픽 트리의 루트에는 여러 토픽 오브젝트 (SYSTEM.BASE.TOPIC , SYSTEM.BROKER.DEFAULT.STREAM및 SYSTEM.BROKER.DEFAULT.SUBPOINT) 가 있습니다. 권한(예: 발행 또는 구독용 권한)은 SYSTEM.BASE.TOPIC의 권한으로 판별됩니다.
SYSTEM.BROKER.DEFAULT.STREAM 또는 SYSTEM.BROKER.DEFAULT.SUBPOINT의 권한은 무시됩니다.
SYSTEM.BROKER.DEFAULT.STREAM 또는 SYSTEM.BROKER.DEFAULT.SUBPOINT가 삭제되고 비어 있지 않은 토픽 문자열로 다시 작성되면 해당 오브젝트에 정의된 권한은 보통 토픽 오브젝트와 같은 방식으로 사용됩니다.

스트림과 토픽 간의 맵핑

큐된 발행/구독 스트림은 큐를 작성하고 스트림과 동일한 이름을 제공하여 IBM MQ 에서 모방됩니다. 때로 큐를 스트림 큐라 부릅니다. 스트림 큐가 큐에 있는 발행/구독 애플리케이션에 이와 같이 표시되기 때문입니다.
SYSTEM.QPUBSUB.QUEUE.NAMELIST라는 특수 이름 목록에 큐를 추가해서 발행/구독 엔진에 큐가 식별됩니다. 추가적인 특수 큐를 이름 목록에 추가해서 필요한 만큼 스트림을 추가할 수 있습니다. 마지막으로 토픽에 발행 및 구독할 수 있도록 스트림과 동일한 이름의 토픽 및 스트림 이름과 동일한 토픽 문자열을 추가할 필요가 있습니다.

그러나 예외적인 상황에서 스트림에 대응하는 토픽에 토픽을 정의할 때 선택한 토픽 문자열을 제공할 수 있습니다. 토픽 문자열의 목적은 토픽 공간에서 토픽에 고유한 이름을 제공하는 것입니다. 일반적으로 스트림 이름은 이 목적을 완벽하게 수행합니다. 때때로 스트림 이름과 기존 토픽 이름이 충돌합니다. 문제점을 해결하려면 스트림과 연관된 토픽에 대해 다른 토픽 문자열을 선택하십시오. 토픽 문자열을 선택하고 고유한지 확인하십시오.

토픽 정의에 정의된 토픽 문자열에는 MQOPEN 또는 MQSUB MQI 호출을 사용하여 발행자 및 구독자가 제공한 토픽 문자열에 정상적인 방식으로 접두부가 붙습니다. 토픽 오브젝트를 사용하여 토픽을 참조하는 애플리케이션은 접두부 토픽 문자열의 선택에 의해 영향을 받지 않습니다. 따라서 토픽 공간에서 발행을 고유하게 만드는 임의의 토픽 문자열을 선택할 수 있습니다.

다른 스트림을 다른 토픽으로 다시 맵핑하는 작업은 한 토픽 세트를 다른 토픽 세트와 완전히 분리하기 위해 고유하도록 토픽 문자열에 사용되는 접두부에 따라 다릅니다. 맵핑이 작동하도록 엄격하게 지켜지는 보편적인 토픽 이름 지정 규칙을 정의해야 합니다.

IBM MQ에서 접두부 메커니즘을 사용하여 토픽 문자열을 토픽 영역의 다른 위치로 다시 맵핑합니다.

참고: 스트림을 삭제할 때에는 먼저 스트림의 모든 구독을 삭제하십시오. 이 조치는 브로커 계층의 기타 브로커에서 구독이 생성되는 경우 가장 중요합니다.

구독 지점 및 토픽

이름 지정된 구독 지점은 토픽 및 토픽 오브젝트에 의해 에뮬레이트됩니다.

수동으로 구독 지점을 추가하려면 구독 지점 추가를 참조하십시오.

IBM MQ의 구독 지점

IBM MQ는 IBM MQ 토픽 트리 내의 다른 토픽 공간에 구독 지점을 맵핑합니다. 구독 지점이 없는 명령 메시지의 토픽은 IBM MQ 토픽 트리의 루트에 변경되지 않은 상태로 맵핑되며 SYSTEM.BASE.TOPIC에서 특성을 상속합니다.

구독 지점을 포함하는 명령 메시지는 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST의 토픽 오브젝트 목록을 사용하여 처리됩니다. 명령 메시지의 구독 지점 이름은 목록에 있는 각 토픽 오브젝트에 대한 토픽 문자열과 비교하여 일치됩니다. 일치 항목을 찾으면 구독 지점 이름이 토픽 문자열에 토픽 노드로 추가됩니다. 토픽은 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST에 있는 연관된 토픽 오브젝트에서 해당 특성을 상속합니다.

구독 지점 사용 효과는 각 구독 지점에 대해 별도의 토픽 공간을 작성하는 것입니다. 토픽 공간은 구독 지점과 이름이 같은 토픽에 기반을 둡니다. 각 토픽 공간의 토픽은 구독 지점과 이름이 같은 토픽 오브젝트에서 해당 속성을 상속합니다.

일치하는 토픽 오브젝트에 설정되지 않은 특성은 일반적인 방식으로 SYSTEM.BASE.TOPIC에서 상속됩니다.

MQRFH2 메시지 헤더를 사용하는 기존의 큐된 발행/구독 애플리케이션은 Publish 또는 Register subscriber 명령 메시지에서 **SubPoint** 특성을 설정하여 계속 작동합니다. 구독 지점은 명령 메시지에서 토픽 문자열과 결합되며 결과로 생성되는 토픽은 다른 것과 마찬가지로 처리됩니다.

IBM MQ 애플리케이션은 구독 지점의 영향을 받지 않습니다. 애플리케이션이 일치하는 토픽 오브젝트 중 하나에서 상속되는 토픽을 사용하는 경우, 해당 애플리케이션은 일치하는 구독 지점을 사용하여 큐된 애플리케이션과 상호 운용됩니다.

예

기존 WebSphere Message Broker (현재는 IBM Integration Bus 로 알려짐) IBM MQ 로 마이그레이션된 발행/구독 애플리케이션이 해당 토픽 문자열 'GBP' 및 'USD'를 사용하여 두 개의 토픽 오브젝트 GBP 및 USD를 작성했습니다.

구독 지점 USD를 사용하며 IBM MQ에서 실행되도록 마이그레이션된 토픽 NYSE/IBM/SPOT에 대한 기존 발행자는 토픽 USD/NYSE/IBM/SPOT에 대한 발행물을 작성합니다. 마찬가지로 구독 지점 USD를 사용하는 NYSE/IBM/SPOT에 대한 기존 구독자는 USD/NYSE/IBM/SPOT에 대한 구독을 작성합니다.

MQSUB를 호출하여 IBM MQ 발행/구독 프로그램에서 달러 현물 가격을 구독하십시오. 'C' 코드 단편에서 보여준 대로 USD 토픽 오브젝트 및 토픽 문자열 'NYSE/IBM/SPOT'을 사용하여 구독을 작성하십시오.

```
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. 클러스터 토픽 호스트에서 USD 및 GBP 토픽 오브젝트의 CLUSTER 속성을 설정하십시오.
2. 클러스터의 다른 큐 관리자에서 USD 및 GBP 토픽 오브젝트의 모든 사본을 삭제하십시오.
3. USD 및 GBP가 클러스터의 모든 큐 관리자에 있는 SYSTEM.QPUBSUB.SUBPOINT.NAMELIST에 정의되었는지 확인하십시오.

단일 큐 관리자 발행/구독 구성 예제

78 페이지의 그림 31에서는 기본적인 단일 큐 관리자 발행/구독 구성을 보여줍니다. 예제에서는 뉴스 서비스에 대한 구성을 보여줍니다. 여기에서 발행자로부터 여러 토픽에 대한 정보를 사용할 수 있습니다.

- 발행자 1은 Sport 토픽을 사용하여 스포츠 결과에 대한 정보를 발행함
- 발행자 2는 Stock 토픽을 사용하여 주가에 대한 정보를 발행함
- 발행자 3은 Films 토픽을 사용하여 영화 비평에 대한 정보를 발행하고 TV 토픽을 사용하여 TV 프로그램에 대한 정보를 발행함

3명의 구독자는 서로 다른 토픽을 관심사로 등록했으므로 큐 관리자는 관심이 있는 정보를 해당 구독자에게 송신합니다.

- 구독자 1은 스포츠 결과와 주가를 수신함
- 구독자 2는 영화 비평을 수신함
- 구독자 3은 스포츠 결과를 수신함

TV 프로그램을 관심사로 등록한 구독자는 없으므로 이는 배포되지 않습니다.

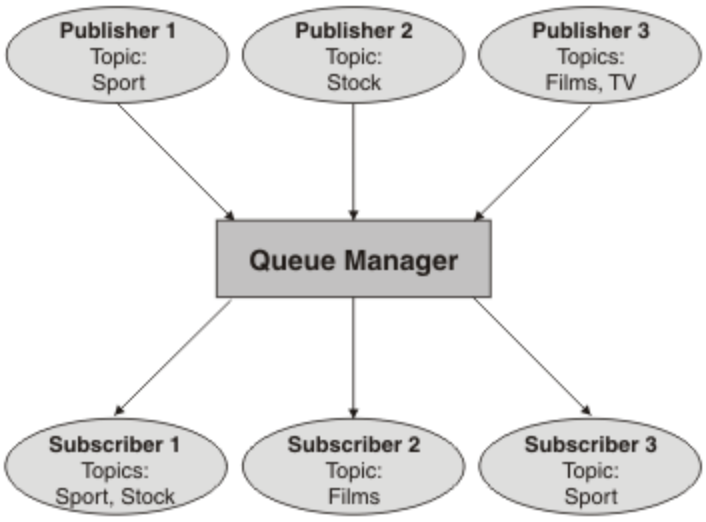


그림 *31.* 단일 큐 관리자 발행/구독 예제

분산 발행/구독 네트워크

각 큐 관리자는 해당 토픽에 가입한 로컬로 구성된 구독을 가진 토픽에 게시된 메시지와 일치합니다. 한 큐 관리자에 연결된 애플리케이션에 의해 발행된 메시지가 네트워크에서 다른 큐 관리자에 작성된 일치하는 구독으로 전달되도록 큐 관리자의 네트워크를 구성할 수 있습니다. 이를 위해서는 큐 관리자 간에 단순 채널에 대한 추가 구성이 필요합니다.

배포된 발행/구독 구성은 함께 연결된 일련의 큐 관리자입니다. 큐 관리자는 모두 동일한 물리적 시스템에 있거나 몇몇 물리적 시스템에 분산되어 있을 수 있습니다. 함께 큐 관리자를 연결하는 경우, 구독자는 하나의 큐 관리자를 구독하여 다른 큐 관리자에게 초기에 발행된 메시지를 수신할 수 있습니다. 이를 설명하기 위해 다음 그림에서는 두 번째 큐 관리자를 78 페이지의 『단일 큐 관리자 발행/구독 구성 예제』에 설명된 구성에 추가합니다.

- 발행자 4는 큐 관리자 2를 사용하여 일기 예보 정보(날씨 토픽 사용) 및 주요 도로의 트래픽 상황에 대한 정보(트래픽 토픽 사용)를 발행합니다.
- 구독자 4도 이 큐 관리자를 사용하며, 트래픽 토픽을 사용하여 트래픽 상황에 대한 정보를 구독합니다.
- 구독자 3은 발행자와 다른 큐 관리자를 사용하긴 하지만 역시 날씨 상황에 대한 정보를 구독합니다. 큐 관리자가 서로 링크되어 있기 때문에 가능합니다.

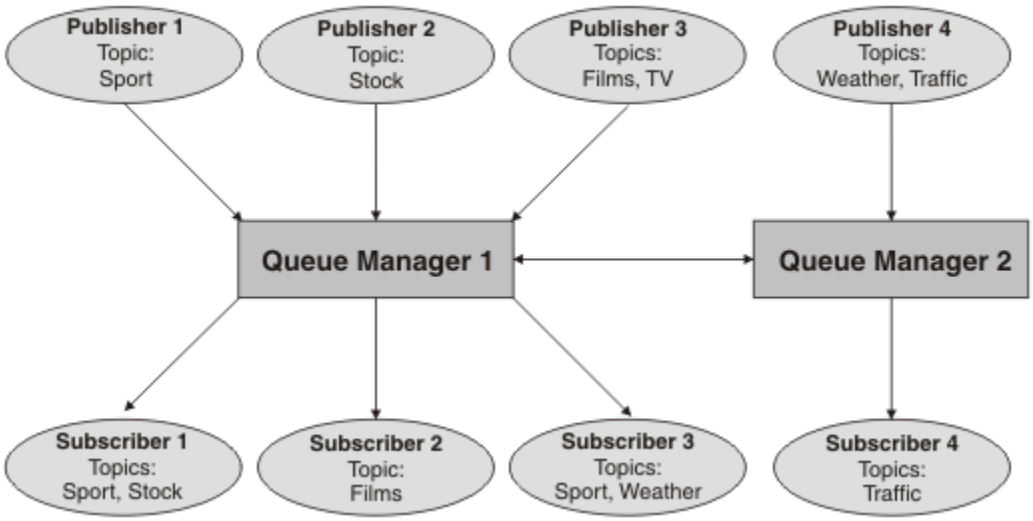


그림 *32.* 큐 관리자가 *2*개인 발행/구독 예제

상위 및 하위 계층에서 큐 관리자를 수동으로 연결하거나 발행/구독 클러스터를 작성하고 IBM MQ에서 대부분의 연결 세부사항을 정의하도록 할 수 있습니다. 또한 계층에서 여러 클러스터를 함께 조인하여 두 토폴로지를 함께 결합하여 사용할 수 있습니다.

발행/구독 클러스터 개요

발행/구독 클러스터는 클러스터에 추가된 하나 이상의 토픽 오브젝트가 있는 표준 클러스터입니다. 클러스터의 큐 관리자에 관리 토픽 오브젝트를 정의하고 클러스터 이름을 지정하여 클러스터된 해당 토픽 오브젝트를 작성할 경우, 해당 토픽에 대한 발행자 및 구독자는클러스터의 큐 관리자에 연결할 수 있으며 발행된 메시지는 큐 관리자 간의 클러스터 채널을 통해 구독자로 라우팅됩니다.

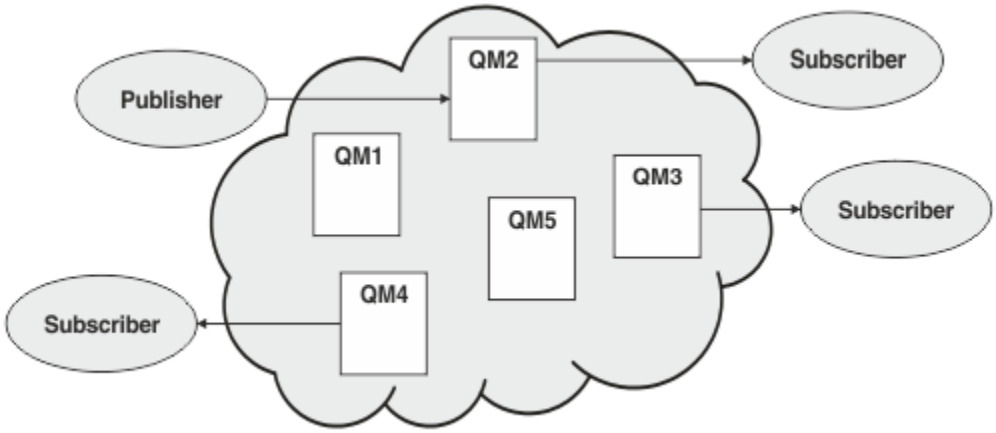


그림 *33.* 발행/구독 클러스터

발행/구독 메시지가 클러스터에서 라우팅되는 방법을 구성하는 데는 2가지 방법이 있습니다.

- 직접 라우팅(direct routing)
- 토픽 호스트 라우팅(topic host routing)

직접 라우팅된 클러스터 토픽을 구성할 경우, 한 큐 관리자에 발행된 메시지는 직접 해당 큐 관리자에서 클러스터의 다른 큐 관리자에 있는 모든 구독으로 보내집니다. 이는 구독에 대한 가장 직접적인 경로를 제공할 수 있지만 실제로는 클러스터의 모든 큐 관리자가 모든 다른 큐 관리자를 인식하게 되며 잠재적으로 이들 간에 설정된 클러스터 채널을 가지게 됩니다.

토픽 호스트 라우팅을 사용할 경우, 한 큐 관리자에 발행된 메시지는 관리 토픽 오브젝트의 정의를 호스팅하는 큐 관리자로 보내집니다. 토픽 호스트 큐 관리자는 클러스터의 다른 큐 관리자에 있는 모든 구독에 메시지를 라우팅합니다. 발행자 또는 구독자가 토픽 호스트 큐 관리자에 없는 경우, 발행에 대한 라우트가 더 길어집니다. 그러나 이점은 토픽 호스트 큐 관리자가 클러스터의 모든 다른 큐 관리자를 인식하게 되므로 잠재적으로 이와 함께 설정된 클러스터 채널을 갖게 된다는 점입니다.

자세한 정보는 81 페이지의 『발행/구독 클러스터』의 내용을 참조하십시오.

발행/구독 계층 개요

발행/구독 계층은 계층로 채널에 의해 연결된 일련의 큐 관리자입니다. 각 큐 관리자는 발행/구독 계층에 큐 관리자 연결에 설명된 대로 해당 상위 큐 관리자를 식별합니다.

토픽에 대한 발행자 및 구독자는 계층에서 큐 관리자에 연결할 수 있으며 메시지는 계층 큐 관리자 연결성을 사용하여 이들 간을 플로우됩니다.

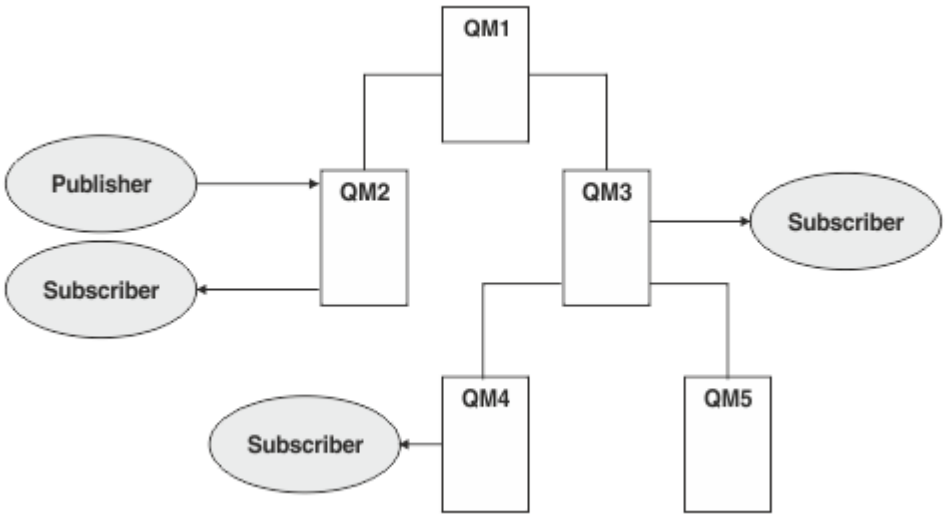


그림 *34.* 발행/구독 계층

앞의 그림에서 QM3 및 QM4의 구독자에 전달된 발행은 QM2에서 QM1로,그 다음은 QM3으로, 그리고 마지막으로 QM4로 라우팅되었습니다.

계층을 통해 사용자는 계층 내의 모든 큐 관리자 간의 관계를 직접 제어할 수 있습니다. 이는 발행자로부터 구독자로의 메시지 라우팅에 대한 보다 상세한 제어를 허용하며, 특히 제한된 연결성을 가진 큐 관리자 네트워크 간의 라우팅 시에 더 유용합니다. 메시지를 발행자에서 구독자로 라우팅하는 방법을 통해 모든 큐 관리자의 가용성과 기능을 신중하게 고려해야 합니다.

자세한 정보는 83 페이지의 『발행/구독 계층』의 내용을 참조하십시오.

큐 관리자 간의 발행 분배

라우팅 선택 외에도 큐 관리자의 네트워크를 통해 발행을 분배하기 위한 두 가지 접근법이 있습니다.

- 한 큐 관리자에서 해당 발행에 대한 구독을 현재 호스팅하는 큐 관리자에게만 발행을 송신합니다.

- 모든 큐 관리자에 각 발행을 송신하고 해당 구독에 대해 일치하도록 합니다.

전자는 발행 메시지가 필요한 곳으로만 보내지도록 하지만 큐 관리자 간에 구독 알림 레벨을 공유해야 합니다. 후자는 구독 알림을 공유할 필요는 없지만 불필요한 발행 메시지가 큐 관리자 간에 보내질 수 있습니다.

기본적으로 IBM MQ는 전자의 메소드를 사용하며, 여기서는 발행이 이에 대한 구독을 가진 큐 관리자로만 보내집니다. 구독 알림은 프록시 구독 양식으로 큐 관리자 간에 전파됩니다. 이는 배포된 발행/구독 토폴로지에서 사용하기에 가장 효율적이이므로 구독의 배포와 수명 주기, 발행의 빈도에 따라 달라집니다. 발행/구독 네트워크에서의 구독 성능을 참조하십시오.

관련 개념

67 페이지의 『토픽 트리』
정의한 각 토픽은 요소 또는 토픽 트리의 노드입니다. 토픽 트리는 비워 두거나 MQSC 또는 PCF 명령을 사용하여 이미 정의된 토픽으로 시작되거나 해당 토픽을 포함할 수 있습니다. 토픽 작성 명령을 사용하거나 발행 또는 구독에 토픽을 처음으로 지정하면 새 토픽을 정의할 수 있습니다.

발행/구독 계층 시나리오

관련 태스크

발행/구독 클러스터 디자인

발행/구독 클러스터

발행/구독 클러스터는 발행이 발행 애플리케이션에서 클러스터의 큐 관리자에 존재하는 구독으로 자동 이동되는 상호 연결된 큐 관리자의 표준 클러스터입니다. 발행/구독 클러스터에 대한 발행물 라우팅에는 직접 라우팅과 토픽 호스트 라우팅이라는 두 가지 옵션이 있습니다. 사용자가 선택하는 라우팅은 사용자 클러스터에 대한 크기와 예상 활동 패턴에 따라 달라집니다.

메시징 발행/구독에 사용되는 클러스터는 표준 IBM MQ 클러스터와 다르지 않습니다. 마찬가지로 발행/구독 클러스터 내 큐 관리자는 실제로 별도의 컴퓨터에 존재할 수 있으며, 큐 관리자의 각 쌍은 필요한 경우 클러스터 채널에 의해 자동으로 함께 연결됩니다. 자세한 정보는 클러스터를 참조하십시오.

발행/구독 메시징에 대한 큐 관리자의 표준 클러스터를 구성하기 위해 클러스터의 큐 관리자에 하나 이상의 관리 토픽 오브젝트를 정의합니다. 토픽을 클러스터 토픽으로 설정하려는 경우, 클러스터의 이름이 있는 **CLUSTER** 특성을 구성합니다. 이를 수행할 경우, 토픽 트리의 해당 지점 또는 그 아래에 있는 발행자 또는 구독자에서 사용된 토픽이 클러스터의 모든 큐 관리자 간에 공유되며 토픽 트리의 클러스터 분기에 발행된 메시지는 클러스터의 다른 큐 관리자에 대한 구독으로 자동 라우팅됩니다.

대상 큐 관리자에 있는 메시지에 대한 구독자 수에 관계없이 각 메시지의 사본 하나만 발행자 큐 관리자와 각각 다른 큐 관리자 간에 보내집니다. 하나 이상의 구독을 가진 큐 관리자에 도착할 경우 메시지가 모든 구독에 대해 중복됩니다.

클러스터에 조인하는 큐 관리자는 자동으로 클러스터된 토픽을 알게 되며 해당 큐 관리자의 발행자와 구독자는 자동으로 클러스터에 참여합니다.

클러스터된 토픽 오브젝트에 속하지 않은 토픽 문자열에 대한 작업을 수행하면 클러스터되지 않은 발행/구독 활동이 발행/구독 클러스터에서 수행될 수도 있습니다.

발행/구독 클러스터에 대한 발행물 라우팅에는 직접 라우팅과 토픽 호스트 라우팅이라는 두 가지 옵션이 있습니다. 클러스터에서 사용할 메시지 라우팅을 선택하려면 관리 토픽 오브젝트의 **CLROUTE** 특성을 다음 값 중 하나로 설정합니다.

- **DIRECT**
- **TOPICHOST**

기본적으로 토픽 라우팅은 **DIRECT**입니다. 큐 관리자에서 직접 라우트되는 클러스터 토픽을 구성하는 경우, 클러스터의 모든 큐 관리자는 클러스터의 다른 모든 큐 관리자를 인식하게 됩니다. 따라서 발행 및 구독 조작을 수행할 경우 큐 관리자가 각각 클러스터에 있는 다른 큐 관리자에 직접 연결될 수 있습니다.

IBM MQ 8.0부터는 대신 토픽 라우팅을 **TOPICHOST**로 구성할 수 있습니다. 토픽 호스트 라우팅을 사용할 경우, 클러스터의 모든 큐 관리자가 라우팅되는 토픽 정의를 호스팅하는 클러스터 큐 관리자(토픽 오브젝트를 정의한 큐 관리자)를 인식하게 됩니다. 발행 및 구독 조작을 수행할 경우, 클러스터의 큐 관리자는 서로 직접 연결되지 않고 이러한 토픽 호스트 큐 관리자에만 연결됩니다. 토픽 호스트 큐 관리자는 구독이 일치하는 큐 관리자에 발행물을 발행하는 큐 관리자에서 발행물을 라우팅하는 작업을 담당합니다.

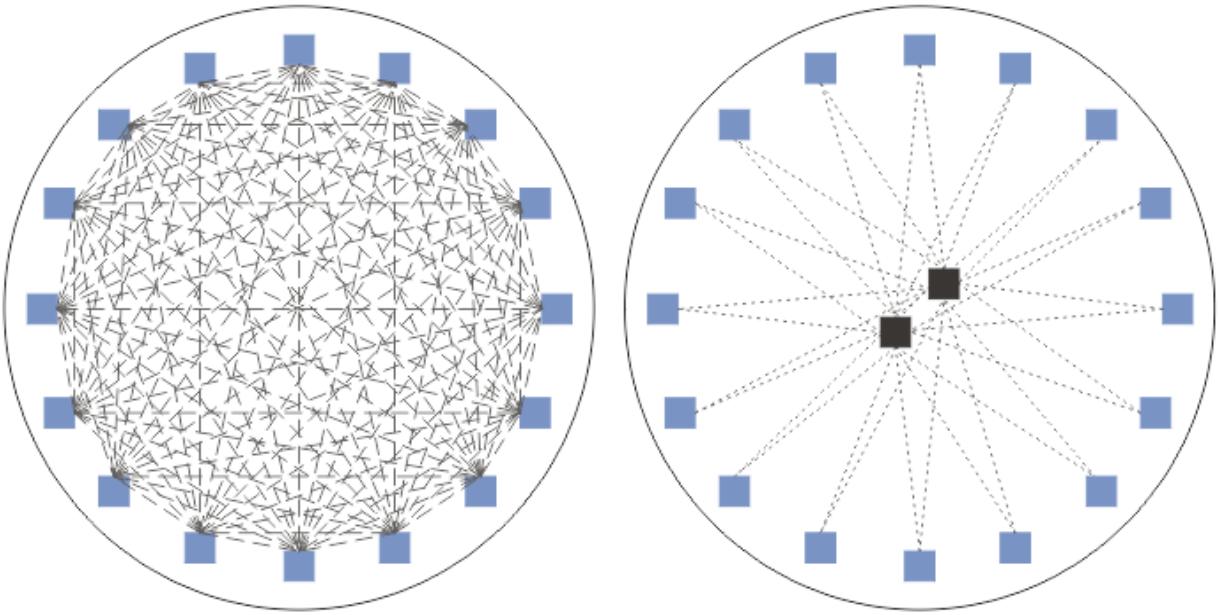


그림 *35.* 직접 라우팅 및 토픽 호스트 라우팅

직접 라우팅 개요

관리 토픽 오브젝트가 직접 라우팅용으로 구성된 경우 이에 대해 파악하려면 모든 큐 관리자에 대해 클러스터의 큐 관리자 중 하나에 토픽 오브젝트를 정의하기만 하면 됩니다. 토픽이 정의되는 큐 관리자 선택은 토픽에 대한 발행/구독 메시징의 작동에는 영향을 주지 않습니다.

각 메시지는 중간 큐 관리자를 통해 전달되는 것이 아니라 발행자 큐 관리자로부터 클러스터의 다른 큐 관리자에 있는 각 구독으로 직접 플로우됩니다.

기본적으로, 메시지는 하나 이상의 구독을 호스팅하는 클러스터의 다른 큐 관리자로만 송신됩니다.

- 이는 현재 하나 이상의 구독이 있는 모든 토픽의 클러스터에 있는 모든 다른 큐 관리자에게 직접 알리는 각 큐 관리자에 달려 있습니다. 결과적으로 클러스터의 모든 큐 관리자는 구독 중인 모든 토픽을 알게 되며 구독을 호스팅하는 큐 관리자는 다른 모든 큐 관리자에 채널을 설정하게 됩니다. 이는 각 큐 관리자가 발행자를 갖고 있는지와는 별개입니다.
- 모든 큐 관리자의 각 개별 구독 토픽에 대한 알림은 구독이 있는지에 상관없이 모든 발행을 클러스터의 모든 큐 관리자에 보내는 모델로 변경함으로써 제거할 수 있습니다. 이는 구독 알림 트래픽을 줄여주지만 발행 트래픽과 각 큐 관리자가 설정하는 채널 수는 늘어날 수 있습니다. 발행/구독 네트워크에서의 구독 성능을 참조하십시오.

직접 라우팅된 클러스터 토픽을 사용하는 발행/구독 메시지는 각 클러스터의 한 큐 관리자를 발행/구독 계층에 추가함으로써 다중 발행/구독 클러스터를 확장할 수 있습니다. 다중 클러스터의 토픽 공간 결합을 참조하십시오.

직접 라우팅에 대한 자세한 탐색을 위해서는 발행/구독 클러스터의 직접 라우팅을 참조하십시오.

토픽 호스트 라우팅 개요

관리 토픽 오브젝트가 토픽 호스트 라우팅용으로 구성된 경우, 클러스터의 큐 관리자로부터의 발행은 토픽 오브젝트가 구성된 큐 관리자("토픽 호스트")를 통해 구독이 존재하는 큐 관리자로 라우팅됩니다.

- 이는 현재 하나 이상의 구독이 있는 모든 토픽의 모든 토픽 호스트를 알리는 각 큐 관리자에 달려있습니다. 구독을 호스팅하는 큐 관리자는구독이 관련되어 있는 토픽에 대한 모든 토픽 호스트에 채널을 설정합니다.
- 비토픽 호스팅 큐 관리자는 발행/구독을 목적으로 클러스터에서 다른 비토픽 호스팅 큐 관리자를 인식하도록 되어 있지 않으며, 채널이 이 목적을 위해 이들 간에 설정되지 않습니다.
- 발행 애플리케이션이 해당 토픽을 호스팅하는 큐 관리자에 연결되어 있는 경우, 발행된 메시지는 추가 '홉(Hop)' 없이도 일치하는 구독이 작성된 큐 관리자에 직접 라우팅됩니다. 마찬가지로 일치하는 구독이 토픽을

호스팅하는 큐 관리자에서만 작성되면 해당 토픽을 구독한 메시지는 추가 홉 없이 큐 관리자에 직접 라우팅됩니다.

- 발행자와 동일한 큐 관리자에 있는 구독은 토픽 오브젝트의 호스트에 대한 첫 발행물 라우팅 없이도 충족됩니다.

클러스터된 큐의 경우 다중 큐 관리자가 동일한 관리 토픽 오브젝트를 구성할 수 있습니다. 이는 더 높은 메시지 라우팅 가용성과 워크로드 밸런스를 통한 수평적 확장을 제공합니다. 토픽 오브젝트를 라우팅한 토픽 호스트의 경우, 다중 큐 관리자가 토픽 트리의 동일한 분기에 대해 동일한 이름의 토픽을 구성할 경우, 각 토픽 호스트는 구독을 호스팅하는 모든 큐 관리자에 의해 구독되는 토픽을 파악할 수 있게 됩니다.

- 메시지가 발행되면 이는 큐 관리자를 호스팅하는 구독에 전달하기 위해 토픽 호스트 큐 관리자 중 하나로 보내집니다. 토픽 호스트 큐 관리자의 선택은 클러스터된 포인트-투-포인트 큐에 대해서와 동일한 기본 워크로드 밸런스 규칙을 따릅니다.
- 하나 이상의 토픽 호스트 큐 관리자를 발행 큐 관리자에 의해 연결하지 못할 경우 메시지가 남아 있는 사용 가능한 토픽 호스팅 큐 관리자에 라우팅됩니다.

토픽 트리의 라우팅된 분기에 있는 토픽에 대한 모든 발행은 클러스터에 해당 토픽에 대한 구독이 없는 경우에도 토픽 호스트 중 하나로 전달됩니다. 기본적으로, 메시지는 여기에서 하나 이상의 구독을 호스팅하는 클러스터의 다른 큐 관리자로만 송신됩니다.

- 이는 클러스터의 각 큐 관리자에서 모든 구독된 토픽 문자열에 대한 알림을 받는 각 토픽 호스트 큐 관리자에 따라 달라집니다.
- 각 개별 구독 토픽에 대한 알림은 구독이 있는지에 상관없이 토픽 호스트에 대해 라우팅된 모든 구독을 클러스터의 모든 큐 관리자에 보내는 모델로 변경함으로써 제거할 수 있습니다. 이는 구독 알림 트래픽을 줄여주지만 발행 트래픽과 잠재적으로 각 토픽 호스팅 큐 관리자와 설정된 채널 수는 늘어날 수 있습니다. 발행/구독 네트워크에서의 구독 성능을 참조하십시오.

클러스터된 토픽을 라우팅한 토픽 호스트를 사용한 발행/구독 메시지 플로우는 발행/구독 계층의 사용을 통해 다중 발행/구독 클러스터를 확장할 수 **없습니다**.

토픽 호스트 라우팅에 대한 자세한 탐색을 위해서는 발행/구독 클러스터의 토픽 호스트 라우팅을 참조하십시오.

발행/구독 계층

채널을 사용하여 큐 관리자를 함께 링크하고 큐 관리자 쌍 간의 하위-상위 관계를 정의하여 발행/구독 계층을 빌드합니다. 메시지는 계층에서 직접 관계를 통해 발행자에서 구독으로 플로우됩니다. 이는 다중 "홉"을 의미할 수 있습니다.

대상 큐 관리자의 메시지에 대한 구독자 수에 상관없이 큐 관리자의 한 쌍 간에는 하나의 메시지 사본만 보내집니다. 하나 이상의 구독을 가진 큐 관리자에 도착할 경우 메시지가 모든 구독에 대해 중복됩니다.

기본적으로, 메시지는 다른 큐 관리자의 구독에 대한 라우트에 있는 계층의 다른 큐 관리자로만 송신됩니다.

- 이는 이 큐 관리자 또는 다른 관계 중 하나에서 현재 하나 이상의 구독을 갖고 있는 모든 토픽의 각 직접 관계를 알리는 각 큐 관리자에 따라 달라집니다. 이렇게 되면 계층의 모든 큐 관리자가 모든 토픽이 구독 중임을 알게 됩니다.
- 이 작동은 존재하는 구독에 상관없이 계층의 모든 큐 관리자에게 항상 구독을 보내도록 변경될 수 있습니다. 이렇게 되면 계층에 걸쳐 구독 정보를 전파할 필요는 없어지지만 구독 트래픽이 늘어날 수 있습니다.

클러스터를 작성할 경우 메시지가 네트워크 내에서 영원히 순환하게 되는 루프를 작성하지 않도록 주의해야 합니다. 이러한 루프는 계층으로 작성할 수 없습니다.

모든 큐 관리자는 고유 큐 관리자 이름을 가져야 합니다.

발행/구독 메시지 플로우는 다중 발행/구독 클러스터로 확장될 수 있습니다. 이를 수행하려면 각 클러스터에서 하나의 큐 관리자를 발행/구독 계층에 추가하십시오.

보다 자세한 탐색을 위해 발행/구독 계층에서 라우팅을 참조하십시오.

발행/구독 네트워크에서의 프록시 구독

프록시 구독은 하나의 큐 관리자에 발행된 토픽에 대해 다른 큐 관리자가 작성하는 구독입니다. 프록시 구독은 구독에서 구독하는 각 개별 토픽 문자열에 대해 큐 관리자 사이에서 플로우됩니다. 사용자는 프록시 구독을 명확하게 작성하지 않지만 큐 관리자는 사용자를 위해 프록시 구독을 명확하게 작성합니다.

발행/구독 클러스터 또는 발행/구독 계층으로 함께 큐 관리자를 연결할 수 있습니다. 프록시 구독은 연결된 큐 관리자 사이에서 전달됩니다. 프록시 구독의 경우 한 큐 관리자에 연결된 발행자가 작성한 토픽에 대한 발행은 다른 큐 관리자에 연결된 해당 토픽에 대한 구독에서 수신됩니다. 78 페이지의 『분산 발행/구독 네트워크』의 내용을 참조하십시오.

개별 토픽 문자열에 대한 수천 개의 구독을 포함하는 발행/구독 토폴로지 또는 이러한 구독의 존재 여부가 빠르게 변화하는 환경에서 프록시 구독 전파의 오버헤드를 고려해야 합니다. 이 토픽의 나머지에서 설명된 자동 집계 외에도 연결된 큐 관리자 간에 프록시 구독 및 발행의 플로우를 추가로 제한하고 연결된 모든 큐 관리자에 프록시 구독을 전파하기 위해 대기하는 지연 시간을 줄이는 수동 구성 변경을 작성할 수 있습니다. 발행/구독 네트워크에서의 구독 성능을 참조하십시오.

프록시 구독에는 로컬 구독에 사용되는 선택자가 포함되지 않으며, 와일드카드가 포함된 구독 토픽 문자열이 간소화될 수 있습니다. 이는 큐 관리자 간의 추가 발행 플로우가 발생하여 실제 구독이 없는 프록시 구독과 일치하는 발행이 발생할 수 있습니다. 구독을 호스팅하는 큐 관리자는 추가 발행이 구독으로 리턴되지 않도록 불일치를 필터링합니다.

프록시 구독 집계

프록시 구독은 중복 제거 시스템을 사용하여 집계됩니다. 해석된 특정 토픽 문자열의 경우 프록시 구독은 수신된 프록시 구독 또는 첫 번째 로컬 구독에서 전송됩니다. 동일한 토픽 문자열에 대한 후속 구독은 이 기존 프록시 구독을 사용합니다.

마지막 로컬 구독 후에 프록시 구독이 취소되거나 수신된 프록시 구독이 취소됩니다.

발행 집계

큐 관리자에 동일한 토픽 문자열에 대한 구독이 둘 이상인 경우 해당 토픽 문자열과 일치하는 각 발행의 사본 하나만 발행/구독 토폴로지의 다른 큐 관리자에서 전송됩니다. 메시지가 도착하면 로컬 큐 관리자는 일치하는 각 구독에 메시지의 사본을 전달합니다.

프록시 구독이 와일드카드를 포함하는 경우 단일 발행의 토픽 문자열과 둘 이상의 구독이 일치할 수 있습니다. 메시지가 하나의 연결된 큐 관리자에서 작성한 둘 이상의 프록시 구독과 일치하는 큐 관리자에서 발행되는 경우 발행의 사본 하나만 리모트 큐 관리자로 전달되어 다중 프록시 구독을 만족시킵니다.

관련 개념

분산 발행/구독 네트워크의 루프 감지

프록시 구독의 와일드카드

구독은 발행에서 다중 토픽 문자열에 대해 일치하도록 토픽 문자열에서 와일드카드를 사용할 수 있습니다.

구독이 사용할 수 있는 와일드카드 스키마는 *topic-based*와 *character-based*의 두 가지입니다. 62 페이지의 『와일드카드 설계』의 내용을 참조하십시오.

IBM MQ 에서 와일드카드 구독에 대한 모든 프록시 구독은 토픽 기반 와일드카드를 사용하도록 변환됩니다. 문자 기반 와일드카드가 나오면, 가장 가까운 / 뒤가 # 문자로 바뀝니다. 예를 들어 `/aaa/bbb/c*d` 는 `/aaa/bbb/#`으로 변환됩니다. 이러한 변환으로 인해 리모트 큐 관리자는 명시적으로 구독하는 것보다 약간 많은 발행을 송신합니다. 추가 발행은 로컬 구독자로 발행을 전달할 때 로컬 큐 관리자로 필터링됩니다.

WILDCARD 특성으로 와일드카드 사용 제어

MQSC **Topic** WILDCARD 특성 또는 동등한 PCF 토픽 `WildcardOperation` 특성을 사용하여 와일드카드 토픽 문자열 이름을 사용하는 구독자 애플리케이션으로 발행물의 전달을 제어할 수 있습니다. WILDCARD 특성은 다음과 같은 두 가지 가능한 값을 보유할 수 있습니다.

WILDCARD
이 토픽에 관한 와일드카드 구독의 동작입니다.

PASSTHRU
이 토픽 오브젝트의 토픽 문자열보다 덜 특정한 와일드카드 토픽에 대한 구독이 이 토픽 및 이 토픽보다 더욱 특정한 토픽 문자열에 대한 발행물을 수신합니다.

BLOCK
이 토픽 오브젝트의 토픽 문자열보다 덜 특정한 와일드카드 토픽에 대한 구독이 이 토픽 또는 이 토픽보다 더욱 특정한 토픽 문자열에 대한 발행물을 수신하지 않습니다.

이 속성의 값은 구독이 정의될 때 사용됩니다. 이 속성을 대체할 경우 기존 구독에 포함된 토픽 세트는 수정의 영향을 받지 않습니다. 이 시나리오는 토픽 오브젝트를 작성 또는 삭제할 때 토폴로지가 변경된 경우에도 적용됩니다. WILDCARD 속성을 수정한 후에 작성된 구독과 일치하는 토픽 세트가 수정된 토폴로지를 사용하여 작성됩니다. 일치하는 토픽 세트를 강제로 기존 구독에 대해 재평가하려는 경우 큐 관리자를 재시작해야 합니다.

예제, 73 페이지의 『예제: Sport 발행/구독 클러스터 작성』 에서는 70 페이지의 그림 23에 표시된 토픽 트리 구조를 작성하는 단계를 수행할 수 있습니다.

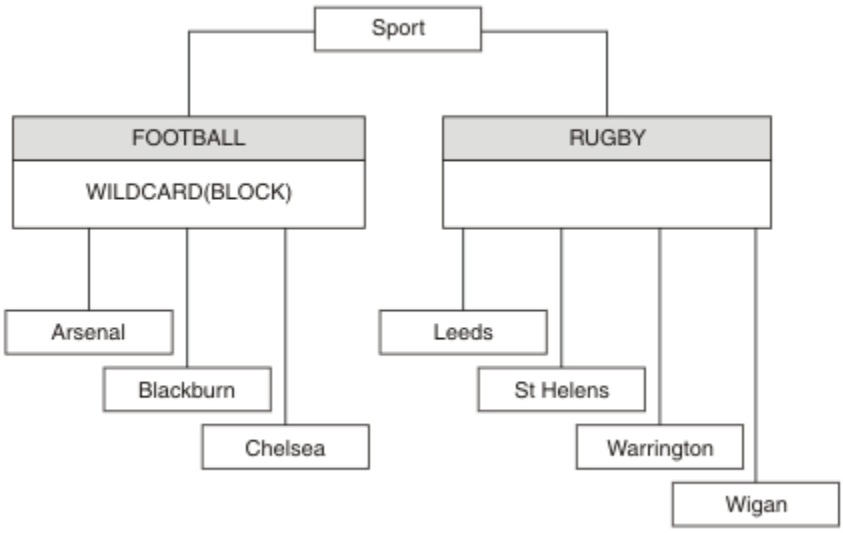


그림 *36. WILDCARD* 특성, *BLOCK*을 사용하는 토픽 트리

와일드카드 토픽 문자열 #을 사용하는 구독자는 Sport 토픽 및 Sport/Rugby 하위 트리에 대한 모든 발행을 수신합니다. 구독자는 Sport/Football 하위 트리에 대한 발행을 수신하지 않습니다. Sport/Football 토픽의 WILDCARD 특성 값은 BLOCK입니다.

PASSTHRU는 기본 설정입니다. Sport 트리의 노드에서 WILDCARD 특성 값 PASSTHRU를 설정할 수 있습니다. 노드에 WILDCARD 특성 값 BLOCK이 없으면 PASSTHRU를 설정해도 Sports 트리의 노드에서 구독자가 관찰하는 작동은 대체되지 않습니다.

예제에서 구독을 작성하여 와일드카드 설정이 전달되는 발행에 미치는 영향을 확인하십시오(75 페이지의 그림 27 참조). 일부 발행을 작성하려면 76 페이지의 그림 30에서 발행 명령을 실행하십시오.

```
pub QMA
```

그림 *37. QMA*에 발행

결과는 71 페이지의 표 3에 표시됩니다. WILDCARD 특성 값 BLOCK을 설정할 때 와일드카드를 포함하는 구독이 와일드카드 범위 내 토픽에 대한 발행을 수신하지 않도록 하는 방법에 주의하십시오.

표 6. QMA에서 수신된 발행			
구독	**토픽 문자열**	**수신된 발행**	**참고**
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Sports/Football에서 WILDCARD(BLOCK)으로 차단된 Football 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby의 기본 WILDCARD는 Leeds에서 와일드카드 구독을 금지하지 않습니다.

참고:

구독에 WILDCARD 특성 값 BLOCK이 있는 토픽 오브젝트와 일치하는 와일드카드가 있다고 가정합니다. 또한 구독이 일치하는 와일드카드 오른쪽에 토픽 문자열을 포함하면 구독은 발행을 수신하지 않습니다. 차단되지 않은 발행 세트는 차단된 와일드카드 상위에 해당하는 토픽에 대한 발행입니다. BLOCK 특성 값이 설정된 토픽의 하위에 해당하는 토픽에 대한 발행은 와일드카드로 차단됩니다. 따라서 와일드카드 오른쪽에 토픽을 포함하는 구독 토픽 문자열은 일치하는 발행을 수신하지 않습니다.

WILDCARD 특성 값을 BLOCK으로 설정해도 와일드카드를 포함하는 토픽 문자열을 사용하여 구독할 수 없음을 의미하지는 않습니다. 이러한 구독은 정상적입니다. 구독이 WILDCARD 특성 값 BLOCK이 설정된 토픽 오브젝트를 포함하는 토픽과 일치하는 명시적 토픽을 보유합니다. 이는 WILDCARD 특성 값 BLOCK이 설정된 토픽의 상위 또는 하위에 해당하는 토픽에 대한 와일드카드를 사용합니다. 70 페이지의 그림 23의 예제에서 Sports/Football/#과 같은 구독은 발행을 수신할 수 있습니다.

와일드카드 및 클러스터 토픽

클러스터 토픽 정의는 클러스터의 모든 큐 관리자로 전파됩니다. 클러스터의 한 큐 관리자에서 클러스터 토픽에 대한 구독으로 인해 큐 관리자가 프록시 구독을 작성할 수 있습니다. 프록시 구독은 클러스터의 다른 모든 큐 관리자에서 작성됩니다. 클러스터 토픽과 결합해 와일드카드를 포함하는 토픽 문자열을 사용하는 구독은 작동을 예상하기 어려울 수 있습니다. 작동은 다음 예제에서 설명됩니다.

예제, 73 페이지의 『예제: Sport 발행/구독 클러스터 작성』에 설정된 클러스터의 경우 QMB에는 QMA와 동일한 구독 세트가 있지만 QMB는 발행자가 QMA에 발행한 후 발행을 수신하지 않습니다(71 페이지의 그림 24 참조). Sports/Football 및 Sports/Rugby 토픽이 클러스터 토픽이어도 fullsubs.tst에 정의된 구독은 클러스터 토픽을 참조하지 않습니다. 프록시 구독은 QMB에서 QMA로 전파되지 않습니다. 프록시 구독이 없으면 QMA에 대한 발행은 QMB로 전달되지 않습니다.

Sports/#/Leeds와 같은 일부 구독은 이 경우 Sports/Rugby에 해당하는 클러스터 토픽을 참조할 수도 있습니다. Sports/#/Leeds 구독은 실제로 토픽 오브젝트 SYSTEM.BASE.TOPIC으로 해석됩니다.

Sports/#/Leeds와 같이 구독에서 참조하는 토픽 오브젝트를 해석하는 규칙은 다음과 같습니다. 토픽 문자열을 첫 번째 와일드카드까지 자릅니다. 토픽 문자열을 왼쪽부터 스캔하여 연관된 관리 토픽 오브젝트가 있는 첫 번째 토픽을 찾습니다. 이 토픽 오브젝트는 클러스터 이름을 지정하거나 로컬 토픽 오브젝트를 정의할 수 있습니다. 예 Sports/#/Leeds에서 잘림 이후의 토픽 문자열은 토픽 오브젝트가 없는 Sports이므로 Sports/#/Leeds 는 로컬 토픽 오브젝트인 SYSTEM.BASE.TOPIC에서 상속합니다.

클러스터 토픽을 구독하여 와일드카드 전파 방식을 변경하는 방법을 확인하려면 배치 스크립트, upsubs.bat를 실행하십시오. 스크립트는 구독 큐를 지우고 fullsubs.tst에서 클러스터 토픽 구독을 추가합니다. puba.bat를 다시 실행하여 발행 배치를 작성하십시오(71 페이지의 그림 24 참조).

72 페이지의 표 4에서는 발행이 발행된, 동일한 큐 관리자에 2개의 새 구독을 추가하는 결과를 보여줍니다. 결과는 예상한 대로입니다. 새 구독은 각각 하나의 발행을 수신하고 다른 구독에서 수신하는 발행 수는 변경되지 않습니다. 다른 클러스터 큐 관리자에서 예상치 못한 결과가 발생합니다(72 페이지의 표 5 참조).

표 *7. QMA*에서 수신된 발행

구독	토픽 문자열	수신된 발행	참고
SPORTS	Sports/#	Sports Sports/Rugby Sports/Rugby/Leeds	Sports/Football에서 WILDCARD(BLOCK)으로 차단된 Football 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함
SLEEDS	Sports/#/Leeds	Sports/Rugby/Leeds	Sports/Rugby의 기본 WILDCARD는 Leeds에서 와일드카드 구독을 금지하지 않습니다.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal은 구독이 와일드카드를 포함하지 않으므로 발행을 수신합니다.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds는 모든 이벤트에서 발행을 수신합니다.

72 페이지의 표 5에서는 QMB에서 2개의 새 구독을 추가하고 QMA에서 발행하는 결과를 보여줍니다. QMB가 이러한 2개의 새 구독 없이 발행을 수신하지 않는다는 점을 상기하십시오. 예상대로 Sports/FootBall 및 Sports/Rugby 가 둘 다 클러스터 토픽이므로 두 개의 새 구독이 발행물을 수신합니다. QMB 는 Sports/Football/Arsenal 및 Sports/Rugby/Leeds 에 대한 프록시 구독을 QMA에 전달하며, 이는 QMB에 발행물을 송신합니다.

예상치 못한 결과는, 이전에 발행을 수신하지 않던 2개의 구독 Sports/# 및 Sports/#/Leeds가 이제 발행을 수신한다는 점입니다. 이유는 바로, 다른 구독에 대해 QMB로 전달되는 Sports/Football/Arsenal 및 Sports/Rugby/Leeds 발행이 이제 QMB에 연결된 모든 구독자에게 사용 가능합니다. 결과적으로 로컬 토픽 Sports/# 및 Sports/#/Leeds에 대한 구독은 Sports/Rugby/Leeds 발행을 수신합니다. Sports/#/Arsenal은 계속해서 발행을 수신하지 않습니다. Sports/Football에서 해당 WILDCARD 특성 값이 BLOCK으로 설정되었기 때문입니다.

표 *8. QMB*에서 수신된 발행

구독	토픽 문자열	수신된 발행	참고
SPORTS	*Sports/#*	*Sports/Rugby/Leeds*	*Sports/Football*에서 *WILDCARD(BLOCK)*에 의해 차단된 *Football* 하위 트리에 대한 모든 발행
SARSENAL	Sports/#/Arsenal	-	Sports/Football에서 WILDCARD(BLOCK)은 Arsenal에서의 와일드카드 구독을 금지함
SLEEDS	*Sports/#/Leeds*	*Sports/Rugby/Leeds*	*Sports/Rugby*의 기본 *WILDCARD*는 *Leeds*에서 와일드카드 구독을 금지하지 않습니다.
FARSENAL	Sports/Football/Arsenal	Sports/Football/Arsenal	Arsenal은 구독이 와일드카드를 포함하지 않으므로 발행을 수신합니다.
FLEEDS	Sports/Rugby/Leeds	Sports/Rugby/Leeds	Leeds는 모든 이벤트에서 발행을 수신합니다.

대부분의 애플리케이션에서는 한 구독이 다른 구독의 작동에 영향을 주는 것은 바람직하지 않습니다. 값이 BLOCK인 WILDCARD 특성의 한 가지 중요한 사용법은 균일하게 작동하는 와일드카드를 포함하는 동일한 토픽 문자열에 대한 구독을 작성하는 것입니다. 발행자와 동일한 큐 관리자 또는 다른 큐 관리자의 구독인지에 상관없이 구독 결과는 동일합니다.

와일드카드 및 스트림

발행/구독 API에 기록된 새 애플리케이션의 경우 * 에 대한 구독이 발행물을 수신하지 않습니다. 모든 Sport 발행을 수신하려면 Sports/* 또는 Sports/#을 구독해야 하며,하고 Business 발행도 마찬가지입니다.

큐된 기존 발행/구독 애플리케이션의 동작은 발행/구독 브로커가 IBM MQ의 이후 버전으로 마이그레이션될 때 변경되지 않습니다. **Publish**, **Register Publisher**또는 **Subscriber** 명령의 **StreamName** 특성은 스트림이 마이그레이션된 토픽의 이름에 맵핑됩니다.

와일드카드 및 구독 지점

발행/구독 API에 기록된 새 애플리케이션의 경우 마이그레이션의 효과는 * 에 대한 구독이 발행물을 수신하지 않는다는 점입니다. 모든 Sport 발행을 수신하려면 Sports/* 또는 Sports/#을 구독해야 하며,하고 Business 발행도 마찬가지입니다.

큐된 기존 발행/구독 애플리케이션의 동작은 발행/구독 브로커가 IBM MQ의 이후 버전으로 마이그레이션될 때 변경되지 않습니다. **Publish**, **Register Publisher** 또는 **Subscriber** 명령에서 **SubPoint** 특성은 구독이 마이그레이션되는 토픽 이름에 맵핑됩니다.

예제: Sport 발행/구독 클러스터 작성

다음 단계는 네 개의 큐 관리자 (두 개의 전체 저장소, CL1A 및 CL1B, 두 개의 부분 저장소, QMA 및 QMB) 가 있는 CL1클러스터를 작성합니다. 전체 저장소는 클러스터 정의만 보유하는 데 사용됩니다. QMA는 클러스터 토픽 호스트를 지정합니다. 지속 가능 구독은 QMA 및 QMB 모두에서 정의됩니다.

참고: 예제는 Windows용으로 코딩되었습니다. 다른 플랫폼에서 예제를 구성하고 테스트하도록 qmgrs.bat 작성 및 pub.bat 작성을 다시 코딩해야 합니다.

1. 스크립트 파일을 작성하십시오.
   a. topics.tst 작성
   b. wildsubs.tst 작성
   c. fullsubs.tst 작성
   d. qmgrs.bat 작성
   e. pub.bat 작성
2. qmgrs.bat 작성을 실행하여 구성을 작성하십시오.

```
qmgrs
```

70 페이지의 그림 23에서 토픽을 작성하십시오. 그림 5에서 스크립트는 클러스터 토픽 Sports/Football 및 Sports/Rugby를 작성합니다.

참고: REPLACE 옵션은 주제의 TOPICSTR 특성을 바꾸지 않습니다. TOPICSTR은(는) 다른 토픽 트리를 테스트하는 예제에서 사용되는 특성입니다. 토픽을 변경하려면 먼저 토픽을 삭제하십시오.

```
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')
```

그림 *38.* 토픽 삭제 및 작성*: topics.tst*

참고: REPLACE가 토픽 문자열을 바꾸지 않으므로 토픽을 삭제합니다.

와일드카드를 사용하여 구독을 작성하십시오. 70 페이지의 그림 23에서 토픽 오브젝트를 포함하는 토픽에 대응하는 와일드카드. 각 구독에 대한 큐를 작성하십시오. 스크립트를 실행하거나 다시 실행하면 큐가 지워지고 구독이 삭제됩니다.

참고: REPLACE 옵션은 구독의 TOPICOBJ 또는 TOPICSTR 특성을 바꾸지 않습니다. TOPICOBJ 또는 TOPICSTR은 다른 구독을 테스트하기 위해 예제에서 다양하게 사용할 수 있는 특성입니다. 변경하려면 먼저 구독을 삭제하십시오.

```
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)
```

그림 *39.* 와일드카드 구독 작성*: wildsubs.tst*

클러스터 토픽 오브젝트를 참조하는 구독을 작성하십시오.

참고:

구분 기호, /는 TOPICOBJ에서 참조하는 토픽 문자열과 TOPICSTR에서 정의하는 토픽 문자열 사이에 자동으로 삽입됩니다.

DEFINE SUB(FARSENAL) TOPICSTR('Sports/Football/Arsenal') DEST(QFARSENAL) 정의는 동일한 구독을 작성합니다. TOPICOBJ는 이미 정의한 토픽 문자열을 참조하는 빠른 방법으로 사용됩니다. 작성되면 구독은 더 이상 토픽 오브젝트를 참조하지 않습니다.

```
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)
```

그림 *40.* 구독 삭제 및 삭제*: fullsubs.tst*

2개의 저장소를 포함하는 클러스터를 작성하십시오. 발행 및 구독을 위해 2개의 부분 저장소를 작성하십시오. 모두 삭제하고 다시 시작하도록 스크립트를 다시 실행하십시오. 또한 스크립트는 토픽 계층 및 초기 와일드카드 구독을 작성합니다.

참고:

다른 플랫폼에 비슷한 스크립트를 작성하거나 모두 명령을 입력하십시오. 스크립트를 사용하면 모두 삭제하고 동일한 구성으로 다시 빠르게 시작할 수 있습니다.

```
@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
```

그림 *41.* 큐 관리자 작성*: qmgrs.bat*

클러스터 토픽에 구독을 추가하여 구성을 업데이트하십시오.

```
@echo off
for %%A in (QMA QMB) do runmqsc %%A < wildsubs.tst
for %%A in (QMA QMB) do runmqsc %%A < upsubs.tst
```

그림 *42.* 구독 업데이트*: upsubs.bat*

큐 관리자에서 매개변수로 pub.bat를 실행하여 발행 토픽 문자열을 포함하는 메시지를 발행하십시오. Pub.bat 는 샘플 프로그램 **amqspub**를 사용합니다.

```
@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
```

그림 *43.* 발행*: pub.bat*

관련 개념
와일드카드 구독 및 보유된 발행

발행 범위

발행/구독 클러스터 또는 계층을 구성하는 경우, 발행 범위는 큐 관리자가 리모트 큐 관리자로 발행을 전달할지 여부를 추가로 제어합니다. **PUBSCOPE** 토픽 속성을 사용하여 발행 범위를 관리합니다.

발행이 리모트 큐 관리자에 전달되지 않으면 로컬 구독자만 발행을 수신합니다.

발행/구독 클러스터를 사용할 경우, 발행의 범위는 주로 토픽 트리의 특정 지점에 있는 클러스터 토픽 오브젝트의 정의에 의해 제어됩니다. 발행 범위는 클러스터의 다른 큐 관리자로 발행의 플로우를 허용하도록 설정되어야 합니다. 특정 큐 관리자에서 특정 토픽의 상세한 제어를 필요로 하는 경우 클러스터 토픽에 대한 발행 범위만 제한해야 합니다.

발행/구독 계층을 사용할 경우, 발행의 범위는 주로 구독 범위 속성과 결합된 이 속성에 의해 제어됩니다.

PUBSCOPE 속성은 특정 토픽에서 만들어진 발행 범위를 판별하는 데 사용됩니다. 속성을 다음 값 중 하나로 설정할 수 있습니다.

QMGR
: 발행은 로컬 구독자에게만 전달됩니다. 이러한 발행을 로컬 발행이라고 합니다. 로컬 발행은 리모트 큐 관리자로 전달되지 않으므로 리모트 큐 관리자에 연결된 구독자는 수신하지 않습니다.

모두
: 발행은 발행/구독 클러스터 또는 계층의 리모트 큐 관리자에 연결된 구독자 및 로컬 구독자에게 전달됩니다. 이러한 발행을 글로벌 발행이라고 합니다.

ASPARENT
: 토픽 트리에 있는 상위 토픽의 **PUBSCOPE** 설정을 사용합니다.

또한 발행자는 MQPMO_SCOPE_QMGR 입력 메시지 옵션을 사용하여 발행 종류(로컬 또는 글로벌)를 지정할 수 있습니다. 이 옵션을 사용하면 **PUBSCOPE** 토픽 속성을 사용하여 설정된 동작을 대체합니다.

관련 개념
68 페이지의 『관리 토픽 오브젝트』
관리 토픽 오브젝트를 사용하여 기본이 아닌 특정 속성을 토픽에 지정할 수 있습니다.

관련 태스크
분산 발행/구독 네트워크 구성

구독 범위

구독 범위는 한 큐 관리자의 구독이 발행/구독 클러스터 또는 계층에 있는 다른 큐 관리자에서 발행된 발행을 수신하는지, 아니면 로컬 발행자의 발행을 수신하는지 여부를 제어합니다.

큐 관리자로 구독 범위를 제한하면 프록시 구독이 발행/구독 토폴로지에 있는 다른 큐 관리자로 전달되는 작업이 중지됩니다. 이로 인해 큐 관리자 내 발행/구독 메시징 트래픽을 줄입니다.

발행/구독 클러스터를 사용할 경우, 구독의 범위는 주로 토픽 트리의 특정 지점에 있는 클러스터 토픽 오브젝트의 정의에 의해 제어됩니다. 구독 범위는 클러스터의 다른 큐 관리자로 구독의 플로우를 허용하도록 설정되어야 합니다. 특정 큐 관리자에서 특정 토픽의 상세한 제어를 필요로 하는 경우 클러스터 토픽에 대한 구독 범위만 제한해야 합니다.

발행/구독 계층을 사용할 경우, 구독의 범위는 주로 발행 범위 속성과 결합된 이 속성에 의해 제어됩니다.

SUBSCOPE 토픽 속성은 특정 토픽에 대한 구독의 범위를 판별하는 데 사용됩니다. 속성을 다음 값 중 하나로 설정할 수 있습니다.

QMGR
구독은 로컬 발행만 수신하며 프록시 구독은 리모트 큐 관리자로 전파되지 않습니다.

모두
프록시 구독은 발행/구독 클러스터 또는 계층의 리모트 큐 관리자로 전파되며 구독자는 로컬 및 원격 발행을 수신합니다.

ASPARENT
토픽 트리에 있는 상위 토픽의 **SUBSCOPE** 설정을 사용합니다.

토픽에 대한 구독 범위가 직접 또는 ASPARENT를 통해 분석된 ALL로 설정된 경우, 구독을 작성할 때 MQSO_SCOPE_QMGR을 지정하여 해당 토픽에 대한 개별 구독이 해당 범위를 QMGR로 제한할 수 있습니다. QMGR의 범위가 있는 토픽에 대한 구독은 ALL로 범위를 확장할 수 없습니다.

관련 개념
68 페이지의 『관리 토픽 오브젝트』
관리 토픽 오브젝트를 사용하여 기본이 아닌 특정 속성을 토픽에 지정할 수 있습니다.

관련 태스크
분산 발행/구독 네트워크 구성

토픽 공간

토픽 공간은 사용자가 구독 및 발행할 수 있는 토픽 세트입니다. 분산 발행/구독 토폴로지의 큐 관리자에는 해당 토폴로지의 연결된 큐 관리자에서 구독 및 발행된 토픽을 잠재적으로 포함하는 토픽 공간이 있습니다.

참고: 관리 토픽 오브젝트, 토픽 문자열 및 토픽 트리와 같은 큐 관리자 내의 토픽에 대한 개요는 60 페이지의 『토픽』의 내용을 참조하십시오. 현재 문서에서 토픽에 대한 추가 참조는 달리 지정하지 않는 한 토픽 문자열을 의미합니다.

토픽은 다음 방법 중 하나로 초기에 작성됩니다.

- 토픽 오브젝트 또는 지속 가능 구독을 정의하는 경우 관리 하에
- 애플리케이션이 새 토픽에 대해 동적으로 발행 또는 구독을 작성할 경우 동적으로

토픽은 프록시 구독 및 관리 클러스터 토픽 오브젝트를 작성하는 방식 모두를 통해 다른 큐 관리자로 전파됩니다. 프록시 구독의 경우 발행자가 연결된 큐 관리자에서 구독의 큐 관리자로 발행이 전달됩니다.

프록시 구독은 큐 관리자 계층에서 상위-하위 관계로 함께 연결되는 모든 큐 관리자 사이에 전파됩니다. 결과적으로 한 큐 관리자에서 계층의 다른 큐 관리자에 정의된 토픽을 구독할 수 있습니다. 큐 관리자 사이에 연결된 경로가 있는 한, 큐 관리자의 연결 방식은 중요하지 않습니다.

프록시 구독은 또한 구독을 위해 발행/구독 클러스터의 클러스터 토픽으로도 전파됩니다. 클러스터 토픽은 **CLUSTER** 속성이 있거나 상위로부터 속성을 상속하는 토픽 오브젝트에 연결된 토픽입니다. 클러스터 토픽이 아닌 토픽은 로컬 토픽이라고 하며 클러스터에 복제되지 않습니다. 클러스터에 대해서는 구독에서 로컬 토픽으로 프록시 구독이 전파되지 않습니다.

즉, 프록시 구독은 두 가지 상황에서 구독자에 대해 작성됩니다.

1. 큐 관리자가 계층의 멤버이고 프록시 구독이 큐 관리자의 상위와 하위로 전달됩니다.
2. 큐 관리자가 클러스터의 멤버이고 구독 토픽 문자열이 클러스터 토픽 오브젝트와 연관된 토픽으로 분석됩니다. 토픽이 직접 라우팅된 클러스터 토픽인 경우, 프록시 구독이 클러스터의 모든 멤버에게 전달됩니다. 토픽이 토픽 호스트 라우트 클러스터 토픽인 경우, 프록시 구독은 클러스터 토픽 오브젝트를 정의한 클러스터의 큐 관리자로만 전달됩니다. 자세한 정보는 81 페이지의 『발행/구독 클러스터』의 내용을 참조하십시오.

큐 관리자가 클러스터 및 계층의 멤버이면 구독자에게 중복 발행을 전달하지 않고도 두 메커니즘에서 프록시 구독이 전파됩니다.

3개 발행/구독 토폴로지의 토픽 공간은 다음 목록에서 설명됩니다.

- 93 페이지의 『케이스 1. 발행/구독 클러스터』.
- 93 페이지의 『케이스 2. 발행/구독 계층』.

별도의 토픽에서, 다음 구성 태스크는 토픽 공간을 결합하는 방법에 대해 설명합니다.

- 발행/구독 클러스터에서 단일 토픽 공간 작성
- 다중 클러스터의 토픽 공간 결합
- 다중 클러스터에서 토픽 공간 결합 및 격리
- 다중 클러스터에서 토픽 공간 발행 및 구독

케이스 1. 발행/구독 클러스터

예에서, 큐 관리자가 발행/구독 계층에 연결되지 않았다고 가정합니다.

큐 관리자가 발행/구독 클러스터의 멤버이면 해당 토픽 공간은 로컬 토픽 및 클러스터 토픽에서 구성됩니다. 로컬 토픽은 **CLUSTER** 속성 없이 토픽 오브젝트에 연관됩니다. 큐 관리자에 로컬 토픽 오브젝트 정의가 있으면 해당 토픽 공간은 로컬로 정의된 고유한 토픽 오브젝트도 포함하는 클러스터의 다른 큐 관리자와는 다릅니다.

구독한 토픽을 클러스터 토픽 오브젝트로 해석하지 않는 한, 발행/구독 클러스터에서 다른 큐 관리자에 정의된 토픽을 구독할 수 없습니다.

클러스터 토픽 오브젝트의 같은 이름을 가진 정의가 다중 큐 관리자에 필요한 경우, 예를 들어 토픽 호스트 라우팅을 사용 중인 경우, 필요할 때 모든 정의가 일치하는 것은 중요합니다. 자세한 정보는 발행/구독 클러스터에서 단일 토픽 공간 작성을 참조하십시오.

토픽 오브젝트의 로컬 정의는 클러스터 토픽에 대한 정의인지, 로컬 토픽에 대한 정의인지에 상관없이 클러스터의 다른 위치에서 정의된 동일한 토픽 오브젝트보다 우선합니다. 다른 곳에서 정의된 오브젝트가 보다 최근 항목이어도 로컬로 정의된 토픽이 사용됩니다.

클러스터 토픽 오브젝트는 클러스터의 모든 위치에서 동일한 토픽 문자열에 연관됩니다. 토픽 오브젝트가 연관된 토픽 문자열은 수정할 수 없습니다. 다른 토픽 문자열과 동일한 토픽 오브젝트를 연관하려면 토픽 오브젝트를 삭제하고 새 토픽 문자열로 다시 작성해야 합니다. 토픽이 클러스터된 경우 효과는 클러스터의 다른 멤버에 저장된 토픽 오브젝트의 사본을 삭제하고 클러스터의 모든 위치에서 새 토픽 오브젝트의 사본을 작성하는 것입니다. 토픽 오브젝트의 사본은 모두 동일한 토픽 문자열을 참조합니다.

클러스터의 서로 다른 큐 관리자에서 서로 다른 토픽 문자열을 사용하여 동일한 이름의 토픽 오브젝트 정의를 우발적으로 두 개 작성할 수 있습니다. 그러면 동작이 혼동될 수 있습니다. 토픽 문자열이 서로 다른, 동일한 토픽 오브젝트의 여러 정의로 인해 토픽을 참조하는 방법과 위치에 따라 다른 결과가 나올 수 있습니다. 이 중요한 시점에 대한 자세한 정보는 동일한 이름의 다중 클러스터 토픽 정의를 참조하십시오.

케이스 2. 발행/구독 계층

예에서 큐 관리자가 발행/구독 클러스터의 구성원이 아니라고 가정합니다.

IBM MQ에서 큐 관리자가 발행/구독 계층의 멤버인 경우 해당 토픽 공간은 로컬 및 연결된 큐 관리자에 정의된 모든 토픽으로 구성됩니다. 계층에서 모든 큐 관리자의 토픽 공간은 동일합니다. 로컬 토픽과 글로벌 토픽으로 토픽을 구분하지 않습니다.

PUBSCOPE 및 **SUBSCOPE** 옵션을 QMGR로 설정하여 토픽의 발행을 발행자로부터 계층의 다른 큐 관리자에 연결된 구독자로 전달되지 않도록 하십시오.

큐 관리자 QMA에서 토픽 문자열 USA/Alabama를 포함하는 토픽 오브젝트 Alabama를 정의한다고 가정하십시오. 결과는 다음과 같습니다.

1. 이제 QMA의 토픽 공간에 토픽 오브젝트 Alabama 및 토픽 문자열 USA/Alabama가 포함됩니다.
2. 애플리케이션 또는 관리자가 토픽 오브젝트 이름 Alabama를 사용하여 QMA에 구독을 작성할 수 있습니다.
3. 애플리케이션은 계층의 큐 관리자에서 USA/Alabama를 포함하여 토픽에 대한 구독을 작성할 수 있습니다. QMA가 로컬로 정의되지 않은 경우 토픽 USA/Alabama가 토픽 오브젝트 SYSTEM.BASE.TOPIC으로 해석됩니다.

관련 개념

91 페이지의 『발행 범위』
발행/구독 클러스터 또는 계층을 구성하는 경우, 발행 범위는 큐 관리자가 리모트 큐 관리자로 발행을 전달할지 여부를 추가로 제어합니다. **PUBSCOPE** 토픽 속성을 사용하여 발행 범위를 관리합니다.

91 페이지의 『구독 범위』
구독 범위는 한 큐 관리자의 구독이 발행/구독 클러스터 또는 계층에 있는 다른 큐 관리자에서 발행된 발행을 수신하는지, 아니면 로컬 발행자의 발행을 수신하는지 여부를 제어합니다.

관련 태스크

분산 발행/구독 네트워크 구성

IBM MQ 멀티캐스트

IBM MQ 멀티캐스트는 낮은 지연, 높은 팬아웃, 신뢰할 수 있는 멀티캐스트 메시징을 제공합니다.

멀티캐스트는 성능을 저하시키지 않고 다수의 구독자에 맞게 스케일링할 수 있기 때문에 발행/구독 메시징의 효율적인 형식입니다. IBM MQ는 수신확인, 부정적 수신확인 및 순서 번호를 사용하여 높은 팬아웃을 갖는 낮은 지연 메시징을 달성함으로써 신뢰할 수 있는 멀티캐스트 메시징을 가능하게 합니다.

IBM MQ 멀티캐스트의 공정한 전달은 거의 동시 전달을 가능하게 하므로, 어떤 수신자도 우위를 차지하지 않도록 보장합니다. IBM MQ 멀티캐스트가 네트워크를 사용하여 메시지를 전달하기 때문에 발행/구독 엔진은 데이터를 팬아웃하기 위해 필요하지 않습니다. 토픽이 그룹 주소에 맵핑된 후, 발행자와 구독자가 피어 투 피어 모드에서 동작할 수 있기 때문에 큐 관리자가 필요없습니다. 여기에서는 큐 관리자 서버에 대한 로드가 감소하므로, 큐 관리자 서버가 더 이상 잠재적인 실패 지점이 되지 않습니다.

초기 멀티캐스트 개념

IBM MQ 멀티캐스트는 통신 정보(COMMINFO) 오브젝트를 사용하여 기존 시스템과 애플리케이션에 쉽게 통합할 수 있습니다. 두 개의 TOPIC 오브젝트 필드를 사용하면 기존 TOPIC 오브젝트의 빠른 구성에서 멀티캐스트 트래픽을 지원하거나 무시할 수 있습니다.

멀티캐스트에 필요한 오브젝트

다음 정보는 IBM MQ 멀티캐스트에 필요한 두 개의 오브젝트에 대한 간단한 개요입니다.

COMMINFO 오브젝트

COMMINFO 오브젝트에는 멀티캐스트 전송과 연관된 속성이 포함됩니다. COMMINFO 오브젝트 매개변수에 대한 자세한 정보는 DEFINE COMMINFO를 참조하십시오.

설정해야 하는 COMMINFO 필드만 COMMINFO 오브젝트의 이름입니다. 이 이름은 토픽에 대해 COMMINFO 오브젝트를 식별하는 데 사용됩니다. 값이 올바른 멀티캐스트 그룹 주소인지 확인하려면 COMMINFO 오브젝트의 **GRPADDR** 필드를 선택해야 합니다.

토픽 오브젝트

토픽은 발행/구독 메시지로 발행된 정보의 주제이며, 이러한 토픽은 TOPIC 오브젝트를 작성하여 정의합니다. TOPIC 오브젝트 매개변수에 대한 자세한 정보는 DEFINE TOPIC을 참조하십시오.

TOPIC 오브젝트 매개변수의 값(**COMMINFO** 및 **MCAST**)을 변경하여 멀티캐스트에서 기존 토픽을 사용할 수 있습니다.

- **COMMINFO** 이 매개변수는 멀티캐스트 통신 정보 오브젝트의 이름을 지정합니다.
- **MCAST** 이 매개변수는 토픽 트리의 이 지점에서 멀티캐스트를 허용할 수 있는지의 여부를 지정합니다. 기본적으로 **MCAST**는 토픽의 멀티캐스트 속성이 상위에서 상속됨을 의미하는 ASPARENT로 설정됩니다. **MCAST**를 ENABLED로 설정하면 이 노드에서 멀티캐스트 트래픽이 가능합니다.

멀티캐스트 네트워크 및 토픽

다음 정보는 여러 유형의 구독 및 토픽 정의가 있는 구독에 발생하는 상황에 대한 개요입니다. 이러한 예에서는 모두 TOPIC 오브젝트 **COMMINFO** 매개변수가 올바른 COMMINFO 오브젝트의 이름으로 설정되어 있다고 가정합니다.

사용 가능한 멀티캐스트로 설정된 토픽

토픽 문자열 **MCAST** 매개변수를 ENABLED로 설정하면, 다음과 같은 경우를 제외하고는 멀티캐스트 가능 클라이언트의 구독이 허용되며 멀티캐스트 구독이 이루어집니다.

- 멀티캐스트 가능 클라이언트의 지속 가능 구독인 경우

- 멀티캐스트 가능 클라이언트의 비관리 구독인 경우
- 멀티캐스트 불가능 클라이언트의 구독인 경우

이러한 경우 멀티캐스트가 아닌 구독이 이루어지며 구독이 일반 발행/구독으로 다운그레이드됩니다.

사용 불가능한 멀티캐스트로 설정된 토픽
토픽 문자열 **MCAST** 매개변수가 DISABLED로 설정되면, 항상 멀티캐스트가 아닌 구독이 이루어지고 구독이 일반 발행/구독으로 다운그레이드됩니다.

멀티캐스트로만 설정된 토픽
토픽 문자열 **MCAST** 매개변수를 ONLY로 설정하면, 다음과 같은 경우를 제외하고는 멀티캐스트 가능 클라이언트의 구독이 허용되며 멀티캐스트 구독이 이루어집니다.

- 지속 가능 구독임: 지속 가능 구독은 이유 코드 2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED로 거부됨
- 비관리 구독임: 비관리 구독은 이유 코드 2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR로 거부됨
- 멀티캐스트가 불가능한 클라이언트의 구독임: 이러한 구독은 이유 코드 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY로 거부됨
- 로컬로 바인드된 애플리케이션의 구독임: 이러한 구독은 이유 코드 2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY로 거부됨

Windows Linux AIX

MQ Telemetry 개요

MQ Telemetry는 큐 관리자의 일부인 텔레메트리(MQXR) 서비스, 직접 작성하거나 무료로 다운로드할 수 있는 텔레메트리 클라이언트와 명령행 및 탐색기 관리 인터페이스로 구성됩니다. 텔레메트리는 다양한 원격 장치에서 데이터를 수집하고 이런 장치를 관리하는 것을 말합니다. MQ Telemetry를 사용하면 데이터 수집과 장치 제어를 웹 애플리케이션으로 통합할 수 있습니다.

MQ Telemetry은 IBM MQ의 구성요소입니다. 이러한 버전으로 업그레이드하려면 IBM MQ의 이후 버전을 설치해야 합니다.

샘플 애플리케이션은 Eclipse Paho및 MQTT.org에서 무료로 계속 사용할 수 있습니다. IBM MQ Telemetry Transport 샘플 프로그램을 참조하십시오.

MQ Telemetry 는 IBM MQ의 구성요소이므로 기본 제품과 함께 또는 기본 제품이 설치된 후에 MQ Telemetry 를 설치할 수 있습니다. 마이그레이션 정보는 Windows 에서 MQ Telemetry 마이그레이션 및 Linux 에서 MQ Telemetry 마이그레이션의 내용을 참조하십시오.

MQ Telemetry에는 다음 컴포넌트가 포함되어 있습니다.

텔레메트리 채널
MQTT 클라이언트와 IBM MQ 사이의 연결을 관리하려면 텔레메트리 채널을 사용하십시오. 텔레메트리 채널은 새 IBM MQ 오브젝트 (예: SYSTEM.MQTT.TRANSMIT.QUEUE) 를 사용하여 IBM MQ와 상호작용합니다.

텔레메트리(MQXR) 서비스
MQTT 클라이언트는 SYSTEM.MQXR.SERVICE 텔레메트리 서비스를 사용하여 텔레메트리 채널에 연결합니다.

IBM MQ Explorer 지원 MQ Telemetry
MQ Telemetry는 IBM MQ Explorer를 사용하여 관리할 수 있습니다.

문서
MQ Telemetry 문서는 표준 IBM MQ 제품 문서에 포함되어 있습니다. Java와 C 클라이언트를 위한 SDK 문서는 Javadoc 및 HTML 형식으로 제품 문서에서 제공됩니다.

Telemetry 개념

우리는 주변 환경에서 정보를 수집하여 무엇을 할 지 결정합니다. 소비자로서 음식을 구입하기 전에 가게에 어떤 물건이 있는지 확인합니다. 환승 예약을 하기 전에 지금 떠나면 얼마나 오래 걸릴 지 알고 싶어합니다. 의사를 방문할까 결정하기 전에 자신의 증상을 확인합니다. 좀 더 기다릴까 정하기 전에 버스가 언제 도착할지 확인합니다. 이러한 결정을 내리게 되는 근거가 되는 정보는 측정기와 디바이스, 문서와 화면, 그리고 우리에게서 나옵니다. 어디에 있든, 필요할 때면 언제나 우리는 정보를 수집하고 종합하여 이를 토대로 행동합니다.

정보의 소스가 넓게 분산되어 있거나 액세스할 수 없으면 가장 정확한 정보를 모으기 어려우며 비용이 많이 들게 됩니다. 큰 변화를 가져오려 하거나 변화를 만들기 어려울 경우 변화시키지 못한 것이 생기거나 덜 효과적일 때 변화가 일어나게 됩니다.

넓게 분산되어 있는 디바이스를 디지털 기술로 인터넷에 연결시켜 이를 제어하고 여기서 정보를 수집하는 비용이 획기적으로 줄어들 수 있다면 어떠십니까? 이 정보는 인터넷과 기업의 자원을 사용하여 분석할 수 있습니다. 정보에 기반하여 결정을 내리고 행동을 취할 수 있는 기회가 더 많아집니다.

기술 트렌드, 환경 및 경제적 요구로 인해 다음과 같은 변화가 일어나고 있습니다.

1. 표준화와 저비용 디지털 프로세서와의 연결을 통해 센서와 작동기를 연결하고 제어하는 비용이 줄어들고 있습니다.
2. 디바이스를 연결하는 데 인터넷과 인터넷 기술이 점점 더 많이 사용되고 있습니다. 몇몇 나라에서는 인터넷 애플리케이션에 대한 연결 수에서 휴대전화가 개인용 컴퓨터를 앞서고 있습니다. 다른 디바이스들도 곧 이 뒤를 따르게 될 것입니다.
3. 인터넷과 인터넷 기술이 애플리케이션이 훨씬 쉽게 데이터를 얻게 해 줍니다. 쉽게 데이터에 액세스할 수 있게 되면서 센서에서 얻을 데이터를 많은 솔루션에 유용한 정보로 바꿔 주는 데이터 분석의 사용이 크게 늘고 있습니다.
4. 자원을 지능적으로 사용하는 것이 탄소 배출량과 비용을 절감하는 더 빠르고 저렴한 방법입니다. 새 자원을 찾거나 기존 자원을 사용하는 새 기술을 개발하는 것과 같은 대안 솔루션은 오랜 기간이 걸릴 수 있습니다. 단기간에 새 기술을 개발하거나 새 자원을 찾는 것은 기존 솔루션을 개선하는 것보다 종종 더 위험하고, 느리며 비용이 많이 듭니다.

예

다음 예는 어떻게 이러한 트렌드가 환경과 지능적으로 상호작용할 수 있는 새 기회를 제공하는지 보여줍니다.

국제해상안전협약(International Convention for the Safety of Life at Sea)은 많은 배에 자동 식별 시스템(AIS, automatic identification system)을 배치하도록 요구합니다. 300톤이 넘는 상선과 여객선에는 필수 사항입니다. AIS는 기본적으로 연안 해운의 충돌 회피 시스템입니다. 해군 당국에서 연안을 모니터하고 제어하는 데에도 사용됩니다.

전 세계의 열정적인 사람들이 저비용 AIS 추적 기지를 배치하고 연안 해운 정보를 인터넷에 올립니다. AIS의 정보를 인터넷의 다른 정보와 결합시키는 애플리케이션을 작성하는 사람도 있습니다. 결과는 웹 사이트에 게시되며 Twitter및 SMS를 사용하여 게시됩니다.

한 애플리케이션에서 사우스햄턴 가까이 있는 AIS 기지의 정보가 배 소유주 및 지리적 정보와 결합됩니다. 이 애플리케이션은 선박 도착 및 출발에 대한 실시간 정보를 Twitter에 제공합니다. 사우스햄턴과 아일오브와이트 사이에서 연락선을 사용하는 정기 통근자들은 Twitter 또는 SMS를 사용하여 새 피드를 구독합니다. 피드에 연락선이 느리게 운행 중이라고 표시되면 통근자들은 출발을 늦추고 연락선이 예정된 도착 시간보다 늦게 정박하면 연락선을 탈 수 있습니다.

다른 예는 98 페이지의 『텔레메트리 유스 케이스』의 내용을 참조하십시오.

관련 태스크

설치 MQ Telemetry
MQ Telemetry 관리
Windows 에서 MQ Telemetry 마이그레이션
Linux 에서 MQ Telemetry 마이그레이션
MQ Telemetry용 애플리케이션 개발
MQ Telemetry 문제점 해결

관련 참조

MQ Telemetry 참조

Windows Linux AIX MQ Telemetry 소개

사람과 비즈니스, 정부는 우리가 살며 일하는 환경과 좀 더 스마트한 상호작용을 하기 위해 MQ Telemetry를 점점 더 많이 활용하고자 합니다. MQ Telemetry는 모든 종류의 디바이스를 인터넷 및 기업과 연결하며 스마트 디바이스를 위한 애플리케이션을 빌드하는 비용을 감소시킵니다.

MQ Telemetry의 개념

- IBM MQ에 제공된 범용 메시징 백본을 광범위한 원격 센서, 작동기 및 텔레메트리 디바이스로 확장하는 IBM MQ 기능입니다. MQ Telemetry는 지능형 엔터프라이즈 애플리케이션, 서비스 및 의사결정자를 도구화된 디바이스 네트워크와 상호 연결할 수 있도록 IBM MQ를 확장합니다.
- MQ Telemetry의 핵심 파트는 다음과 같습니다.

  MQ Telemetry(MQXR) 서비스
  이 서비스는 IBM MQ 서버 내에서 실행되며 IBM MQ Telemetry Transport (MQTT) 프로토콜을 사용하여 텔레메트리 디바이스와 통신합니다.

  사용자가 기록하는 MQTT 애플리케이션
  이러한 애플리케이션은 텔레메트리 디바이스와 IBM MQ 큐 관리자 간에 다뤄지는 정보와 해당 정보에 대한 응답으로 수행되는 조치를 제어합니다. 이러한 애플리케이션을 작성하는 데 도움이 되도록 MQTT 클라이언트 라이브러리를 사용합니다.

사용 목적

- MQTT는 다양한 디바이스에 사용할 MQTT 구현을 작성할 수 있게 해주는 개방형 메시징 전송입니다.
- MQTT 클라이언트는 자원이 제한된 작은 풋프린트 디바이스에서 실행할 수 있습니다.
- MQTT는 대역폭이 낮거나, 송신 데이터 비용이 많이 소비되거나, 불안정한 네트워크에서 효과적으로 작동합니다.
- 메시지 전달은 보장되고, 애플리케이션과 구분됩니다.
- 애플리케이션 프로그래머에게 통신 프로그래밍 지식이 요구될 필요가 없습니다.
- 메시지는 다른 메시징 애플리케이션과 교환될 수 있습니다. 이는 다른 텔레메트리 애플리케이션, MQI, JMS 또는 엔터프라이즈 메시징 애플리케이션일 수 있습니다.

사용 방법

- 샘플 IBM MQ Telemetry Transport v3 클라이언트 애플리케이션(`mqttv3app.jar`)에서 작동하는 샘플 스크립트가 제공됩니다. IBM MQ Telemetry Transport 샘플 프로그램을 참조하십시오.
- IBM MQ Explorer 및 연관된 도구를 사용하여 IBM MQ의 텔레메트리 기능을 관리하십시오.
- 큐 관리자에 연결하고 발행/구독 메시징을 사용하는 MQTT 애플리케이션을 작성하는 데 도움이 되도록 클라이언트 라이브러리를 사용하십시오.
- 애플리케이션 및 클라이언트 라이브러리를 애플리케이션을 실행할 디바이스에 분배하십시오.

동작 방법

- MQTT는 발행/구독 프로토콜입니다. MQTT 클라이언트 애플리케이션은 MQTT 서버에 메시지를 발행하거나 MQTT 서버에 연결하는 애플리케이션이 송신하는 메시지를 구독할 수 있습니다.
- MQTT 클라이언트 애플리케이션은 MQTT 메시지 전송을 구현하는 클라이언트 라이브러리를 사용합니다.
- 기본 MQTT 클라이언트 애플리케이션은 표준 MQ 클라이언트처럼 작동하지만 더 광범위한 플랫폼 및 네트워크에서 실행할 수 있습니다.
- MQ Telemetry(MQXR) 서비스는 IBM MQ 큐 관리자를 MQTT 서버로 전환합니다.
- IBM MQ 큐 관리자가 MQTT 서버 역할을 수행하는 경우, 큐 관리자에 연결하는 다른 애플리케이션은 MQTT 클라이언트를 대상으로 메시지를 구독하고 수신할 수 있습니다.
- 큐 관리자는 발행 애플리케이션에서 구독 애플리케이션으로 메시지를 분배하는 라우터 역할을 합니다.
- 메시지 서로 다른 유형의 클라이언트 애플리케이션 간에 분배될 수 있습니다. 예를 들어 텔레메트리 클라이언트와 JMS 클라이언트 간에 분배될 수 있습니다.

참고: MQ Telemetry는 WebSphere Message Broker(지금은 IBM Integration Bus라고 함)의 버전 7에서 제거된 SCADA 노드를 대체하며 Windows, Linux 및 AIX에서 실행됩니다.

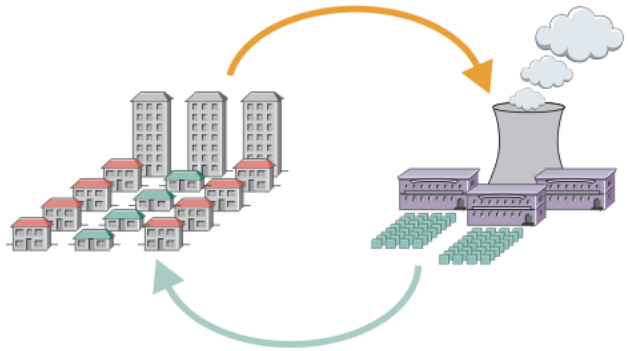
Windows Linux AIX 텔레메트리 유스 케이스

텔레메트리는 자동화된 감지, 데이터 측정 및 원격 디바이스의 제어를 가리킵니다. 주안점은 디바이스에서 중앙 제어 지점으로의 데이터 전송입니다. 또한 텔레메트리는 디바이스로 구성 및 제어 정보를 송신하는 것을 포함합니다.

MQ Telemetry는 MQTT protocol을 사용하는 소형 디바이스를 연결하고 해당 디바이스를 IBM MQ를 사용하는 다른 애플리케이션으로 연결합니다. MQ Telemetry는 디바이스와 인터넷 사이의 틈새를 중계하여 "스마트 솔루션"을 더 쉽게 구축할 수 있게 해 줍니다. 스마트 솔루션은 디바이스를 모니터하고 제어하는 애플리케이션을 위해 인터넷과 엔터프라이즈 애플리케이션으로부터 사용 가능한 풍부한 정보를 제공합니다.

다음 다이어그램은 MQ Telemetry의 일반적인 사용을 보여줍니다.

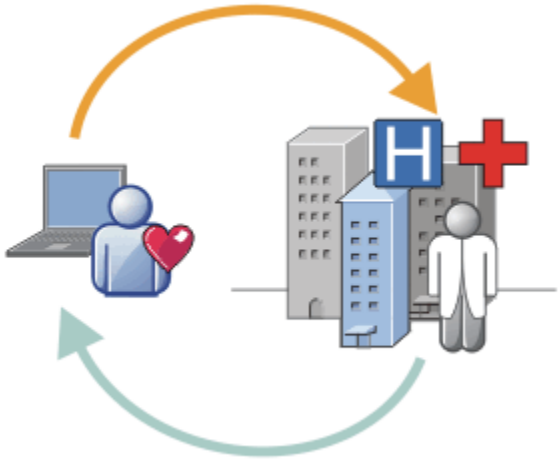
Telemetry: Smart Electricity



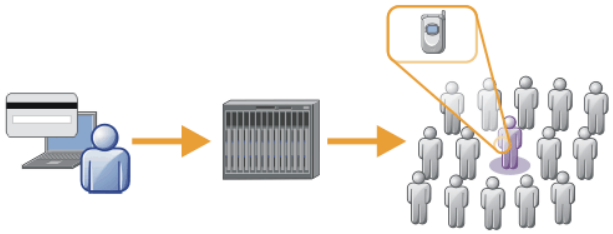
- 서비스 제공자에 전송된 에너지 사용 데이터를 포함하는 MQTT 메시지입니다.
- MQ Telemetry는 에너지 사용 데이터의 분석에 기반하여 CONTROL COMMANDS를 보냅니다.
- 자세한 정보는 다음 유스 케이스를 참조하십시오. 100 페이지의 『텔레메트리 유스 케이스: 가정 에너지 모니터링 및 제어』

Telemetry: Smart Health Services

- MQ Telemetry는 의료 데이터를 병원과 의사에게 송신합니다.
- MQTT 메시지 경보 또는 피드백은 의료 데이터의 분석을 기초로 송신될 수 있습니다.
- 자세한 정보는 다음 유스 케이스를 참조하십시오. 99 페이지의 『텔레메트리 유스 케이스: 가정 환자 모니터링』



Telemetry: 수천 명의 사람들 중 한 사람 식별



- 단순 카드 트랜잭션이 은행의 서버에 전송됩니다.
- MQ Telemetry는 수천 명의 사람들 중에서 한 사람을 식별하여 카드를 사용한 고객을 알려줍니다.
- MQ Telemetry는 가장 간단한 정보 입력을 사용하여 해당 개인을 찾습니다.

하위 주제에서 설명하는 유스 케이스는 실제 예제에서 유래합니다. 여기에서는 Telemetry를 사용하는 몇 가지 방법과 Telemetry 기술에서 해결해야 하는 몇 가지 공용된 문제점을 보여줍니다.

Windows Linux AIX 텔레메트리 유스 케이스: 가정 환자 모니터링

심장병 환자 간호 시스템에서의 IBM과 보건 의료 서비스 제공자와의 협업에서는 이식된 제세동기가 병원과 통신합니다. 환자와 이식된 디바이스에 대한 데이터는 RF 텔레메트리를 이용해 환자의 가정에 있는 MQTT 디바이스로 전송됩니다.

일반적으로 전송은 병상에 설치된 송신기에서 밤에 진행됩니다. 송신기는 전화 시스템을 이용해 데이터를 안전하게 병원으로 전송하며, 병원에서 이 데이터를 분석합니다.

이 시스템은 환자가 의사를 만나야 하는 횟수를 줄여 줍니다. 환자나 디바이스에 주의가 필요할 때를 감지하며 응급시에는 통화 가능한 의사에게 알립니다.

IBM과 보건 의료 서비스 제공자 사이의 협업에는 다수의 텔레메트리 유스 케이스에 공통되는 다음과 같은 몇 가지 특징이 있습니다.

투명성

디바이스는 전원, 전화 회선을 공급하는 것과 디바이스의 범위 안에 있을 것을 제외하고 사용자 개입을 필요로 하지 않습니다. 조작은 안정적이며 사용하기 간단합니다.

환자가 디바이스를 설정해야 할 필요를 없애기 위해 디바이스 공급자는 디바이스를 사전 구성합니다. 환자는 전원을 연결하기만 하면 됩니다. 환자가 직접 구성하지 않도록 함으로써 조작이 간편해지며 디바이스가 잘못 구성될 확률이 줄어듭니다.

MQTT 클라이언트가 디바이스의 한 부분으로 임베드되어 있습니다. 디바이스 개발자는 MQTT 클라이언트 구현을 디바이스에 임베드하며 개발자 또는 공급자가 사전 구성의 일부로서 MQTT 클라이언트를 구성합니다.

MQTT 클라이언트는 Java SE JAR 파일로 제공되며 개발자는 이를 자신의 Java 애플리케이션에 포함시킵니다. 이 디바이스와 같은 비Java 환경에서 디바이스 개발자는 발행된 MQTT 포맷 및 프로토콜을 이용하여 다른 언어로 클라이언트를 구현할 수 있습니다. 또는, 개발자는 Windows, Linux 및 ARM 플랫폼을 위한 공유 라이브러리로 제공된 C 클라이언트 중 하나를 사용할 수 있습니다.

불규칙한 연결

제세동기와 병원 사이의 통신에는 불규칙한 네트워크 특성이 있습니다. 환자에게서 데이터를 수집하고, 데이터를 병원으로 송신하는 서로 다른 문제를 해결하기 위해 두 개의 서로 다른 네트워크가 사용됩니다. 환자와 MQTT 디바이스 사이에는 단거리 저전력 RF 네트워크가 사용됩니다. 송신기는 VPN TCP/IP 연결을 사용하여 낮은 대역폭 전화 회선으로 병원과 연결합니다.

모든 디바이스를 인터넷 프로토콜 네트워크와 직접 연결하는 방법을 찾는 것은 보통 실용적이지 않습니다. 허브로 연결된 두 네트워크를 사용하는 것이 보통 솔루션입니다. MQTT 디바이스는 간단한 허브로, 환자에게서 정보를 저장하고 이를 병원으로 전달합니다.

보안

의사는 환자 데이터의 신뢰성을 믿을 수 있어야 하며 환자는 개인정보인 자신의 데이터가 보호되기를 바랍니다.

몇몇 상황에서는 VPN 또는 TLS를 사용하여 연결을 암호화하는 것으로 충분합니다. 데이터가 저장된 후에도 이를 보호하는 것이 더 좋은 상황도 있습니다.

텔레메트리 디바이스가 안전하지 않은 경우도 있습니다. 예를 들면, 공동 주택에 있거나 하는 경우입니다. 데이터가 올바른 환자의 것임을 보장할 수 있도록 디바이스 사용자는 반드시 인증을 받아야 합니다. 디바이스 자체도 TLS를 사용하여 서버의 인증을 받을 수 있고, 서버 또한 디바이스의 인증을 받을 수 있습니다.

디바이스와 큐 관리자 사이의 텔레메트리 채널은 사용자 인증을 위해 JAAS를 지원하며 통신 암호화와 디바이스 인증을 위해 TLS를 지원합니다. 발행에 대한 액세스는 IBM MQ의 오브젝트 권한 관리자에 의해 제어됩니다.

사용자 인증에 사용된 ID는 공용 환자 ID와 같은 다른 ID에 맵핑될 수 있습니다. 공용 ID는 IBM MQ의 발행 토픽에 대한 권한 구성을 단순화합니다.

연결성

MQTT 디바이스와 병원 사이의 연결은 전화 접속을 사용하며 1초에 300보오 정도의 낮은 대역폭으로 작동합니다.

300보오에서 효율적으로 작업하기 위해 MQTT protocol은 TCP/IP 헤더 외에 추가 바이트를 메시지에 조금 추가합니다.

MQTT protocol은 단일 전송 전송 후 삭제 메시징을 제공하며 이는 대기 시간을 낮게 유지합니다. 또한 보장된 전달이 응답 시간보다 더 중요한 경우에는 적어도 한 번 및 정확히 한 번*e* 전달을 보장하기 위해 다중 전송을 사용할 수 있습니다. 전달을 보장하기 위해 메시지는 전달이 완료될 때까지 디바이스에 저장됩니다. 디바이스가 무선으로 연결되어 있을 경우 보장된 전달은 특히 유용합니다.

확장성

텔레메트리 디바이스는 보통 수만에서 수백만에 이르는 규모로 많은 수가 배치됩니다.

시스템에 많은 디바이스를 연결할 경우 많은 솔루션이 필요하게 됩니다. 디바이스와 그 소프트웨어에 대한 비용과 같은 비즈니스적 요구, 라이센스, 디바이스와 사용자를 관리하는 관리 요구가 이에 해당합니다. 기술적 요구는 네트워크나 서버의 부하를 포함합니다.

연결을 여는 것은 열린 연결을 유지하는 것보다 많은 서버 자원을 사용합니다. 하지만 전화 회선을 사용하는 이와 같은 유스 케이스에서 연결 비용은 필요할 때를 제외하고 연결이 열려 있지 않다는 것을 의미합니다. 데이터 전송은 대부분 일괄 처리로 이뤄집니다. 연결은 취침 시간에 갑자기 증가하는 연결을 피하기 위해 심야에 스케줄될 수 있습니다.

클라이언트 측면에서 클라이언트의 확장성은 필요한 클라이언트 구성이 적을수록 향상됩니다. MQTT 클라이언트는 디바이스에 임베드되어 있습니다. 디바이스를 환자에 배치하는 데 있어서 구성이나 MQTT 클라이언트 라이센스 수락 단계와 같은 요구사항은 없습니다.

서버 측면에서 MQ Telemetry에는 큐 관리자당 50,000개의 열린 연결이라는 초기 목표가 있습니다.

연결은 IBM MQ Explorer를 사용하여 관리됩니다. IBM MQ Explorer는 관리 가능한 숫자만을 표시하도록 연결을 필터합니다. 적절하게 선택된, 클라이언트에 대한 ID 할당 설계를 사용하면 지정학적으로, 또는 환자 이름의 영문자 순서대로 연결을 필터할 수도 있습니다.

Windows Linux AIX 텔레메트리 유스 케이스: 가정 에너지 모니터링 및 제어

스마트 측정기는 기존 측정기보다 에너지 소비에 대한 세부 사항을 더 많이 수집합니다.

스마트 측정기는 가정의 개별 어플라이언스를 모니터 및 제어하기 위해 보통 로컬 텔레메트리 네트워크와 결합됩니다. 몇몇은 원거리에서 모니터링과 제어를 할 수 있도록 원격으로도 연결됩니다.

원격 연결은 개별 기기, 전원 유틸리티 및 중앙 제어 지점에 의해 설정될 수 있습니다. 원격 제어 지점은 전력 사용을 읽고 사용 데이터를 제공할 수 있습니다. 이는 연속적인 가격 설정이나 날씨 정보에 영향을 줄 수 있는 데이터를 제공할 수 있습니다. 전체적인 전력 생산 효율을 높이기 위해 부하를 제한할 수도 있습니다.

스마트 측정기는 점점 더 널리 보급되고 있습니다. 예를 들면, 영국 정부는 2020년까지 모든 국내 가정에 스마트 측정기를 보급하는 것에 대해 협의 중에 있습니다.

가정 측정 유스 케이스에는 다음과 같은 몇 가지 공용되는 특성이 있습니다.

투명성

사용자가 측정기를 사용하여 에너지를 절약하고자 하지 않는 이상 측정기는 사용자 개입을 필요로 하지 않습니다. 이는 개별 어플라이언스에 대한 에너지 공급 안정성을 저하시키지 않아야 합니다.

MQTT 클라이언트는 측정기와 함께 배치된 소프트웨어에 임베드될 수 있으며 별도의 설치나 구성이 필요하지 않습니다.

불규칙한 연결

어플라이언스와 스마트 측정기는 측정기와 원격 연결 지점 사이의 연결과는 다른 연결성 기준을 요구합니다.

스마트 측정기에서 어플라이언스로의 연결은 가용성이 높아야 하며 홈 네트워크의 네트워크 기준에 부합해야 합니다.

원격 네트워크는 다양한 실제 접속을 사용할 가능성이 높습니다. 이 중 무선과 같은 몇몇은 전송 비용이 높으며 단속적일 수 있습니다. MQTT v3 스펙은 원격 연결 및 로컬 어댑터와 스마트 측정기 사이의 연결을 주요 대상으로 하고 있습니다.

전원 콘센트와 어플라이언스, 그리고 측정기 사이의 연결은 Zigbee와 같은 홈 네트워크를 사용합니다. 센서 네트워크용 MQTT(MQTT-S)는 Zigbee 및 기타 낮은 대역폭 네트워크 프로토콜과 작업하도록 설계되었습니다. MQ Telemetry는 MQTT-S를 직접 지원하지 않습니다. MQTT-S를 MQTT v3에 연결하려면 게이트웨이가 필요합니다.

가정 환자 모니터링과 마찬가지로 가정 에너지 모니터링 및 제어에 대한 솔루션은 다중 네트워크를 필요로 하며 이는 스마트 측정기를 허브로 하여 연결되어 있어야 합니다.

보안

스마트 측정기와 연관된 보안 문제에는 몇 가지 있습니다. 이 문제에는 트랜잭션 부인 방지, 시작된 모든 제어 조치의 권한 부여와 전력 소비 데이터의 개인정보 보호가 포함되어 있습니다.

확실한 개인정보 보호를 위해 측정기와 원격 제어 지점 사이에 MQTT로 전송된 데이터는 TLS를 사용하여 암호화될 수 있습니다. 제어 조치에 대한 확실한 권한 부여를 위해 측정기와 원격 제어 지점 사이의 MQTT 연결은 TLS를 사용하여 상호 인증하도록 할 수 있습니다.

연결성

원격 네트워크의 실제 환경은 매우 다양할 수 있습니다. 기존 광대역 연결을 사용하거나 높은 비용을 지불하며 모바일 네트워크를 사용할 수도 있고, 가용성은 단속적일 수 있습니다. 고비용이며 간헐적인 연결에 대해 MQTT는 효율적이며 안정적인 프로토콜입니다. 99 페이지의 『텔레메트리 유스 케이스: 가정 환자 모니터링』의 내용을 참조하십시오.

확장성

전력 회사나 중앙 제어 지점은 언젠가 수천만 개의 스마트 측정기를 설치하려고 계획하고 있습니다. 처음 배치 당 측정기의 수는 수만에서 수십만 단위입니다. 이 숫자는 초기 MQTT 목표였던 큐 관리자당 50,000개의 열린 클라이언트 연결과 비슷합니다.

가정 에너지 모니터링 및 제어의 아키텍처가 가진 중요한 면은 스마트 측정기를 네트워크 집선기로서 사용한다는 점입니다. 각 어플라이언트 어댑터는 별도의 센서입니다. 이들을 MQTT를 사용하여 로컬 허브에 연결함으로써 허브는 중앙 제어 지점과의 단일 TCP/IP 세션으로 데이터 플로우를 집중시킬 수 있으며 세션 정전 시를 대비해 짧은 시간 동안 메시지를 저장할 수도 있습니다.

가정 에너지 유스 케이스에서 원격 연결은 두 가지 이유로 인해 열려 있어야 합니다. 첫째, 연결을 여는 것은 요청을 송신하는 것과 비교해 시간이 오래 걸리기 때문입니다. 짧은 간격 안에 "부하 제한" 요청을 보내기 위해 많은 연결을 여는 데는 시간이 너무 오래 걸립니다. 둘째, 전력 회사로부터 부하 제한 요청을 수신하려면 클라이언트로부터 먼저 연결이 열려야 합니다. MQTT를 사용하면 연결은 항상 클라이언트에 의해 시작되며 전력 회사로부터 부하 제한 요청을 수신하기 위해서 연결은 열린 채로 남겨져 있어야 합니다.

연결 열기 등급이 중요이거나 서버가 시간에 쫓기는 요청을 시작할 경우 보통 솔루션은 많은 열린 연결을 유지하는 것입니다.

Windows Linux AIX 텔레메트리 유스 케이스: RFID(Radio Frequency Identification)

RFID는 오브젝트를 무선으로 식별하고 추적하는 임베드된 RFID 태그 사용법입니다. RFID 태그는 수 미터 범위 안에서, 태그와 PFID 리더 사이에 물체가 있더라도 읽을 수 있습니다. 수동 태그는 RFID 리더에 의해 활성화됩니다. 활성 태그는 외부에서 작동시키지 않더라도 전송됩니다. 활성 태그에는 전원이 있어야 합니다. 수동 태그는 그 범위를 늘리기 위해 전원이 있을 수 있습니다.

RFID는 많은 애플리케이션과 수많은 유형의 유스 케이스에서 사용됩니다. RFID 유스 케이스에는 가정 환자 모니터링, 가정 에너지 모니터링, 제어 유스 케이스와 몇 가지 유사한 점과 다른 점이 있습니다.

투명성

많은 유스 케이스에서 RFID 리더는 다수 배치되며 사용자 개입 없이 작동해야 합니다. 리더는 중앙 제어 지점과의 통신을 위한 임베드된 MQTT 클라이언트를 포함하고 있습니다.

예를 들면, 유통 창고에서 리더는 팰릿을 감지하는 데 동작 센서를 사용합니다. 이는 팰릿 위에 놓인 물건의 RFID 태그를 활성화시키며 데이터와 요청을 중앙 애플리케이션으로 송신합니다. 이 데이터는 재고의 위치를 업데이트하는 데 사용됩니다. 요청은 팰릿에 수행될 다음 작업(예: 특정 공간으로 이를 이동시키는 등)을 제어합니다. 항공사나 공항 수화물 시스템은 이런 방식으로 RFID를 사용합니다.

일부 RFID 유스 케이스에서는 리더에 표준 컴퓨팅 환경(예: Java ME(Java Platform, Micro Edition))이 있습니다. 이러한 경우 MQTT 클라이언트는 제조 후 고유한 구성 단계를 거쳐 배치될 수 있습니다.

불규칙한 연결

RFID 리더는 MQTT 클라이언트를 포함하고 있는 로컬 제어 디바이스와 분리될 수 있으며, 각 리더가 MQTT 클라이언트를 임베드할 수도 있습니다. 일반적으로 지형적 또는 통신상의 요인이 토폴로지를 결정하게 됩니다.

보안

RFID 태그를 사용에 있어서 개인정보 보호 및 신뢰성은 주요 보안 대상입니다. RFID 태그는 눈에 잘 띄지 않으며 비밀리에 모니터, 위조나 도용의 대상이 될 수 있습니다.

RFID 보안 문제에 대한 솔루션은 새 RFID 솔루션이 배치될 기회를 증가시킵니다. 그러나 보안 취약점은 주로 RFID 태그에 있으며, 중앙 정보 처리를 사용하는 로컬 리더는 다양한 위협을 제거할 방법을 제시해 줍니다. 예를 들면, 태그 도용은 입고와 출고에 대해 동적으로 상호 연관된 재고량을 통해 발견될 수 있습니다.

연결성

RFID 애플리케이션은 보통 RFID 리더에서 수집한 일괄 처리되는 저장 후 전달 정보, 그리고 즉각적 조회를 모두 포함하고 있습니다. 유통 창고 유스 케이스에서 RFID 리더는 항상 연결되어 있습니다. 태그가 읽히면 이는 리더에 관한 정보와 함께 발행됩니다. 창고 애플리케이션은 리더에게 응답을 발행합니다.

창고 애플리케이션에서 네트워크는 일반적으로 안정적이며 즉각적 요청은 낮은 지연시간 성능을 위해 전송 후 삭제 메시지를 사용할 수 있습니다. 일괄처리되는 저장 후 전달 데이터는 데이터 삭제와 연관된 관리 비용을 최소화하기 위해 정확히 한 번 메시징을 사용할 수 있습니다.

확장성

RFID 애플리케이션이 1 - 2초 정도 소요되는 즉각적 응답을 필요로 할 경우 RFID 리더는 계속 연결되어 있어야 합니다.

Windows Linux AIX 텔레메트리 유스 케이스: 환경 감지

환경 감지는 강물의 깊이와 수질, 대기 오염원 및 기타 환경 데이터를 수집하는 데 텔레메트리를 사용합니다.

센서는 보통 외진 곳에, 유선 통신에 대한 액세스 없이 설치됩니다. 무선 대역폭은 비경제적이며 안정성은 낮을 수 있습니다. 작은 공간에 있는 다수의 환경 센서가 안전한 장소의 로컬 모니터링 디바이스에 연결되어 있는 것이 전형입니다. 로컬 연결은 유선 또는 무선입니다.

투명성

센서 디바이스는 보통 중앙 모니터링 디바이스보다 액세스하기 어렵고, 전력을 덜 사용하며 다수 배치됩니다. 센서는 때때로 "조용하며" 로컬 모니터링 디바이스는 센서 데이터를 변환하고 저장하는 어댑터를 포함하고 있습니다. 모니터링 디바이스에는 Java SE(Java Platform, Standard Edition) 또는 Java ME(Java Platform, Micro Edition)를 지원하는 범용 컴퓨터가 포함되어 있을 가능성이 큽니다. 투명성은 MQTT 클라이언트를 구성할 때 주요 요구사항은 아닙니다.

불규칙한 연결

센서의 기능, 원격 연결의 비용과 대역폭 때문에 보통 로컬 모니터링 허브는 중앙 서버에 연결됩니다.

보안

솔루션이 군사 또는 방어 관련 유스 케이스에서 사용되는 것이 아닐 경우 보안은 주요 요구사항이 아닙니다.

연결성

대부분의 경우 연속적 모니터링이나 데이터의 즉시 가용성이 필요하지 않습니다. 홍수 레벨 경보와 같은 예외 데이터는 즉시 전달되어야 할 필요가 있습니다. 센서 데이터는 연결 및 통신 비용 절감을 위해 로컬 모니터에서 집계되어 스케줄된 연결을 사용하여 전송됩니다. 예외 데이터는 모니터에서 감지되는 즉시 전달됩니다.

확장성

센서는 로컬 허브에 집중되어 있으며 센서 데이터는 스케줄에 따라 전송되는 패킷에 집계됩니다. 이 두 요소가 직접 연결된 센서를 사용할 때 발생할 수 있는 중앙 서버의 부담을 덜어줍니다.

Windows Linux AIX 텔레메트리 유스 케이스: 모바일 애플리케이션

모바일 애플리케이션은 무선 디바이스에서 실행되는 애플리케이션입니다. 이 디바이스는 일반 애플리케이션 플랫폼이거나 특수 디바이스입니다.

일반 플랫폼에는 휴대전화나 PDA와 같은 소형 디바이스나 노트북 컴퓨터와 같은 휴대용 디바이스가 포함됩니다. 특수 디바이스는 특정 애플리케이션에 맞춰진 특수 목적 하드웨어를 사용합니다. 특수 모바일 디바이스의 한 예로는 "전달 확인" 소포 전달을 기록하는 디바이스가 있습니다. 특수 모바일 디바이스의 애플리케이션은 보통 일반 소프트웨어 플랫폼에서 빌드됩니다.

투명성

사용자 정의 모바일 애플리케이션의 배치가 관리되며 MQTT 클라이언트 애플리케이션 구성을 포함할 수 있습니다. 투명성은 MQTT 클라이언트를 구성하는 데 주된 요구사항은 아닙니다.

불규칙한 연결

앞에 나온 유스 케이스의 로컬 허브 토폴로지와 달리 모바일 클라이언트는 원격으로 연결합니다. 클라이언트 애플리케이션 계층은 중앙 허브의 애플리케이션으로 직접 연결합니다.

보안

물리적 보안성이 높지 않기 때문에 모바일 디바이스와 모바일 사용자는 인증을 받아야 합니다. 디바이스 ID를 확인하는 데는 TLS가, 사용자를 인증하는 데는 JAAS가 사용됩니다.

연결성

모바일 애플리케이션이 무선 범위에 종속적일 경우 이는 오프라인으로 조작이 가능하며 인터럽트된 연결을 효율적으로 다룰 수 있어야 합니다. 이와 같은 환경에서 목표는 연결을 유지하는 동시에 애플리케이션에서 메시지를 저장하고 전달할 수 있게 하는 것입니다. 메시지는 종종 명령이나 전달 확인인 경우가 있으며 중요한 비즈니스 가치가 있습니다. 이들은 안정적으로 저장 및 전달되어야 합니다.

확장성

확장성은 주요 문제는 아닙니다. 사용자 정의 모바일 애플리케이션 유스 케이스에서 애플리케이션 클라이언트의 수는 수천 또는 수만을 넘어가지 않을 가능성이 큽니다.

Windows Linux AIX 텔레메트리 디바이스를 큐 관리자에 연결

텔레메트리 디바이스는 MQTT v3 클라이언트를 사용하여 큐 관리자에 연결합니다. MQTT v3 클라이언트는 TCP/IP를 사용하여 텔레메트리(MQXR) 서비스라고 하는 TCP/IP 리스너에 연결합니다.

텔레메트리 디바이스를 큐 관리자에 연결하는 경우, MQTT 클라이언트는 `MqttClient.connect` 메소드를 사용하여 TCP/IP 연결을 시작합니다. IBM MQ 클라이언트처럼 MQTT 클라이언트는 메시지를 송신하고 수신하려면 큐 관리자에 연결되어 있어야 합니다. 연결은 텔레메트리(MQXR) 서비스라고 하는 MQ Telemetry와 함께 설치된 TCP/IP 리스너를 사용하여 서버에서 이루어집니다. 각각의 큐 관리자는 최대로 하나의 텔레메트리(MQXR) 서비스를 실행합니다.

텔레메트리(MQXR) 서비스는 텔레메트리 채널에 대한 연결을 할당하기 위해 `MqttClient.connect` 메소드에서 각 클라이언트가 설정한 원격 소켓 주소를 사용합니다. 소켓 주소는 TCP/IP 호스트 이름과 포트 번호의 조합입니다. 동일한 원격 소켓 주소를 사용하는 다수의 클라이언트는 텔레메트리(MQXR) 서비스에 의해 동일한 텔레메트리 채널에 연결됩니다.

서버에 다수의 큐 관리자가 있을 경우 큐 관리자 사이에 텔레메트리 채널을 분할하십시오. 큐 관리자 사이에 원격 소켓 주소를 할당하십시오. 각 텔레메트리 채널을 고유한 원격 소켓 주소로 정의하십시오. 두 텔레메트리 채널은 같은 소켓 주소를 사용해서는 안됩니다.

다수의 큐 관리자에 텔레메트리 채널에 대해 같은 소켓 주소가 구성되어 있을 경우 처음 연결하는 텔레메트리 채널이 남게 됩니다. 동일한 주소에 연결하는 후속 채널은 실패합니다.

서버에 다수의 네트워크 어댑터가 있을 경우 텔레메트리 채널 사이에 원격 소켓 주소를 분할하십시오. 하나의 특정 소켓 주소라도 유일한 텔레메트리 채널에 구성되어 있는 한 소켓 주소 할당은 완전히 임의로 이뤄집니다.

IBM MQ Explorer용 MQ Telemetry 부록에 제공된 마법사를 사용하여 MQTT 클라이언트를 연결하도록 IBM MQ 를 구성하십시오. 또는 수동으로 텔레메트리를 구성하려면 Linux 및 AIX에서 텔레메트리에 대해 큐 관리자 구성 및 Windows에서 텔레메트리에 대해 큐 관리자 구성의 지시사항을 따르십시오.

관련 참조
MQXR 특성

Windows Linux AIX 텔레메트리 연결 프로토콜

MQ Telemetry는 TCP/IP IPv4 및 IPv6과 TLS를 지원합니다.

Windows Linux AIX 텔레메트리(MQXR) 서비스

텔레메트리(MQXR) 서비스는 TCP/IP 리스너이며, IBM MQ 서비스로 관리됩니다. IBM MQ Explorer 마법사나 **runmqsc** 명령을 사용하여 서비스를 작성하십시오.

MQ Telemetry (MQXR) 서비스를 SYSTEM.MQXR.SERVICE .

IBM MQ Explorer용 MQ Telemetry 함수에서 제공되는 텔레메트리 샘플 구성 마법사는 텔레메트리 서비스 및 샘플 텔레메트리 채널을 작성합니다. IBM MQ Explorer 를 사용하여 MQ Telemetry 설치 확인 을 참조하십시오.

명령행에서 샘플 구성을 작성하십시오. 명령행을 사용한 MQ Telemetry 설치 확인을 참조하십시오.

텔레메트리(MQXR) 서비스는 큐 관리자에서 자동으로 시작되고 중지됩니다. IBM MQ Explorer의 서비스 폴더를 사용하여 서비스를 제어하십시오. 서비스를 보려면 아이콘을 클릭하여 IBM MQ Explorer 표시에서 SYSTEM 오브젝트 필터링을 중지해야 합니다.

서비스를 수동으로 작성하는 방법에 대한 예제는 다음을 참조하십시오.

- Linux AIX Linux에서 SYSTEM.MQXR.SERVICE 작성.
- Windows Windows에서 SYSTEM.MQXR.SERVICE 작성.

이 주제에서는 MQTT TLS 채널에 대한 비밀번호 문구를 암호화해야 하는 기본 키도 지정합니다. 자세한 정보는 MQTTTLS 채널에 대한 비밀번호 문구의 암호화를 참조하십시오.

Windows Linux AIX 텔레메트리 채널

Java 인증 및 권한 부여 서비스(JAAS)나 TLS 인증과 같은 다양한 특성과 연결을 작성하기 위해서나 클라이언트 그룹을 관리하려면 텔레메트리 채널을 작성하십시오.

IBM MQ Explorer용 MQ Telemetry 함수에 제공된 **New Telemetry Channel** 마법사를 사용하여 텔레메트리 채널을 작성하십시오. 특정 TCP/IP의 MQTT 클라이언트로부터 연결을 허용하려면 마법사를 사용하여 채널을 구성하십시오. IBM WebSphere MQ 7.1부터 명령행 프로그램 **runmqsc**를 사용하여 MQ Telemetry 를 구성할 수 있습니다.

클라이언트를 그룹으로 나눠 대규모 클라이언트 연결을 관리하기 쉽도록 서로 다른 포트에 다중 텔레메트리 채널을 작성하십시오. 각 텔레메트리 채널은 서로 다른 이름을 가지고 있습니다.

다양한 유형의 연결을 작성하기 위해 다양한 보안 속성으로 텔레메트리 채널을 구성할 수 있습니다. 다양한 TCP/IP 주소로부터 클라이언트 연결을 수락하려면 다중 채널을 작성하십시오. TLS를 사용하여 메시지를 암호화하고 텔레메트리 채널 및 클라이언트를 인증하십시오. MQTT 클라이언트 및 텔레메트리 채널의 TLS 구성을 참조하십시오. IBM MQ 오브젝트에 대한 액세스 권한을 부여하는 작업을 단순화하려면 사용자 ID를 지정하십시오. JAAS로 MQTT 사용자를 인증할 수 있도록 JAAS 구성을 지정하십시오. MQTT 클라이언트 식별, 권한 부여 및 인증을 참조하십시오.

Windows Linux AIX IBM MQ Telemetry Transport 프로토콜

IBM MQ Telemetry Transport (MQTT) v3 프로토콜은 낮은 대역폭 또는 비용이 많이 드는 연결에서 소형 디바이스 간에 메시지를 교환하고 메시지를 안정적으로 전송하도록 설계되었습니다. TCP/IP를 사용합니다.

MQTT protocol은 발행됩니다. IBM MQ Telemetry Transport 형식 및 프로토콜을 참조하십시오. 프로토콜 버전 3은 발행/구독을 사용하며 다음과 같은 세 가지 서비스 품질을 제공합니다. 전송 후 삭제, 적어도 한 번 및 정확히 한 번

프로토콜 헤더의 작은 크기와 바이트 배열 메시지 페이로드로 인해 메시지 용량은 작습니다. 헤더는 2바이트의 고정된 헤더와 12바이트까지의 추가 변수 헤더를 포함합니다. 프로토콜은 구독 및 연결을 하는 데 12바이트의 변수 헤더를 사용하며 대부분의 발행에 대한 변수 헤더에는 2바이트만을 사용합니다.

세 가지 서비스 품질(QoS)을 사용하여 낮은 지연 시간과 안정성 사이에서 적절하게 균형을 유지할 수 있습니다. MQTT 클라이언트가 제공하는 서비스 품질(QoS)을 참조하십시오. 전송 후 삭제는 지속적 디바이스 스토리지를 사용하지 않으며 발행물 송신 및 수신에 하나의 전송만을 사용합니다. 적어도 한 번 및 정확히 한 번에는 인식될 때까지 프로토콜 상태를 유지하고 메시지를 저장할 지속적 스토리지가 디바이스에 필요합니다.

Windows Linux AIX MQTT 클라이언트

MQTT 클라이언트 애플리케이션은 Telemetry 디바이스에서 정보를 수집하고 서버에 연결하며 정보를 서버에 발행하는 역할을 합니다. 또한 토픽을 구독하고 서적을 수신하며 텔레메트리 디바이스를 제어할 수도 있습니다.

IBM MQ 클라이언트 애플리케이션과 달리 MQTT 클라이언트 애플리케이션은 IBM MQ 애플리케이션이 아닙니다. 이 클라이언트는 연결할 큐 관리자를 지정하지 않습니다. 특정 IBM MQ 프로그래밍 인터페이스를 사용해야 하는 제약도 받지 않습니다. 대신에, MQTT 클라이언트는 MQTT 3 프로토콜을 구현합니다. 프로그래밍 언어를 사용하여 선택한 플랫폼에서 MQTT protocol에 대한 클라이언트 라이브러리의 인터페이스를 작성할 수 있습니다. IBM MQ Telemetry Transport 형식 및 프로토콜을 참조하십시오.

MQTT 클라이언트 앱 작성을 단순화하려면 여러 플랫폼에 대해 MQTT protocol 를 캡슐화하는 C, Java및 JavaScript 클라이언트 라이브러리를 사용하십시오. 이러한 라이브러리를 MQTT 앱에 통합하면 완전한 기능을 갖춘 MQTT 클라이언트가 15행의 코드로 짧을 수 있습니다. MQTT 클라이언트 라이브러리는 Eclipse Paho 및 MQTT.org에서 무료로 제공됩니다. IBM MQ Telemetry Transport 샘플 프로그램을 참조하십시오.

MQTT 클라이언트 애플리케이션은 항상 텔레메트리 채널로 연결을 시작합니다. 연결된 후 MQTT 클라이언트 애플리케이션이나 IBM MQ 애플리케이션 모두에서 메시지 교환을 시작할 수 있습니다.

MQTT 클라이언트 애플리케이션과 IBM MQ 애플리케이션은 동일한 토픽 세트에 발행 및 구독합니다. IBM MQ 애플리케이션은 클라이언트 애플리케이션이 구독을 먼저 작성하지 않고 MQTT 클라이언트 애플리케이션에 직접 메시지를 송신할 수 있습니다. MQTT 클라이언트에 메시지를 송신하도록 분산 큐잉 구성을 참조하십시오.

MQTT 클라이언트 애플리케이션은 텔레메트리 채널을 사용하여 IBM MQ에 연결됩니다. 텔레메트리 채널은 MQTT 및 IBM MQ가 사용하는 서로 다른 유형의 메시지 사이에서 브릿지 역할을 수행합니다. 이는 MQTT 클라이언트 애플리케이션을 대신하여 큐 관리자에서 발행 및 구독을 작성합니다. 텔레메트리 채널은 MQTT 클라이언트 애플리케이션의 구독과 일치하는 발행을 큐 관리자에서 MQTT 클라이언트 애플리케이션으로 송신합니다.

Windows Linux AIX MQTT 클라이언트에 메시지 송신

IBM MQ 애플리케이션은 클라이언트가 작성한 구독을 발행하거나 직접 메시지를 보내서 MQTT v3 클라이언트 메시지를 보낼 수 있습니다. MQTT 클라이언트는 다른 클라이언트가 구독한 토픽을 발행하는 방식으로 메시지를 다른 클라이언트에게 보낼 수 있습니다.

MQTT 클라이언트가 IBM MQ에서 수신한 발행을 구독함

IBM MQ 에서 MQTT 클라이언트로 발행물을 보내려면 108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』 태스크를 수행하십시오.

MQTT v3 클라이언트가 메시지를 수신하는 일반적인 방법은 토픽 또는 토픽 세트에 구독을 작성하는 것입니다. 이 예제 코드 스니펫 106 페이지의 그림 44에서는 MQTT 클라이언트가 토픽 문자열 `"MQTT Examples"`을 사용하여 구독합니다. IBM MQ C 애플리케이션, 107 페이지의 그림 45는 토픽 문자열 `"MQTT Examples"`를 사용하여 토픽을 발행합니다. 이 코드 스니펫 107 페이지의 그림 46에서는 MQTT 클라이언트가 콜백 메소드 `messageArrived`로 발행을 수신합니다.

MQTT 클라이언트의 구독에 대한 응답으로 발행물을 송신하도록 IBM MQ 를 구성하는 방법에 대한 자세한 정보는 MQTT 클라이언트 구독에 대한 응답으로 메시지 발행을 참조하십시오.

IBM MQ 애플리케이션이 MQTT 클라이언트로 직접 메시지를 보냄

IBM MQ 에서 MQTT 클라이언트로 직접 메시지를 보내려면 112 페이지의 『IBM MQ Explorer 를 사용하여 MQTT 클라이언트에 메시지 송신』 태스크를 수행하십시오.

이러한 방식으로 MQTT 클라이언트로 보낸 메시지를 요청하지 않은 메시지라고 합니다. MQTT v3 클라이언트는 토픽 이름 세트가 포함된 발행으로 요청하지 않은 메시지를 수신합니다. 텔레메트리(MQXR) 서비스는 토픽 이름을 리모트 큐 이름으로 설정합니다.

MQTT 클라이언트에 직접 메시지를 보내도록 IBM MQ 를 구성하는 방법에 대한 자세한 정보는 클라이언트에 직접 메시지 보내기를 참조하십시오.

MQTT 클라이언트가 메시지를 발행함

MQTT v3 클라이언트는 다른 MQTT v3 클라이언트가 수신한 메시지를 발행할 수 있지만 요청하지 않은 메시지는 보낼 수 없습니다. 코드 스니펫 107 페이지의 그림 47 에서는 Java로 작성된 MQTT v3 클라이언트가 메시지를 발행하는 방법을 보여줍니다.

특정 MQTT v3 클라이언트로 메시지를 보내는 일반적인 방법은 각 클라이언트가 고유의 `ClientIdentifier` 에 구독을 작성하는 것입니다. `ClientIdentifier` 를 토픽 문자열로 사용하여 한 MQTT 클라이언트에서 다른 MQTT 클라이언트로 메시지를 발행하려면 113 페이지의 『특정 MQTT v3 클라이언트로 메시지 발행』 태스크를 수행하십시오.

예제 코드 스니펫

106 페이지의 그림 44 의 코드 스니펫은 Java 로 작성된 MQTT 클라이언트가 구독을 작성하는 방법을 보여줍니다. 구독에 대한 발행을 수신하려면 `messageArrived`라는 콜백 메소드도 필요합니다.

```
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);
```

그림 *44. MQTT v3* 클라이언트 구독자

107 페이지의 그림 45의 코드 스니펫에서는 C로 작성된 IBM MQ 애플리케이션이 발행을 보내는 방법을 보여줍니다. 코드 스니펫은 변수 토픽에 발행자 작성 태스크를 통해 추출됩니다.

```
/* 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);
```

그림 *45. IBM MQ* 발행자

발행물이 도착하면 MQTT 클라이언트는 MQTT 애플리케이션 클라이언트 `MqttCallback` 클래스의 `messageArrived` 메소드를 호출합니다.

```
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
}
```

그림 *46. messageArrived* 메소드

107 페이지의 그림 47에서는 MQTT v3가 106 페이지의 그림 44에서 작성된 구독에 메시지를 발행하는 방법을 보여줍니다.

```
      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);
```

그림 *47. MQTT v3* 클라이언트 발행자

Windows Linux AIX IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행

IBM MQ Explorer를 사용하여 메시지를 발행하고 MQTT 클라이언트 유틸리티를 사용하여 해당 메시지를 구독하려면 다음 태스크의 단계를 수행하십시오. 다음 태스크에서는 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정하지 않고 큐 관리자 알리어스를 구성하는 방법을 보여줍니다.

시작하기 전에

이 태스크에서는 사용자가 IBM MQ 및 IBM MQ Explorer에 익숙하고 IBM MQ 및 MQ Telemetry 기능이 설치되어 있다고 가정합니다.

이 태스크에서 큐 관리자 자원을 작성하는 사용자는 이에 대한 충분한 권한을 가지고 있어야 합니다. 데모를 위해 IBM MQ Explorer 사용자 ID가 mqm 그룹의 구성원이라고 가정합니다.

이 태스크 정보

이 태스크에서는 IBM MQ에 토픽을 작성하고 MQTT 클라이언트 유틸리티를 사용하여 해당 토픽을 구독합니다. IBM MQ Explorer를 사용하여 토픽을 발행하면 MQTT 클라이언트가 발행을 수신합니다.

프로시저

다음 태스크 중 하나를 수행하십시오.

- MQ Telemetry를 설치했지만 아직 시작하지 않았습니다. 108 페이지의 『텔레메트리(MQXR) 서비스가 아직 정의되지 않은 태스크 시작』 태스크를 수행하십시오.
- 이전에 IBM MQ Telemetry를 실행했지만 데모를 위해 새 큐 관리자를 사용하려고 합니다. 108 페이지의 『텔레메트리(MQXR) 서비스가 아직 정의되지 않은 태스크 시작』 태스크를 수행하십시오.
- 텔레메트리 자원이 정의되지 않은 기존의 큐 관리자를 사용하는 태스크를 수행하려고 합니다. **샘플 구성 정의** 마법사를 실행하지 않으려고 합니다.

  a. 텔레메트리를 설정하려면 다음 태스크 중 하나를 수행하십시오.

  - Linux 및 AIX에서 텔레메트리에 대해 큐 관리자 구성
  - Windows에서 텔레메트리에 대해 큐 관리자 구성

  b. 109 페이지의 『실행 중인 텔레메트리(MQXR) 서비스가 있는 태스크 시작』 태스크를 수행하십시오.
- 텔레메트리 자원이 이미 정의되어 있는 기존의 큐 관리자를 사용하는 태스크를 수행하려는 경우 109 페이지의 『실행 중인 텔레메트리(MQXR) 서비스가 있는 태스크 시작』 태스크를 수행하십시오.

다음에 수행할 작업

클라이언트 유틸리티로 직접 메시지를 보내려면 112 페이지의 『IBM MQ Explorer 를 사용하여 MQTT 클라이언트에 메시지 송신』 태스크를 수행하십시오.

텔레메트리(*MQXR*) 서비스가 아직 정의되지 않은 태스크 시작

큐 관리자를 작성하고 **샘플 구성 정의**를 실행하여 큐 관리자에 대한 샘플 텔레메트리 자원을 정의합니다. IBM MQ Explorer를 사용하여 메시지를 발행하고 MQTT 클라이언트 유틸리티로 해당 메시지를 구독합니다.

이 태스크 정보

샘플 구성 정의 마법사를 사용하여 샘플 텔레메트리 자원을 설정할 때 해당 마법사가 게스트 사용자 ID 권한을 설정합니다. 이러한 방식으로 게스트 사용자 ID에 권한을 부여할지 신중하게 고려하십시오. Windows의 guest 및 Linux의 nobody 에는 토픽 트리의 루트를 발행하고 구독하며 SYSTEM.MQTT.TRANSMIT.QUEUE에 메시지를 넣을 수 있는 권한이 부여됩니다.

또한 마법사는 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정하는데, 이는 기존의 큐 관리자에서 실행 중인 애플리케이션에 지장을 줄 수 있습니다. 기본 전송 큐를 사용하지 않고 텔레메트리를 구성하는 것은 가능하지만 번거로운 작업입니다. 110 페이지의 『큐 관리자 알리어스 사용』 태스크에 따라 작업을 수행하십시오. 이 태스크에서는 큐 관리자를 작성하여 기존의 기본 전송 큐에 지장을 주지 않도록 합니다.

프로시저

1. IBM MQ Explorer를 사용하여 새 큐 관리자를 작성하고 시작하십시오.
   a) Queue Managers 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **큐 관리자...**를 클릭하십시오. 큐 관리자 이름을 입력하고 **마침**을 클릭하십시오.

      큐 관리자 이름을 작성하십시오(예: MQTTQMGR).
2. 텔레메트리(MQXR) 서비스를 작성 및 시작하고 샘플 텔레메트리 채널을 작성하십시오.
   a) Queue Managers*QmgrName*\Telemetry 폴더를 여십시오.
   b) **샘플 구성 정의...** > **마침**을 클릭하십시오.

      MQTT 클라이언트 유틸리티 시작 선택란을 선택하십시오.
3. MQTT 클라이언트 유틸리티를 사용하여 MQTT Example 에 대한 구독을 작성하십시오.
   a) **연결**을 클릭하십시오.

      클라이언트 실행 기록은 Connected 이벤트를 기록합니다.
   b) MQTT Example을 **Subscription\Topic** 필드에 입력하고 **구독**을 입력하십시오.

   클라이언트 실행 기록은 Subscribed 이벤트를 기록합니다.
4. IBM MQ에서 MQTTExampleTopic을 작성하십시오.
   a) **MQ 탐색기**에서 Queue Managers*QmgrName*\Topics 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **토픽**을 클릭하십시오.
   b) MQTTExampleTopic을 **이름**으로 입력하고 **다음**을 클릭하십시오.
   c) MQTT Example을 **토픽 문자열**로 입력하고 **마침**을 클릭하십시오.
   d) 수신확인 창을 닫으려면 **확인**을 클릭하십시오.
5. IBM MQ Explorer를 사용하여 Hello World! 를 MQTT Example 토픽에 발행하십시오.
   a) IBM MQ Explorer에서 Queue Managers*QmgrName*\Topics 폴더를 클릭하십시오.
   b) MQTTExampleTopic을 마우스 오른쪽 단추로 클릭하고 **발행물 테스트...**를 클릭하십시오.
   c) **메시지 데이터** 필드에 Hello World! 를 입력하고 > **메시지 발행** > MQTT 클라이언트 유틸리티 창으로 전환하십시오.

   클라이언트 실행 기록은 Received 이벤트를 기록합니다.

실행 중인 텔레메트리*(MQXR)* 서비스가 있는 태스크 시작

텔레메트리 채널 및 토픽을 작성합니다. 사용자에게 토픽 및 텔레메트리 전송 큐를 사용할 수 있는 권한을 부여합니다. IBM MQ Explorer를 사용하여 메시지를 발행하고 MQTT 클라이언트 유틸리티로 해당 메시지를 구독합니다.

시작하기 전에

이 태스크 버전에서는 큐 관리자 *QmgrName*이 정의되어 실행되고 있습니다. 텔레메트리(MQXR) 서비스가 정의되어 실행 중입니다. 텔레메트리(MQXR) 서비스는 수동으로 또는 **샘플 구성 정의** 마법사를 실행하여 작성되었을 수 있습니다.

이 태스크 정보

이 태스크에서는 기존 큐 관리자를 구성하여 발행을 MQTT 클라이언트 유틸리티로 보냅니다.

태스크의 110 페이지의 『1』 단계에서는 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정하는데, 이는 기존의 큐 관리자에서 실행 중인 애플리케이션에 지장을 줄 수 있습니다. 기본 전송 큐를 사용하지 않고 텔레메트리를 구성하는 것은 가능하지만 번거로운 작업입니다. 110 페이지의 『큐 관리자 알리어스 사용』 태스크에 따라 작업을 수행하십시오.

프로시저

1. SYSTEM.MQTT.TRANSMIT.QUEUE를 기본 전송 큐로 설정하십시오.
   a) Queue Managers*QmgrName* folder를 마우스 오른쪽 단추로 클릭하고 **특성...**을 클릭하십시오.
   b) 네비게이터에서 **통신**을 클릭하십시오.
   c) **선택...**을 클릭하고 SYSTEM.MQTT.TRANSMIT.QUEUE > **확인** > **확인**을 선택하십시오.
2. 텔레메트리 채널 MQTTExampleChannel을 작성하여 MQTT 클라이언트 유틸리티를 IBM MQ에 연결하고 MQTT 클라이언트 유틸리티를 시작하십시오.
   a) **MQ 탐색기**에서 Queue Managers*QmgrName* \Telemetry\Channels 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **텔레메트리 채널...**을 클릭하십시오.
   b) MQTTExampleChannel을 **채널 이름** 필드에 입력하고 > **다음** > **다음**을 클릭하십시오.
   c) 클라이언트 권한 패널에서 **고정 사용자 ID**를 MQTTExample를 발행하고 구독하려는 사용자 ID로 변경한 후 **다음**을 클릭하십시오.
   d) **클라이언트 유틸리티 시작**을 선택하고 **마침**을 클릭하십시오.
3. MQTT 클라이언트 유틸리티를 사용하여 MQTT Example 에 대한 구독을 작성하십시오.
   a) **연결**을 클릭하십시오.

      클라이언트 실행 기록은 Connected 이벤트를 기록합니다.
   b) MQTT Example을 **Subscription\Topic** 필드에 입력하고 **구독**을 입력하십시오.

   클라이언트 실행 기록은 Subscribed 이벤트를 기록합니다.
4. IBM MQ에서 MQTTExampleTopic을 작성하십시오.
   a) **MQ 탐색기**에서 Queue Managers*QmgrName*\Topics 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **토픽**을 클릭하십시오.
   b) MQTTExampleTopic을 **이름**으로 입력하고 **다음**을 클릭하십시오.
   c) MQTT Example을 **토픽 문자열**로 입력하고 **마침**을 클릭하십시오.
   d) 수신확인 창을 닫으려면 **확인**을 클릭하십시오.
5. mqm 그룹에 속하지 않은 사용자가 MQTTExample 토픽을 발행하고 구독하도록 하려면 다음을 수행하십시오.
   a) 사용자에게 MQTTExampleTopic 토픽을 발행 및 구독할 수 있는 권한을 부여하십시오.

      ```
      setmqaut -m qMgrName -t topic -n MQTTExampleTopic -p User ID -all +pub +sub
      ```
   b) 사용자에게 SYSTEM.MQTT.TRANSMIT.QUEUE에 메시지를 넣을 수 있는 권한을 부여하십시오.

      ```
      setmqaut -m qMgrName -t q -n SYSTEM.MQTT.TRANSMIT.QUEUE -p User ID -all +put
      ```
6. IBM MQ Explorer를 사용하여 Hello World! 를 MQTT Example 토픽에 발행하십시오.
   a) IBM MQ Explorer에서 Queue Managers*QmgrName*\Topics 폴더를 클릭하십시오.
   b) MQTTExampleTopic을 마우스 오른쪽 단추로 클릭하고 **발행물 테스트...**를 클릭하십시오.
   c) **메시지 데이터** 필드에 Hello World! 를 입력하고 > **메시지 발행** > MQTT 클라이언트 유틸리티 창으로 전환하십시오.

   클라이언트 실행 기록은 Received 이벤트를 기록합니다.

큐 관리자 알리어스 사용

기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정하지 않고 IBM MQ Explorer 를 사용하여 MQTT 클라이언트 유틸리티에 메시지를 발행하십시오.

이 태스크는 이전 태스크와 연결되는 것으로, 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정하지 않도록 하기 위해 큐 관리자 알리어스를 사용합니다.

시작하기 전에

108 페이지의 『텔레메트리(MQXR) 서비스가 아직 정의되지 않은 태스크 시작』 태스크 또는 109 페이지의 『실행 중인 텔레메트리(MQXR) 서비스가 있는 태스크 시작』 태스크를 완료하십시오.

이 태스크 정보

MQTT 클라이언트가 구독을 작성하면 IBM MQ가 `ClientIdentifier`를 리모트 큐 관리자 이름으로 사용하여 해당 응답을 보냅니다. 이 태스크에서는 `ClientIdentifier`, `MyClient`를 사용합니다.

`MyClient`라는 큐 관리자 알리어스나 전송 큐가 없으면 응답이 기본 전송 큐에 배치됩니다. 기본 전송 큐를 `SYSTEM.MQTT.TRANSMIT.QUEUE`로 설정하면 MQTT 클라이언트가 응답을 가져옵니다.

큐 관리자 알리어스를 사용하면 기본 전송 큐를 `SYSTEM.MQTT.TRANSMIT.QUEUE`로 설정하지 않아도 됩니다. 각 `ClientIdentifier`에 대해 큐 관리자 알리어스를 설정해야 합니다. 일반적으로 클라이언트가 지나치게 많아 실제로 큐 관리자 알리어스를 사용할 수 없습니다. 대개 `ClientIdentifier`는 예측이 불가능하므로 이러한 방식으로 텔레메트리를 구성하는 것은 불가능합니다.

그럼에도 불구하고 특정 환경에서는 기본 전송 큐를 `SYSTEM.MQTT.TRANSMIT.QUEUE`이외의 다른 설정으로 구성해야 합니다. 프로시저 의 단계에서는 기본 전송 큐를 `SYSTEM.MQTT.TRANSMIT.QUEUE`로 설정하는 대신 큐 관리자 알리어스를 구성합니다.

프로시저

1. `SYSTEM.MQTT.TRANSMIT.QUEUE`를 기본 전송 큐로 제거하십시오.
   a) `Queue Managers\`*`QmgrName`* `folder`를 마우스 오른쪽 단추로 클릭하고 **특성...**을 클릭하십시오.
   b) 네비게이터에서 **통신**을 클릭하십시오.
   c) **기본 전송 큐** 필드에서 `SYSTEM.MQTT.TRANSMIT.QUEUE`를 제거하고 **확인**을 클릭하십시오.
2. MQTT 클라이언트 유틸리티에서 더 이상 구독을 작성할 수 없는지 확인하십시오.
   a) **연결**을 클릭하십시오.

      클라이언트 실행 기록은 `Connected` 이벤트를 기록합니다.
   b) `MQTT Example`을 **Subscription\Topic** 필드에 입력하고 **구독**을 입력하십시오.

   클라이언트 실행 기록은 `Subscribe failed` 및 `Connection lost` 이벤트를 기록합니다.
3. `ClientIdentifier`, `MyClient`에 대한 큐 관리자 알리어스를 작성하십시오.
   a) `Queue Managers\`*`QmgrName`*`\Queues` 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **리모트 큐 정의**를 클릭하십시오.
   b) 정의 이름을 `MyClient`로 지정하고 **다음**을 클릭하십시오.
   c) **리모트 큐 관리자** 필드에 `MyClient`를 입력하십시오.
   d) **전송 큐** 필드에 `SYSTEM.MQTT.TRANSMIT.QUEUE`를 입력하고 **마침**을 클릭하십시오.
4. MQTT 클라이언트 유틸리티를 다시 연결하십시오.
   a) **클라이언트 ID**가 `MyClient`로 설정되어 있는지 확인하십시오.
   b) **연결**

   클라이언트 실행 기록은 `Connected` 이벤트를 기록합니다.
5. MQTT 클라이언트 유틸리티를 사용하여 `MQTT Example` 에 대한 구독을 작성하십시오.
   a) **연결**을 클릭하십시오.

      클라이언트 실행 기록은 `Connected` 이벤트를 기록합니다.
   b) `MQTT Example`을 **Subscription\Topic** 필드에 입력하고 **구독**을 입력하십시오.

   클라이언트 실행 기록은 `Subscribed` 이벤트를 기록합니다.
6. IBM MQ Explorer를 사용하여 `Hello World!` 를 `MQTT Example` 토픽에 발행하십시오.

a) IBM MQ Explorer에서 Queue Managers*QmgrName*\Topics 폴더를 클릭하십시오.

b) MQTTExampleTopic을 마우스 오른쪽 단추로 클릭하고 **발행물 테스트...**를 클릭하십시오.

c) **메시지 데이터** 필드에 Hello World! 를 입력하고 > **메시지 발행** > MQTT 클라이언트 유틸리티 창으로 전환하십시오.

클라이언트 실행 기록은 Received 이벤트를 기록합니다.

Windows Linux AIX IBM MQ Explorer 를 사용하여 MQTT 클라이언트에 메시지 송신

IBM MQ Explorer를 사용하여 IBM MQ 큐에 메시지를 넣어 MQTT 클라이언트 유틸리티에 메시지를 송신하십시오. 다음 태스크에서는 MQTT 클라이언트로 직접 메시지를 보내도록 리모트 큐 정의를 구성하는 방법을 보여줍니다.

시작하기 전에

108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』 태스크를 수행하십시오. MQTT 클라이언트 유틸리티를 연결된 상태로 두십시오.

이 태스크 정보

다음 태스크에서는 토픽을 발행하지 않고 큐를 사용하여 MQTT 클라이언트로 메시지를 보내는 방법을 보여줍니다. 클라이언트에서 구독을 작성하지 않습니다. 이 태스크의 112 페이지의 『2』 단계에서는 삭제된 이전 구독에 대해 설명합니다.

프로시저

1. MQTT 클라이언트 유틸리티의 연결을 끊었다가 다시 연결하여 기존의 구독을 모두 제거하십시오.

   기본값을 변경하지 않으면 MQTT 클라이언트 유틸리티가 정리 세션과 연결되므로 구독이 제거됩니다. 정리 세션을 참조하십시오.

   태스크를 더 쉽게 수행하려면 MQTT 클라이언트 유틸리티에서 작성한 생성된 ClientIdentifier 를 사용하지 말고 사용자 자신의 ClientIdentifier를 입력하십시오.

   a) **연결 끊기**를 클릭하여 텔레메트리 채널에서 MQTT 클라이언트 유틸리티의 연결을 끊으십시오.

   클라이언트 실행 기록은 Disconnected 이벤트를 기록합니다.

   b) **클라이언트 ID**를 MyClient로 변경하십시오.

   c) **연결**을 클릭하십시오.

   클라이언트 실행 기록은 Connected 이벤트를 기록합니다.

2. MQTT 클라이언트 유틸리티가 MQTTExampleTopic에 대한 발행물을 더 이상 수신하지 않는지 확인하십시오.

   a) IBM MQ Explorer에서 Queue Managers*QmgrName*\Topics 폴더를 클릭하십시오.

   b) MQTTExampleTopic을 마우스 오른쪽 단추로 클릭하고 **발행물 테스트...**를 클릭하십시오.

   c) **메시지 데이터** 필드에 Hello World! 를 입력하고 > **메시지 발행** > MQTT 클라이언트 유틸리티 창으로 전환하십시오.

   이 경우 **클라이언트 실행 기록**에 이벤트가 기록되지 않습니다.

3. 클라이언트에 대한 리모트 큐 정의를 작성하십시오.

   ClientIdentifier, MyClient를 리모트 큐 정의의 리모트 큐 관리자 이름으로 설정하십시오. 원하는 이름을 리모트 큐 이름으로 사용하십시오. 리모트 큐 이름이 MQTT 클라이언트에 토픽 이름으로 전달됩니다.

   a) Queue Managers*QmgrName*\Queues 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **리모트 큐 정의**를 클릭하십시오.

   b) 정의 이름을 MyClientRemoteQueue로 지정하고 **다음**을 클릭하십시오.

c) **리모트 큐** 필드에 MQTTExampleQueue를 입력하십시오.

d) **리모트 큐 관리자** 필드에 MyClient를 입력하십시오.

e) **전송 큐** 필드에 SYSTEM.MQTT.TRANSMIT.QUEUE를 입력하고 **마침**을 클릭하십시오.

4. 테스트 메시지를 MyClientRemoteQueue에 넣으십시오.

a) **MyClientRemoteQueue**를 마우스의 오른쪽 단추로 클릭하고 **테스트 메시지 넣기...**를 클릭하십시오.

b) 메시지 데이터 필드에 Hello queue!를 입력하고 **메시지 넣기** > **닫기**를 클릭하십시오.

클라이언트 실행 기록은 Received 이벤트를 기록합니다.

5. SYSTEM.MQTT.TRANSMIT.QUEUE를 기본 전송 큐로 제거하십시오.

a) Queue Managers*QmgrName* folder를 마우스 오른쪽 단추로 클릭하고 **특성...**을 클릭하십시오.

b) 네비게이터에서 **통신**을 클릭하십시오.

c) **기본 전송 큐** 필드에서 SYSTEM.MQTT.TRANSMIT.QUEUE를 제거하고 **확인**을 클릭하십시오.

6. 113 페이지의 『4』 단계를 다시 실행하십시오.

MyClientRemoteQueue는 전송 큐에 명시적으로 이름 지정된 리모트 큐 정의입니다. MyClient로 메시지를 보내기 위해 기본 전송 큐를 정의할 필요가 없습니다.

다음에 수행할 작업

기본 전송 큐가 더 이상 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정되지 않으면 MQTT 클라이언트 유틸리티는 큐 관리자 알리어스가 ClientIdentifier, MyClient에 대해 정의되지 않는 한 새 구독을 작성할 수 없습니다. 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE로 복원하십시오.

Windows Linux AIX 특정 MQTT v3 클라이언트로 메시지 발행

ClientIdentifier 를 토픽 이름으로 사용하고 IBM MQ 를 발행/구독 브로커로 사용하여 하나의 MQTT v3 클라이언트에서 다른 클라이언트로 메시지를 발행합니다.

시작하기 전에

108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』 태스크를 수행하십시오. MQTT 클라이언트 유틸리티를 연결된 상태로 두십시오.

이 태스크 정보

이 태스크에서는 다음 두 가지 사항에 대해 설명합니다.

1. 한 MQTT 클라이언트의 토픽을 구독하고 다른 MQTT 클라이언트에서 발행을 수신합니다.
2. ClientIdentifier를 토픽 문자열로 사용하여 "포인트-투-포인트" 구독을 설정합니다.

프로시저

1. MQTT 클라이언트 유틸리티의 연결을 끊었다가 다시 연결하여 기존의 구독을 모두 제거하십시오.

기본값을 변경하지 않으면 MQTT 클라이언트 유틸리티가 정리 세션과 연결되므로 구독이 제거됩니다. 정리 세션을 참조하십시오.

태스크를 더 쉽게 수행하려면 MQTT 클라이언트 유틸리티에서 작성한 생성된 ClientIdentifier 를 사용하지 말고 사용자 자신의 ClientIdentifier를 입력하십시오.

a) **연결 끊기**를 클릭하여 텔레메트리 채널에서 MQTT 클라이언트 유틸리티의 연결을 끊으십시오.

클라이언트 실행 기록은 Disconnected 이벤트를 기록합니다.

b) **클라이언트 ID**를 MyClient로 변경하십시오.

c) **연결**을 클릭하십시오.

클라이언트 실행 기록은 Connected 이벤트를 기록합니다.

2. 토픽, MyClient에 대한 구독을 작성하십시오.

   MyClient는 이 클라이언트의 ClientIdentifier입니다.

   a) MyClient를 **Subscription\Topic** 필드>에 입력하고 **구독**을 클릭하십시오.

   클라이언트 실행 기록은 Subscribed 이벤트를 기록합니다.
3. 다른 MQTT 클라이언트 유틸리티를 시작하십시오.
   a) Queue Managers*QmgrName*\Telemetry\channels 폴더를 여십시오.
   b) **일반 텍스트** 채널을 마우스의 오른쪽 단추로 클릭하고 **MQTT 클라이언트 유틸리티 실행...**을 클릭하십시오.
   c) **연결**을 클릭하십시오.

   클라이언트 실행 기록은 Connected 이벤트를 기록합니다.
4. Hello MyClient!를 MyClient 토픽에 발행하십시오.
   a) ClientIdentifier, MyClient로 실행 중인 MQTT 클라이언트 유틸리티에서 구독 토픽 MyClient를 복사하십시오.
   b) MyClient를 각 MQTT 유틸리티 인스턴스의 **Publication\Topic** 필드에 붙여넣으십시오.
   c) **Publication\message** 필드에 Hello MyClient! 를 입력하십시오.
   d) 두 인스턴스 모두에서 **발행**을 클릭하십시오.

결과

ClientIdentifier가 MyClient인 MQTT 클라이언트 유틸리티의 **클라이언트 실행 기록**에 두 개의 **Received** 이벤트와 하나의 **Published** 이벤트가 기록됩니다. 다른 MQTT 클라이언트 유틸리티 인스턴스는 한 개의 **발행됨** 이벤트를 기록합니다.

수신됨 이벤트가 하나만 표시되는 경우 다음이 원인일 수 있습니다.

1. 큐 관리자에 대한 기본 전송 큐가 SYSTEM.MQTT.TRANSMIT.QUEUE로 설정되었습니까?
2. 다른 실습에서 MyClient를 참조하는 리모트 큐 정의나 큐 관리자 알리어스를 작성했습니까? 구성 문제점이 있는 경우 큐 관리자 알리어스 또는 전송 큐와 같은 MyClient를 참조하는 자원을 모두 삭제하십시오. 클라이언트 유틸리티의 연결을 끊고 텔레메트리(MQXR) 서비스를 중지한 후에 다시 시작하십시오.

Windows Linux AIX MQTT 클라이언트에서 IBM MQ 애플리케이션으로 메시지 전송

IBM MQ 애플리케이션은 토픽을 구독함으로써 MQTT v3 클라이언트에서 메시지를 수신할 수 있습니다. MQTT 클라이언트는 텔레메트리 채널을 사용하여 IBM MQ에 연결하며 같은 토픽에 발행함으로써 IBM MQ 애플리케이션에 메시지를 송신합니다.

114 페이지의 『MQTT 클라이언트에서 IBM MQ 에 메시지 공개』 태스크를 수행하여 MQTT 클라이언트에서 IBM MQ에 정의된 구독으로 발행을 송신하는 방법을 배우십시오.

토픽이 클러스터되어 있거나 발행/구독 계층을 사용하여 분배되어 있을 경우 구독은 다른 큐 관리자에서 MQTT 클라이언트가 연결되어 있는 큐 관리자로 될 수 있습니다.

Windows Linux AIX MQTT 클라이언트에서 IBM MQ 에 메시지 공개

IBM MQ Explorer를 사용하여 토픽의 구독을 작성하고 MQTT 클라이언트 유틸리티를 사용하여 토픽에 발행하십시오.

시작하기 전에

108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』 태스크를 수행하십시오. MQTT 클라이언트 유틸리티를 연결된 상태로 두십시오.

이 태스크 정보

이 태스크는 MQTT 클라이언트로 메시지를 발행하는 것과 IBM MQ Explorer로 작성된 관리되지 않은 지속 가능 구독을 사용하여 발행을 수신하는 것을 보여줍니다.

프로시저

1. 토픽 문자열 `MQTT Example`에 대한 지속 가능 구독을 작성하십시오.

   IBM MQ Explorer를 사용하여 큐와 구독을 작성하려면 다음 단계를 수행하십시오.

   a) IBM MQ Explorer에서 `Queue Managers\`*`QmgrName`*`\Queues` 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **로컬 큐...**를 클릭하십시오.

   b) `MQTTExampleQueue`를 큐 이름으로 입력하고 **마침**을 클릭하십시오.

   c) IBM MQ Explorer에서 `Queue Managers\`*`QmgrName`*`\Subscriptions` 폴더를 마우스 오른쪽 단추로 클릭하고 **새로 작성** > **구독...**을 클릭하십시오.

   d) `MQTTExampleSubscription`을 큐 이름으로 입력하고 **다음**을 클릭하십시오.

   e) **선택...** > `MQTTExampleTopic` > **확인**을 클릭하십시오.

   108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』의 109 페이지의 『4』 단계에서 `MQTTExampleTopic` 토픽을 이미 작성했습니다.

   f) `MQTTExampleQueue`를 대상 이름으로 입력하고 **마침**을 클릭하십시오.
2. 선택적 단계로, `mqm` 권한이 없는 다른 사용자가 사용하도록 큐를 설정하십시오.

   `mqm`보다 낮은 권한을 가진 사용자에 대한 구성을 설정할 경우 `MQTTExampleQueue`에 `put` 및 `get` 권한을 부여해야 합니다. 토픽과 전송 큐에 대한 액세스는 108 페이지의 『IBM MQ Explorer 에서 MQTT 클라이언트 유틸리티에 메시지 발행』에서 구성되었습니다.

   a) 사용자에게 `MQTTExampleQueue` 큐에 대한 넣기 및 가져오기 권한을 부여하십시오.

   ```
   setmqaut -m qMgrName -t queue -n MQTTExampleQueue -p User ID -all +put +get
   ```
3. MQTT 클라이언트 유틸리티를 사용하여 토픽 `MQTT Example` 에 `Hello IBM MQ!` 를 발행하십시오.

   MQTT 클라이언트 유틸리티가 연결되어 있지 않은 경우 **일반 텍스트** 채널을 마우스의 오른쪽 단추로 클릭하고 **MQTT 클라이언트 유틸리티 실행...** > **연결**을 클릭하십시오.

   a) **Publication\Topic** 필드에 `MQTT Example`을 입력하십시오.

   b) **Publication\Message** 필드 > **Publish**에 `Hello IBM MQ!` 를 입력하십시오.
4. `Queue Managers\`*`QmgrName`*`\Queues` 폴더를 열고 `MQTTExampleQueue`를 찾으십시오.

   현재 큐 용량 필드는 1입니다.
5. `MQTTExampleQueue` > **메시지 찾아보기 ...** 를 마우스 오른쪽 단추로 클릭하십시오. 그리고 서적을 검토하십시오.

Windows Linux AIX MQTT 발행/구독 애플리케이션

MQTT 애플리케이션 쓰기에 토픽 기반 발행/구독을 사용하십시오.

MQTT 클라이언트가 연결되면 발행은 클라이언트와 서버 사이에서 오고 가게 됩니다. 발행은 클라이언트에서 정보가 발행되면 클라이언트에서 전송됩니다. 메시지가 클라이언트에서 작성한 구독과 일치하는 토픽에 발행되었을 경우에는 클라이언트에서 발행을 수신합니다.

IBM MQ 발행/구독 브로커는 MQTT 클라이언트에서 작성한 토픽과 구독을 관리합니다. MQTT 클라이언트에서 작성된 토픽은 IBM MQ 애플리케이션에서 작성된 토픽과 동일한 토픽 공간을 공유합니다.

MQTT 클라이언트 구독의 토픽 문자열과 일치하는 발행물은 리모트 큐 관리자 이름이 클라이언트의 `ClientIdentifier` 로 설정된 `SYSTEM.MQTT.TRANSMIT.QUEUE` 에 배치됩니다. 텔레메트리(MQXR) 서비스는 구독을 작성한 클라이언트에 발행물을 전달합니다. 클라이언트를 식별하기 위해 리모트 큐 관리자 이름으로 설정된 `ClientIdentifier`를 사용합니다.

일반적으로 SYSTEM.MQTT.TRANSMIT.QUEUE는 기본 전송 큐로 정의되어야 합니다. 기본 전송 큐를 사용하지 않도록 MQTT를 구성하는 것은 가능하지만 번거로운 작업입니다. 메시지를 MQTT 클라이언트에 송신하도록 분산 큐잉 구성을 참조하십시오.

MQTT 클라이언트는 지속 세션을 작성할 수 있습니다. 118 페이지의 『MQTT 상태 비저장 및 상태 저장 세션』의 내용을 참조하십시오. 지속 세션에 작성된 구독은 지속 가능합니다. 지속 세션과 함께 클라이언트에 도착하는 발행은 SYSTEM.MQTT.TRANSMIT.QUEUE에 저장되며, 다시 연결될 때 클라이언트로 전달됩니다.

MQTT 클라이언트는 보유된 발행물에 발행하고 구독할 수도 있습니다. 보유된 발행물 및 MQTT 클라이언트를 참조하십시오. 보유된 발행물 토픽에 대한 구독자는 토픽의 최근 발행을 수신합니다. 구독자는 보유된 발행물이 구독을 작성하거나 이전 세션에 다시 연결할 때 보유된 발행물을 수신합니다.

Windows Linux AIX 텔레메트리 애플리케이션

IBM MQ 또는 IBM Integration Bus 메시지 플로우를 사용하여 텔레메트리 애플리케이션을 씁니다.

JMS, MQI 또는 기타 IBM MQ 프로그래밍 인터페이스를 사용하여 IBM MQ에서 텔레메트리 애플리케이션을 프로그래밍하십시오.

텔레메트리(MQXR) 서비스가 MQTT v3 메시지와 IBM MQ 메시지를 상호 변환합니다. MQTT 클라이언트를 대신해 구독과 발행을 작성하며 발행을 MQTT 클라이언트에 전달합니다. 발행은 MQTT v3 메시지의 페이로드입니다. 페이로드는 메시지 헤더와 jms-bytes 형식의 바이트 배열을 포함합니다. 텔레메트리 서버는 MQTT v3 메시지와 IBM MQ 메시지 간의 헤더를 상호 맵핑합니다. 116 페이지의 『큐 관리자와 MQ Telemetry의 통합』의 내용을 참조하십시오.

Publication, MQInput및 JMSInput 노드를 사용하여 IBM Integration Bus 및 MQTT 클라이언트 간에 publication을 송신 및 수신하십시오.

메시지 플로우를 사용하여 텔레메트리를 HTTP를 사용하는 웹 사이트와 통합하고 IBM MQ 및 WebSphere Adapters를 사용하는 다른 애플리케이션과 통합할 수 있습니다.

Windows Linux AIX 큐 관리자와 MQ Telemetry의 통합

MQTT 클라이언트는 발행/구독 애플리케이션으로 IBM MQ와 통합되어 있습니다. IBM MQ의 토픽을 발행 또는 구독하고 새 토픽을 작성하며 기존 토픽을 사용할 수 있습니다. 자체를 포함한 MQTT 클라이언트 또는 해당 구독의 토픽에 발행하는 다른 IBM MQ 애플리케이션의 결과로 IBM MQ 에서 발행물을 수신합니다. 발행의 속성을 결정하는 규칙이 적용됩니다.

IBM MQ에서 제공되는 토픽, 발행, 구독 및 메시지와 연관된 많은 속성은 지원되지 않습니다. 116 페이지의 『MQTT 클라이언트에서 IBM MQ 발행/구독 브로커로』와 118 페이지의 『IBM MQ에서 MQTT 클라이언트로 메시지 전송』에서 발행의 속성을 설정하는 방법에 대해 설명합니다. 설정값은 발행물이 IBM MQ 발행/구독 브로커로 송신되는지 또는 이 브로커로부터 수신되는지에 따라 다릅니다.

IBM MQ에서 발행/구독 토픽은 관리 토픽 오브젝트와 연관됩니다. MQTT 클라이언트에서 작성되는 토픽도 이와 마찬가지입니다. MQTT 클라이언트가 발행의 토픽 문자열을 작성하면 IBM MQ 발행/구독 브로커는 이를 관리 토픽 오브젝트와 연관시킵니다. 브로커는 발행의 토픽 문자열을 제일 가까운 관리 토픽 오브젝트 상위에 맵핑합니다. 맵핑은 IBM MQ 애플리케이션의 경우와 동일합니다. 사용자가 작성한 토픽이 없을 경우 발행 토픽은 SYSTEM.BASE.TOPIC에 맵핑됩니다. 발행에 적용되는 토픽은 토픽 오브젝트에서 도출된 것입니다.

IBM MQ 애플리케이션 또는 관리자가 구독을 작성하는 경우 구독에 이름이 지정됩니다. 다음을 사용하여 구독 나열IBM MQ Explorer , 또는 다음을 사용하여**runmqsc** 또는 PCF 명령. 모든 MQTT 클라이언트 구독이 이름 지정됩니다. 다음과 같은 형식의 이름이 지정됩니다.*ClientIdentifier:Topic name*

MQTT 클라이언트에서 IBM MQ 발행/구독 브로커로

MQTT 클라이언트는 발행물을 IBM MQ로 송신했습니다. 텔레메트리(MQXR) 서비스는 IBM MQ 메시지로 발행물을 변환합니다. IBM MQ 메시지에는 다음 세 가지 부분이 포함됩니다.

1. MQMD
2. RFH2
3. 메시지

MQMD 특성은 117 페이지의 표 9에 기록되어 있지 않은 이상 각각 기본값으로 설정됩니다.

표 *9. MQMD* 특성의 설정

MQMD 필드	**유형**	**값**
Format	MQCHAR8	MQFMT_RF_HEADER_2
UserIdentifier	MQCHAR12	다음 중 하나로 설정하십시오. MqttClient.ClientIdentifier MqttConnectOptions.UserName 텔레메트리 채널용으로 IBM MQ 관리자가 설정한 사용자 ID
Priority	MQLONG	MQPRI_PRIORITY_AS_Q_DEF(IBM MQ의 경우 기본값이며, 기본값이 4인 JMS와 다릅니다.)
Persistence	MQLONG	QoS=0→MQPER_NOT_PERSISTENT QoS=1→MQPER_PERSISTENT QoS=2→MQPER_PERSISTENT

RFH2 헤더는 JMS 메시지의 유형을 정의하는 <msd> 폴더를 포함하지 않습니다. 텔레메트리(MQXR) 서비스는 IBM MQ 메시지를 기본 JMS 메시지로 작성합니다. 기본 JMS 메시지-유형은 jms-bytes 메시지입니다. 애플리케이션은 메시지 특성으로서의 추가 헤더 정보에 액세스할 수 있습니다. 메시지 특성을 참조하십시오.

RFH2 값은 117 페이지의 표 10에 표시된 것과 같이 설정됩니다. 형식 특성은 RFH2 고정된 헤더에 설정되며 다른 값은 RFH2 폴더 안에 설정됩니다.

표 *10. RFH2* 특성의 설정

RFH2 특성	**유형/폴더**	**헤더**
Format	MQCHAR8	MQFMT_NONE
ClientIdentifier	mqtt/clientId	MqttClient.ClientIdentifier를 1 - 23바이트 길이로 복사하십시오.
QoS	mqtt/qos	수신 MQTT 메시지에서 QoS 를 복사하십시오.
메시지 ID	mqtt/msgid	QoS 가 1 또는 2인 경우 수신 MQTT 메시지에서 메시지 ID 를 복사하십시오.
MQIsRetained	mqps/Ret	기존 MQTT 발행이 RETAIN 특성이 설정되어 전송되었으며 메시지가 보유된 발행물로 수신된 경우 설정하십시오.
MQTopicString	mqps/Top	MQTT 메시지 발행 대상이 된 토픽입니다.

MQTT 발행의 페이로드는 IBM MQ 메시지의 컨텐츠에 맵핑됩니다.

표 *11. MQTT* 발행의 페이로드가 *IBM MQ* 메시지 컨텐츠에 맵핑되는 방법

메시지 컨텐츠	**유형**	**IBM MQ 메시지의 컨텐츠**
Buffer	MQBYTE *n*	수신되는 MQTT 메시지에서 바이트를 복사하십시오. 길이는 0이 될 수도 있습니다.

IBM MQ에서 MQTT 클라이언트로 메시지 전송

클라이언트가 발행 토픽을 구독했습니다. IBM MQ 애플리케이션이 토픽에 발행되어 IBM MQ 발행/구독 브로커가 MQTT 구독자에게 발행을 송신합니다. 또는 IBM MQ 애플리케이션이 요청되지 않은 메시지를 MQTT 클라이언트로 직접 전송했습니다. 118 페이지의 표 12에서는 고정된 메시지 헤더를 MQTT 클라이언트로 전송된 메시지에 설정하는 방법에 대해 설명합니다. IBM MQ 메시지 헤더의 다른 모든 데이터 또는 다른 모든 헤더는 제거됩니다. IBM MQ 메시지의 메시지 데이터는 변경 없이 MQTT 메시지 안의 메시지 페이로드로 전송됩니다. MQTT 메시지는 텔레메트리(MQXR) 서비스에 의해 MQTT 클라이언트로 전송됩니다.

표 *12. MQTT* 클라이언트에 전송된 *IBM MQ* 메시지에서 고정 메시지 헤더를 설정하는 방법

MQTT 필드	**유형**	**값**
DUP	부울	QoS = 1 또는 2이며 이전 전송에서 메시지가 이 클라이언트로 전송되었고 이 메시지가 시간이 지난 뒤에도 수신확인되지 않았을 경우 설정하십시오.
QoS	int	IBM MQ 의 발행/구독 브로커에서 발신되는 발행물의 QoS 값이 설정되는 방법은 수신되는 발행물에 따라 다릅니다. QoS는 수신되는 발행물을 MQTT 클라이언트에서 송신했는지 또는 IBM MQ 애플리케이션에서 송신했는지에 따라 다릅니다. **MQTT** 수신되는 발행과 구독자에 의해 요청된 QoS의 QoS 값을 낮추십시오. **IBM MQ** 수신되는 발행에서 도출되는 QoS의 값을 낮추십시오. MQPER_NOT_PERSISTENT→QoS=0 MQPER_PERSISTENT→QoS=2 구독자에 의해 요청된 QoS의 QoS 값도 낮추십시오. 구독 없이 메시지가 클라이언트에 전송된 경우 QoS는 기본값인 2로 설정됩니다. 클라이언트는 다른 QoS로 DEFAULT.QoS를 구독하여 이 값을 변경할 수 있습니다.
RETAIN	부울	수신되는 발행에 보유된 특성이 설정되어 있을 경우 설정하십시오.

118 페이지의 표 13에서는 MQTT 클라이언트로 전송된 MQTT 메시지에 변수 메시지 헤더가 설정되는 방법에 대해 설명합니다.

표 *13. MQTT* 클라이언트로 전송된 *MQTT* 메시지에 *MQTT* 변수 헤더 특성을 설정하는 방법

MQTT 필드	**유형**	**값**
Topic name	문자열	메시지와 같이 발행된 토픽 문자열입니다.
Message ID	문자열	발행을 SYSTEM.MQTT.TRANSMIT.QUEUE에 넣었을 때 그 발행에 있는 MQMD.MsgId 특성의 마지막 2바이트입니다.
Payload	byte[]	수신되는 발행의 바이트 사본을 발행/구독 브로커에 전달하십시오. 길이는 0이 될 수도 있습니다.

Windows Linux AIX MQTT 상태 비저장 및 상태 저장 세션

MQTT는 큐 관리자를 사용하여 상태 저장 세션을 작성할 수 있습니다. 상태 저장 MQTT 클라이언트의 연결이 끊어지면 큐 관리자가 클라이언트에 의해 작성된 구독 및 인플라이트 메시지를 유지합니다. 클라이언트가 다시 연결되면 인플라이트 메시지를 해석합니다. 전달을 위해 큐에 대기된 모든 메시지를 보내고 연결이 끊어진 동안 구독에 대해 발행한 모든 메시지를 수신합니다.

MQTT 클라이언트가 텔레메트리 채널에 연결되면 새 세션을 시작하거나 기존 세션을 재개합니다. 새 세션에는 수신확인되지 않은 미해결 메시지, 구독 및 전달되기를 기다리는 발행이 포함되어 있지 않습니다. 클라이언트가 연결되면 정리 세션으로 시작하는지 또는 기존 세션을 재개하는지 여부를 지정합니다. 정리 세션을 참조하십시오.

클라이언트가 기존 세션을 재개하면 연결이 끊어지지 않았던 것처럼 계속 진행됩니다. 전달을 기다리는 발행은 클라이언트로 전송되고 커미트되지 않은 메시지 전송이 완료됩니다. 지속 세션의 클라이언트와 텔레메트리(MQXR) 서비스와의 연결이 끊어질 때 클라이언트가 작성한 모든 구독은 그대로 남아 있습니다. 클라이언트가 다시 연결되면 해당 클라이언트에게 구독에 대한 발행이 전송됩니다. 이전 세션을 재개하지 않고 다시 연결하면 텔레메트리(MQXR) 서비스가 발행물을 제거합니다.

큐 관리자가 SYSTEM.MQTT.PERSISTENT.STATE 큐에 세션 상태 정보를 저장합니다.

IBM MQ 관리자는 세션의 연결을 끊고 세션을 제거할 수 있습니다.

Windows Linux AIX MQTT 클라이언트가 연결되지 않았을 때

클라이언트가 연결되지 않았을 때 큐 관리자는 이를 대신해 발행을 계속 수신할 수 있습니다. 발행은 클라이언트가 다시 연결할 때 여기로 전달됩니다. 클라이언트는 예상치 못하게 연결이 끊어질 경우 큐 관리자가 자신을 대신해 발행하는 "이상 종료 시 메시지"를 작성할 수 있습니다.

클라이언트가 예상치 못하게 연결이 끊겼을 때 알림을 받고 싶으면 이상 종료 시 메시지 발행을 등록할 수 있습니다. 이상 종료 시 메시지 발행을 참조하십시오. 클라이언트에 대한 연결이 클라이언트의 요청 없이 중단되었음을 감지한 경우, 이는 텔레메트리(MQXR) 서비스에 의해 전송됩니다.

클라이언트는 언제든지 보유된 발행물을 발행할 수 있습니다. 보유된 발행물 및 MQTT 클라이언트를 참조하십시오. 토픽에 대한 새 구독은 토픽과 연관된 어떤 보유된 발행물도 송신해 달라고 요청할 수 있습니다. 이상 종료 시 메시지를 보유된 발행물로서 작성할 경우 이를 클라이언트 상태를 모니터하는 데 사용할 수 있습니다.

예를 들면, 한 클라이언트는 연결할 때 자신의 가용성을 알리며 보유된 발행물을 발행합니다. 동시에, 비가용성을 알리는 보유된 이상 종료 시 메시지 발행을 작성합니다. 또한 계획된 연결 종료를 하기 전에 보유된 발행물로서 자신의 비가용성을 발행합니다. 클라이언트가 사용 가능한지 알아보려면 이 보유된 발행의 토픽을 구독합니다. 사용자는 항상 세 가지 발행 중 하나를 수신합니다.

클라이언트가 연결이 끊어져 있을 때 발행된 메시지를 수신하고자 하는 경우 클라이언트를 이전 세션에 다시 연결하십시오. 118 페이지의 『MQTT 상태 비저장 및 상태 저장 세션』의 내용을 참조하십시오. 그 구독은 삭제될 때까지 또는 클라이언트가 정리 세션을 작성할 때까지 활성 상태입니다.

Windows Linux AIX MQTT 클라이언트와 IBM MQ 애플리케이션 사이의 느슨한 결합

MQTT 클라이언트와 IBM MQ 애플리케이션 사이의 발행 플로우는 느슨히 결합되어 있습니다. 발행은 MQTT 클라이언트 또는 IBM MQ 애플리케이션에서 설정된 순서 없이 생성될 수 있습니다. 발행자와 구독자는 느슨히 결합되어 있습니다. 이들은 발행과 구독을 통해 서로 상호작용합니다. 또한 IBM MQ 애플리케이션에서 MQTT 클라이언트로 직접 메시지를 보낼 수도 있습니다.

MQTT 클라이언트와 IBM MQ 애플리케이션은 다음과 같은 두 가지 측면에서 느슨히 결합되어 있습니다.

1. 발행자와 구독자는 발행, 구독과 토픽의 연관을 통해 느슨히 결합되어 있습니다. 발행자와 구독자는 보통 다른 발행 또는 구독 소스의 주소나 ID를 알고 있습니다.
2. MQTT 클라이언트는 발행을 발행, 구독, 수신하며 분리된 스레드에서 전달 수신확인을 처리합니다.

MQTT 클라이언트 애플리케이션은 발행이 전달될 때까지 대기하지 않습니다. 애플리케이션은 MQTT 클라이언트로 메시지를 전달하고 그 후 애플리케이션은 자신의 스레드에서 계속됩니다. 발행물의 전달과 애플리케이션을 동기화하기 위해 전달 토큰이 사용됩니다. 전달 토큰을 참조하십시오.

MQTT 클라이언트에 메시지를 전달한 후 애플리케이션은 전달 토큰을 기다릴지 선택할 수 있습니다. 기다리는 대신 클라이언트는 IBM MQ에 발행이 전달되었을 때 호출되는 콜백 메소드를 제공할 수 있습니다. 이는 또한 전달 토큰을 무시할 수 있습니다.

메시지의 서비스 품질에 따라 전달 토큰은 콜백 메소드로 즉시 리턴되거나 상당한 시간이 지난 뒤 리턴될 수 있습니다. 전달 토큰은 클라이언트 연결이 끊어진 후 다시 연결되었을 때 리턴될 수도 있습니다. 서비스 품질이 전송

후 삭제인 경우 전달 토큰은 즉시 리턴됩니다. 다른 두 경우 전달 토큰은 클라이언트가 발행이 구독자로 송신되었다는 수신확인을 수신했을 때만 리턴됩니다.

클라이언트 구독의 결과로 MQTT 클라이언트로 송신된 발행은 `messageArrived` 콜백 메소드로 전달됩니다. `messageArrived`는 기본 애플리케이션과 다른 스레드에서 실행됩니다.

MQTT 클라이언트로 직접 메시지 송신

둘 중 한 가지 방법으로 특정 MQTT 클라이언트에 메시지를 송신할 수 있습니다.

1. IBM MQ 애플리케이션은 구독 없이 MQTT 클라이언트에 직접 메시지를 보낼 수 있습니다. 클라이언트에 직접 메시지 보내기를 참조하십시오.
2. 또 다른 방법은 `ClientIdentifier` 이름 지정 규칙을 사용하는 것입니다. 모든 MQTT 구독자가 자신의 고유한 `ClientIdentifier`를 토픽으로 사용하여 구독을 작성하도록 하십시오. *ClientIdentifier*에 발행하십시오. 발행은 토픽 *ClientIdentifier*를 구독한 클라이언트에게 송신됩니다. 이 기술을 사용하면 발행을 특정 MQTT 구독자에게 송신할 수 있습니다.

Windows Linux AIX MQ Telemetry 보안

텔레메트리 디바이스는 대부분 휴대용이며 잘 제어되지 않은 환경에서 사용하게 될 경우가 많으므로 디바이스를 보호하는 것은 중요합니다. VPN을 사용하여 MQTT 디바이스에서 텔레메트리(MQXR) 서비스로의 연결을 보호할 수 있습니다. MQ Telemetry는 TLS와 JAAS라는 다른 두 가지 보안 메커니즘을 제공합니다.

TLS는 기본적으로 디바이스 및 텔레메트리 채널 간의 통신을 암호화하며 디바이스가 올바른 서버에 연결 중인지 인증하기 위해 사용됩니다. TLS를 사용하여 텔레메트리 채널 인증을 참조하십시오. TLS를 사용하여 클라이언트 디바이스가 서버에 연결할 수 있는지 확인할 수도 있습니다. TLS를 사용하여 MQTT 클라이언트 인증을 참조하십시오.

JAAS는 기본적으로 디바이스의 사용자가 서버 애플리케이션을 사용할 수 있는지 확인하는 데 사용됩니다. 비밀번호를 사용하여 MQTT 클라이언트 인증을 참조하십시오. JAAS는 싱글 사인온 디렉토리를 사용하는 비밀번호를 확인하기 위해 LDAP와 함께 사용할 수 있습니다.

TLS 및 JAAS는 두 인수 인증을 제공하기 위해 결합해 사용할 수 있습니다. TLS에서 사용하는 암호를 FIPS 표준을 만족시키는 암호로 제한할 수 있습니다.

최소 수만 명의 사용자가 있을 경우 개별 보안 프로파일을 제공하는 것은 실용적이지 않을 수 있습니다. IBM MQ 오브젝트에 액세스하려는 개별 사용자를 인증하는 데 프로파일을 사용하는 것 또한 마찬가지입니다. 대신 토픽에 대한 발행 및 구독 인증과, 클라이언트로의 발행물 송신을 위해 사용자를 클래스로 그룹 지으십시오.

클라이언트를 공용 클라이언트 사용자 ID로 맵핑하도록 각 텔레메트리 채널을 구성하십시오. 특정 채널에서 연결하는 모든 클라이언트에 대해 공용 사용자 ID를 사용하십시오. MQTT 클라이언트 ID 및 권한을 참조하십시오.

사용자 그룹에 권한을 부여하는 것이 각 개인 인증에 문제를 일으키지는 않습니다. 각 개별 사용자는 클라이언트와 서버에서 자신의 `Username`과 `Password`로 인증을 받은 후 공용 사용자 ID를 사용하는 서버에서 권한 부여를 받을 수 있습니다.

Windows Linux AIX MQ Telemetry 다국어 지원

MQTT v3 프로토콜의 메시지 페이로드는 바이트 배열로 인코드됩니다. 일반적으로 텍스트를 처리하는 애플리케이션은 UTF-8로 메시지 페이로드를 작성합니다. 텔레메트리 채널은 메시지 페이로드를 UTF-8로 설명하지만 어떤 코드 페이지 변환도 수행하지 않습니다. 발행 토픽 문자열은 UTF-8이어야 합니다.

영문자 데이터를 올바른 코드 페이지로, 숫자 데이터를 올바른 숫자 인코딩으로 변환하는 것은 애플리케이션의 역할입니다.

MQTT Java 클라이언트에는 간편한 `MqttMessage.toString` 메소드가 있습니다. 이 메소드는 메시지 페이로드를 로컬 플랫폼의 기본 문자 세트(일반적으로 UTF-8)로 인코드된 것으로 취급합니다. 이는 페이로드를 Java String으로 변환합니다. Java에는 로컬 플랫폼의 기본 문자 세트를 사용하여 문자열을 인코드된 바이트 배열로 변환하는 String 메소드인 `getBytes`가 있습니다. 같은 기본 문자 세트를 가진 플랫폼 사이에서 메시지 페이로드로 텍스트를 교환하는 두 MQTT Java 프로그램은 UTF-8로 이를 쉽고 효율적으로 수행합니다.

한 플랫폼의 기본 문자 세트가 UTF-8이 아닐 경우에 애플리케이션은 메시지를 교환하는 데 변환을 설정해야 합니다. 예를 들면, 발행자가 getBytes("UTF8") 메소드를 사용하여 문자열에서 UTF-8로의 변환을 지정하는 경우가 있습니다. 메시지의 텍스트를 수신하기 위해 구독자는 메시지가 UTF-8 문자 세트로 인코드되어 있다고 추정합니다.

텔레메트리(MQXR) 서비스는 MQTT 클라이언트 메시지로부터 수신되는 모든 발행물의 인코딩이 UTF-8임을 설명합니다. MQMD.CodedCharSetId 를 UTF-8로, RFH2.CodedCharSetId 를 MQCCSI_INHERIT 로 116 페이지의 『큐 관리자와 MQ Telemetry의 통합』 참조. 발행의 형식은 MQFMT_NONE으로 설정되어 채널이나 MQGET에 의해 변환이 수행되지 않도록 합니다.

Windows Linux AIX MQ Telemetry의 성능 및 확장성

다수의 클라이언트를 관리하고 MQ Telemetry의 확장성을 향상시킬 때 다음과 같은 요인을 고려하십시오.

용량 계획

MQ Telemetry의 성능 보고서에 대한 정보를 보려면 MQ 성능 문서를 참조하십시오.

연결

연결과 관련된 비용은 다음을 포함하고 있습니다.

- 프로세서 사용 및 시간의 관점에서 본 연결 자체를 설정하는 비용.
- 네트워크 비용.
- 연결을 열어놓고 사용하지 않는 동안 사용되는 메모리.

클라이언트가 연결된 채일 때 발생하는 추가적인 부하가 있습니다. 연결이 열려 있는 경우 TCP/IP에는 플로우가 계속 발생하며 MQTT 메시지는 연결이 계속 있는지 확인하기 위해 네트워크를 사용합니다. 또한 서버에서 열려 있는 채인 각 클라이언트 연결에 대해 메모리가 사용됩니다.

1분에 하나 이상 메시지를 송신할 경우 새 연결을 시작하는 비용을 피하기 위해 연결을 계속 열어두십시오. 10 - 15분 동안 하나 이하의 메시지를 송신할 경우 연결을 열어두는 비용을 피하기 위해 연결을 끊는 것을 고려하십시오. TLS 연결은 설정하는 데 비용이 많이 소요되기 때문에 열어둔 채로 두는 편이 좋을 수 있습니다.

추가적으로 클라이언트의 기능을 고려하십시오. 클라이언트에 저장 후 전달 기능이 있는 경우 메시지를 배치해 두고 배치를 송신하는 사이에 연결을 끊을 수 있습니다. 그러나 클라이언트의 연결이 끊긴 경우에는 클라이언트가 서버에서 메시지를 수신할 수 없습니다. 따라서 애플리케이션의 목적이 의사결정에 영향을 미칩니다.

파일 전송과 같이 시스템에 많은 메시지를 송신하는 클라이언트가 있을 경우 메시지마다 서버 응답을 기다리지 마십시오. 대신 모든 메시지를 송신하고 마지막에 메시지가 모두 수신되었는지 확인하십시오. 또는 QoS(Quality of Service)를 사용하십시오.

중요하지 않은 메시지는 QoS 0을 사용하여 전달하고 중요한 메시지는 QoS 2를 사용하여 전달하는 것처럼 메시지마다 QoS를 달리 할 수 있습니다. 메시지 처리량은 QoS가 2일 때보다 QoS가 0일 때 약 두 배 더 높을 수 있습니다.

이름 지정 규칙

다수의 클라이언트를 대상으로 애플리케이션을 설계할 경우 효율적인 이름 지정 규칙을 구현하십시오. 각 클라이언트를 올바르게 ClientIdentifier에 맵핑하려면 ClientIdentifier를 의미있게 작성하십시오. 좋은 이름 지정 규칙은 어느 클라이언트가 실행 중인지 관리자가 쉽게 알 수 있게 해 줍니다. 이름 지정 규칙은 관리자가 IBM MQ 탐색기에서 긴 클라이언트 목록을 필터링하고 문제점을 판별하는 데 도움이 됩니다. 클라이언트 ID를 참조하십시오.

처리량

토픽 이름의 길이는 네트워크를 흘러가는 바이트 수에 영향을 줍니다. 발행하거나 구독할 때 메시지의 바이트 수는 중요할 수 있습니다. 그러므로 토픽 이름의 문자 개수를 제한하십시오. MQTT 클라이언트가 토픽을 구독할 때 IBM MQ는 여기에 다음과 같은 형식의 이름을 부여합니다.

```
ClientIdentifier: TopicName
```

MQTT 클라이언트의 모든 구독을 보려면 IBM MQ MQSC **DISPLAY** 명령을 사용할 수 있습니다.

```
DISPLAY SUB(' ClientID1:*')
```

IBM MQ에 MQTT 클라이언트에서 사용하기 위한 자원 정의

MQTT 클라이언트가 IBM MQ에 리모트 큐 관리자를 연결합니다. IBM MQ 애플리케이션이 MQTT 클라이언트에 메시지를 송신하는 두 가지 기본 방법이 있습니다. 기본 전송 큐를 SYSTEM.MQTT.TRANSMIT.QUEUE 로 설정하거나 큐 관리자 알리어스를 사용하십시오. 다수의 MQTT 클라이언트가 있을 경우 큐 관리자의 기본 전송 큐를 정의하십시오. 기본 전송 큐 설정을 사용하면 관리 작업이 간소화됩니다. MQTT 클라이언트에 메시지를 송신하도록 분산 큐잉 구성을 참조하십시오.

구독하지 않음으로서 오는 확장성 향상

MQTT V3 클라이언트가 토픽을 구독하는 경우, 구독은 IBM MQ의 텔레메트리(MQXR) 서비스에 의해 작성됩니다. 구독은 클라이언트가 구독한 발행을 SYSTEM.MQTT.TRANSMIT.QUEUE로 라우팅합니다. 각 발행물의 전송 헤더에 있는 리모트 큐 관리자 이름은 구독을 작성한 MQTT 클라이언트의 ClientIdentifier 로 설정됩니다. 각각 자신의 구독을 작성하고 있는 다수의 클라이언트가 있을 경우 IBM MQ 발행/구독 클러스터 또는 계층 전반에 많은 프록시 구독이 유지되는 결과가 발생합니다. 발행/구독을 사용하는 대신 포인트-투-포인트 기반 솔루션 사용에 대한 정보는 클라이언트에 직접 메시지 보내기를 참조하십시오.

다수의 클라이언트 관리

다수의 동시에 연결된 클라이언트를 지원하려면 JVM 매개변수 **-Xms** 및 **-Xmx**를 설정하여 텔레메트리(MQXR) 서비스에 사용할 수 있는 메모리를 늘리십시오. 다음 단계를 수행하십시오.

1. 텔레메트리 서비스 구성 디렉토리에서 java.properties 파일을 찾으십시오. Windows의 텔레메트리(MQXR) 서비스 구성 디렉토리 또는 Linux의 텔레메트리 서비스 구성 디렉토리를 참조하십시오.
2. 파일의 지시사항을 따르십시오. 50,000개의 동시 연결된 클라이언트를 유지하는 데는 1GB 힙 정도면 충분합니다.

   ```
   # Heap sizing options - uncomment the following lines to set the heap to 1G
   #-Xmx1024m
   #-Xms1024m
   ```

3. java.properties 파일에 텔레메트리(MQXR) 서비스를 실행 중인 JVM에 전달할 다른 명령행 인수를 추가하십시오. 텔레메트리(MQXR) 서비스에 JVM 매개변수 전달을 참조하십시오.

Linux에서 열린 파일 디스크립터의 수를 늘리려면 /etc/security/limits.conf/에 다음 행을 추가하고 다시 로그인하십시오.

```
@mqm soft nofile 65000
@mqm hard nofile 65000
```

각 소켓은 하나의 파일 디스크립터를 필요로 합니다. 텔레메트리 서비스는 몇몇 추가 파일 디스크립터를 필요로 하며 따라서 이 숫자는 요구되는 열린 소켓의 개수보다 커야 합니다.

큐 관리자는 각 지속 불가능 구독에 오브젝트 핸들을 사용합니다. 많은 활성 지속 불가능 구독을 지원하려면 큐 관리자에서 활성 핸들의 최대 개수를 증가시키십시오. 예:

```
echo ALTER QMGR MAXHANDS(999999999) | runmqsc qMgrName
```

그림 *48. Windows*에서 최대 핸들 수 대체

```
echo "ALTER QMGR MAXHANDS(999999999)" | runmqsc qMgrName
```

그림 *49. Linux*에서 최대 핸들 수 대체

기타 고려사항

시스템 요구사항을 계획할 때 다시 시작하는 데 드는 시간의 길이를 고려하십시오. 계획된 중단 시간은 처리될 때까지 큐에 들어가 대기하는 메시지의 개수에 영향을 줄 수 있습니다. 메시지가 허용 가능한 범위의 시간 안에 처리 완료될 수 있도록 시스템을 구성하십시오. 디스크 스토리지, 메모리 및 처리 능력을 검토하십시오. 몇몇 클라이언트 애플리케이션의 경우 클라이언트가 다시 연결할 때 메시지를 버릴 수도 있습니다. 메시지를 제거하려면 클라이언트 연결 매개변수의 `CleanSession`을 설정하십시오. 정리 세션을 참조하십시오. 또는 MQTT 클라이언트에서 최상의 서비스 품질 (QoS) `0`을 사용하여 발행 및 구독하십시오. 서비스 품질을 참조하십시오. IBM MQ에서 메시지를 보낼 때 비지속 메시지를 사용하십시오. 이 서비스 품질의 메시지는 시스템이나 연결이 다시 시작할 때 복구되지 않습니다.

Windows Linux AIX MQ Telemetry에서 지원되는 디바이스

MQTT 클라이언트는 센서 및 작동기부터 휴대용 디바이스 및 차량 시스템까지 다양한 범위의 디바이스에서 실행할 수 있습니다.

MQTT 클라이언트는 크기가 작으며, 적은 메모리를 갖고 처리 능력이 낮은 디바이스에서 실행됩니다. MQTT protocol은 안정적이며 헤더가 작고, 이는 대역폭이 낮고 사용 비용이 높으며 단속적인 네트워크에 적합합니다.

MQ Telemetry는 MQTT 클라이언트 애플리케이션을 통해 텔레메트리 디바이스와 통신합니다. 이러한 애플리케이션은 모두 MQTT v3 프로토콜을 구현하는 다음 자원을 사용합니다.

- 다음 클라이언트 라이브러리:
  - *MQTT client for Java*, 이는 예를 들어 Android, OS X, Linux 또는 Windows 디바이스에 대한 고유 애플리케이션을 빌드하는 데 사용됩니다. 이 클라이언트 라이브러리를 사용하는 애플리케이션은 가장 작은 CLDC(Connected Limited Device Configuration)/MIDP(Mobile Information Device Profile)에서부터 CDC(Connected Device Configuration)/Foundation, J2SE(Java Platform, Standard Edition), 그리고 J2EE(Java Platform, Enterprise Edition)까지 Java의 모든 변형 형태에서 실행할 수 있습니다. IBM jclRM 사용자 정의된 클래스 라이브러리 또한 지원됩니다. Java ME 플랫폼은 작동 장치, 센서, 휴대전화 및 기타 임베드된 디바이스와 같은 소형 디바이스에 보통 사용됩니다. Java SE 플랫폼은 데스크탑 컴퓨터 및 서버와 같은 더 고사양의 임베드된 디바이스에 보통 설치됩니다.
  - *MQTT client for C*, 이는 예를 들어 iOS, OS X, Linux 또는 Windows 디바이스에 대한 고유 애플리케이션을 빌드하는 데 사용됩니다. 이 클라이언트 라이브러리는 Windows 및 Linux 시스템에 대해 사전 빌드된 고유 클라이언트와 함께 C 참조 구현을 제공합니다. C 참조 구현은 MQTT가 다양한 디바이스와 플랫폼으로 포트될 수 있도록 해 줍니다. Windows 7을 포함한 Intel의 일부 Windows 시스템, RedHat, Ubuntu 및 ARM 플랫폼(예: Eurotech Viper)의 일부 Linux 시스템은 C 클라이언트를 실행하는 Linux 버전을 구현하지만 IBM에서는 이러한 플랫폼에 대한 서비스 지원을 제공하지 않습니다. 사용자의 IBM 지원 센터를 호출하려면 지원되는 플랫폼에서 클라이언트로 이 문제점을 다시 발생시켜야 합니다.
  - *MQTT client for Java*, 이는 브라우저 기반 웹 애플리케이션을 빌드하는 데 사용됩니다.

MQTT 클라이언트 라이브러리는 Eclipse Paho 및 MQTT.org에서 무료로 제공됩니다. IBM MQ Telemetry Transport 샘플 프로그램을 참조하십시오.

IBM MQ의 보안

IBM MQ에는 권한 서비스 인터페이스, 사용자 작성 또는 써드파티, 채널 엑시트, TLS(Transport Layer Security)를 사용하는 채널 보안, 채널 인증 레코드 및 메시지 보안 등과 같이 보안을 제공하는 여러 가지 방법이 있습니다.

권한 서비스 인터페이스

MQI 호출에 대한 인증, 명령 및 오브젝트에 대한 액세스는 기본적으로 사용 설정된 **오브젝트 권한 관리자**(OAM)에 의해 제공됩니다. IBM MQ 엔티티에 대한 액세스는 IBM MQ 사용자 그룹 및 OAM을 통해 제어됩니다. 관리자는 명령행 인터페이스를 사용하여 필요에 따라 권한을 부여하거나 폐기할 수 있습니다.

권한 서비스 컴포넌트 작성에 대한 자세한 정보는 AIX, Linux, and Windows 시스템에서 보안 설정을 참조하십시오.

사용자 작성 또는 써드파티 채널 엑시트

채널은 사용자 작성 또는 써드파티 채널 엑시트를 사용할 수 있습니다. 자세한 정보는 메시지 채널의 채널-엑시트 프로그램을 참조하십시오.

TLS를 사용하는 채널 보안

TLS(Transport Layer Security) 프로토콜은 도청, 도용 및 위장에 대한 보호와 산업 표준 채널 보안을 제공합니다.

TLS는 메시지 기밀성 및 무결성을 제공하는 공용 키 및 대칭 기술과 상호 인증을 사용합니다.

TLS에 대한 자세한 정보를 포함하여 IBM MQ의 보안에 대한 포괄적인 검토는 보안을 참조하십시오. 이 절에 설명된 명령에 대한 포인터를 포함하여 TLS의 개요는 암호화 보안 프로토콜: TLS를 참조하십시오.

채널 인증 레코드

채널 인증 레코드를 사용하여 채널 레벨에서 시스템 연결에 부여된 액세스 권한에 대해 정확한 제어를 실행합니다. 자세한 정보는 채널 인증 레코드를 참조하십시오.

메시지 보안

IBM MQ의 별도로 설치되고 라이센스가 부여된 구성요소인 Advanced Message Security를 사용하여 IBM MQ를 사용하여 보내고 받은 메시지에 암호화 보호를 제공하십시오. Advanced Message Security의 내용을 참조하십시오.

관련 태스크

보안 설정

보안 요구사항 계획

IBM MQ.NET 관리 대상 클라이언트 TLS 지원

IBM MQ.NET 완전 관리 클라이언트는 Microsoft.NET SSLStreams 킷을 기반으로 하는 TLS(Transport Layer Security) 지원을 제공합니다. 이는 IBM Global Security Kit (GSKit)를 기반으로 하는 다른 IBM MQ 클라이언트와 다릅니다.

관리 모드 또는 비관리 모드로 실행되는 IBM MQ.NET 애플리케이션을 개발할 수 있습니다.

- 관리 모드에서 .NET 애플리케이션은 C MQI 호출 등의 교차 플랫폼 호출을 수행하지 않고 .NET CLR(Common Language Runtime) 내에서 작동합니다.
- 비관리 모드에서 C MQI는 기본 MQI 조작을 위해 호출됩니다. 기본적으로 비관리 모드 인터페이스는 C MQI 외에 .NET 랩퍼 클래스로 구성됩니다.

관리 IBM MQ.NET 클라이언트는 Microsoft.NET Framework 라이브러리를 사용하여 TLS 보안 소켓 프로토콜을 구현합니다. Microsoft 의 System.NET.Security.SSLStream 클래스는 IBM MQ.NET에서 보안 (TLS) 을 구현하는 데 사용됩니다.

비관리 IBM MQ.NET 클라이언트 모드는 C MQI (및 GSKit) 를 기반으로 하는 TLS 기능을 이미 지원합니다. 즉, TLS 조작은 C MQI에서 처리합니다. 이 경우 GSKit 는 TLS 보안 소켓 프로토콜을 구현합니다.

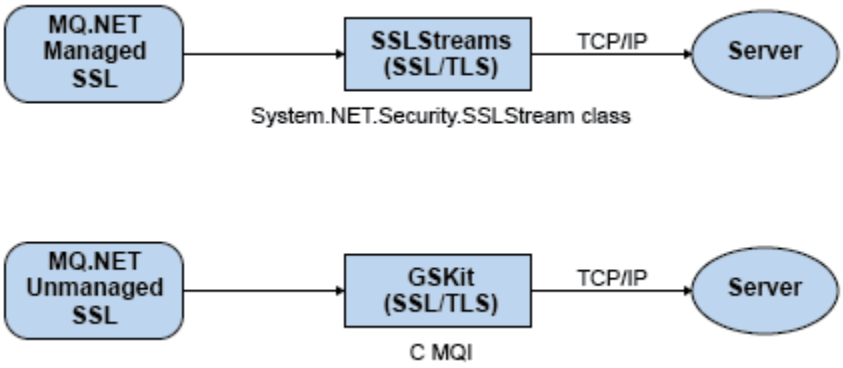


그림 *50. IBM MQ.NET* 관리 및 비관리 *TLS* 비교

다음 표는 관리와 비관리 구현 사이의 차이점을 요약합니다.

표 *14.* 관리와 비관리 구현 사이의 차이점

모드	프로토콜	구현	주석
IBM MQ.NET 관리 SSL	TLS	System.NET.Security.SSLStream 클래스 SSLStream 클래스는 연결된 TCP 소켓을 통한 스트림으로 작동	TLS 1.0 TLS 1.2(Microsoft.NET Framework v4.5 전용)
IBM MQ.NET 비관리 SSL	TLS	GSKit 및 C-MQI	TLS 보안 소켓 프로토콜

관련 개념

.NET용 SSL(Secure Sockets Layer) 및 TLS(Transport Layer Security) 지원

IBM MQ MQI clients

IBM MQ MQI client는 큐 관리자가 실행되지 않는 시스템에 설치할 수 있는 IBM MQ 제품의 컴포넌트입니다.

IBM MQ MQI 클라이언트는 시스템에서 실행 중인 애플리케이션이 다른 시스템에서 실행 중인 큐 관리자에 대한 MQI 호출을 실행할 수 있도록 허용하는 컴포넌트입니다. 호출의 출력은 클라이언트로 되돌아오고 이는 애플리케이션으로 다시 전달됩니다.

IBM MQ MQI client를 사용하여 클라이언트와 동일한 시스템에서 실행되는 애플리케이션을 다른 시스템에서 실행 중인 큐 관리자에 연결할 수 있습니다. 애플리케이션은 해당 큐 관리자에게 MQI 호출을 발행할 수 있습니다. 이러한 애플리케이션을 IBM MQ MQI client 애플리케이션이라 하며 큐 관리자는 서버 큐 관리자라 합니다.

IBM MQ 서버는 하나 이상의 클라이언트에게 큐잉 서비스를 제공하는 큐 관리자입니다. 모든 IBM MQ 오브젝트(예: 큐)는 큐 관리자 시스템(IBM MQ 서버 시스템)에만 존재하며 클라이언트에는 존재하지 않습니다. IBM MQ 서버는 로컬 IBM MQ 애플리케이션도 지원할 수 있습니다.

IBM MQ 서버와 일반 큐 관리자 간의 차이점은 서버가 각 클라이언트에 대한 전용 통신 링크를 가진다는 점입니다. 클라이언트 및 서버의 채널 작성에 대한 자세한 정보는 분산 큐잉 구성을 참조하십시오.

IBM MQ MQI client 애플리케이션과 서버 큐 관리자는 *MQI* 채널을 사용하여 서로 통신합니다. MQI 채널은 클라이언트 애플리케이션이 **MQCONN** 또는 **MQCONNX** 호출을 발행하여 큐 관리자에 연결할 때 시작하고 클라이언트 애플리케이션이 **MQDISC** 호출을 발행하여 큐 관리자와의 연결을 끊을 때 종료됩니다. MQI 호출의 입력 매개변수는 MQI 채널에서 한 방향으로 플로우하고 출력 매개변수는 반대 방향으로 플로우합니다.

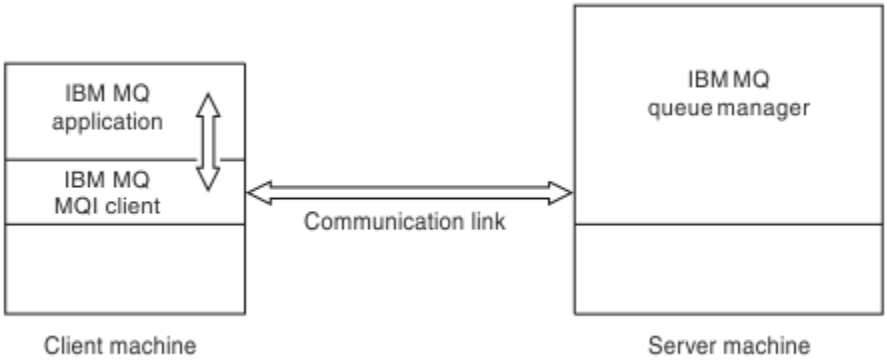


그림 *51.* 클라이언트와 서버 사이의 링크

다음 플랫폼을 사용할 수 있습니다. 이 결합은 사용하고 있는 IBM MQ 제품에 따라 다르며 127 페이지의 『IBM MQ 클라이언트의 플랫폼 지원』에 설명되어 있습니다.

IBM MQ MQI client	**IBM MQ 서버**
AIX and Linux Windows IBM i	AIX and Linux Windows IBM i z/OS

MQI는 클라이언트 플랫폼에서 실행되고 있는 애플리케이션에 사용 가능하며 큐 및 기타 IBM MQ 오브젝트는 서버에 설치한 큐 관리자에 보유됩니다.

IBM MQ MQI client 환경에서 실행하려는 애플리케이션은 관련 클라이언트 라이브러리와 먼저 링크되어야 합니다. 애플리케이션이 MQI 호출을 발행하는 경우 IBM MQ MQI client가 이 요청을 큐 관리자에게 전달하면 큐 관리자에서 이 요청이 처리되며 응답이 IBM MQ MQI client로 다시 전송됩니다.

애플리케이션과 IBM MQ MQI client 사이의 링크는 런타임 시에 동적으로 설정됩니다.

또한 IBM MQ classes for .NET, IBM MQ classes for Java 또는 IBM MQ classes for Java Message Service(JMS)을(를) 사용하여 클라이언트 애플리케이션을 개발할 수 있습니다. 다음 플랫폼에서 Java 및 JMS 클라이언트를 사용할 수 있습니다.

- IBM i
- AIX
- Linux
- Windows

Java 및 JMS의 사용은 여기에 설명되어 있지 않습니다. IBM MQ classes for Java 및 IBM MQ classes for JMS의 설치, 구성 및 사용에 대한 자세한 내용은 IBM MQ classes for Java 사용 및 IBM MQ classes for JMS 사용을 참조하십시오.

클라이언트 서버 환경의 IBM MQ 애플리케이션

서버에 링크되면 클라이언트 IBM MQ 애플리케이션은 로컬 애플리케이션과 같은 방식으로 MQI 호출의 대부분을 발행할 수 있습니다. 클라이언트 애플리케이션은 지정된 큐 관리자에 연결하기 위한 MQCONN 호출을 발행합니다. 연결 요청에서 리턴된 연결 핸들을 지정하는 추가 MQI 호출은 이 큐 관리자가 처리할 수 있습니다.

애플리케이션을 적절한 클라이언트 라이브러리로 링크해야 합니다. IBM MQ MQI clients의 애플리케이션 빌드를 참조하십시오.

관련 개념

127 페이지의 『IBM MQ 클라이언트를 사용하는 이유』
IBM MQ 클라이언트 사용은 IBM MQ 메시징 및 큐잉을 구현하는 효율적인 방법입니다.

128 페이지의 『확장된 트랜잭션 클라이언트 개념』
IBM MQ 확장된 트랜잭션 클라이언트는 외부 트랜잭션 관리자의 제어 하에 다른 자원 관리자가 관리하는 자원을 업데이트할 수 있습니다.

129 페이지의 『클라이언트를 서버에 연결하는 방법』
클라이언트는 MQCONN 또는 MQCONNX를 사용하여 서버에 연결되고 채널을 통해서 통신합니다.

131 페이지의 『트랜잭션 관리 및 지원』
트랜잭션 관리 및 IBM MQ가 트랜잭션을 지원하는 방법을 소개합니다.

132 페이지의 『큐 관리자 기능 확장』
사용자 엑시트, API 엑시트 또는 설치 가능 서비스를 사용하여 큐 관리자 기능을 확장할 수 있습니다.

관련 정보

IBM MQ MQI client를 설정하는 방법

IBM MQ 클라이언트를 사용하는 이유

IBM MQ 클라이언트 사용은 IBM MQ 메시징 및 큐잉을 구현하는 효율적인 방법입니다.

한 시스템에서 실행 중인 MQI 및 다른 시스템에서 실행 중인 큐 관리자를 사용하는 애플리케이션을 가질 수 있습니다(실제 또는 가상). 이를 실행할 경우의 장점은 다음과 같습니다.

- 클라이언트 시스템에 전체 IBM MQ 구현이 필요하지 않습니다.
- 클라이언트 시스템에 대한 하드웨어 요구사항이 감소됩니다.
- 시스템 관리 요구사항이 감소됩니다.
- 클라이언트에서 실행되는 IBM MQ 애플리케이션을 다른 시스템의 다중 큐 관리자에 연결할 수 있습니다.
- 다른 전송 프로토콜을 사용하는 대체 채널을 사용할 수 있습니다.

IBM MQ 클라이언트의 플랫폼 지원

지원되는 모든 서버 플랫폼의 IBM MQ는 플랫폼의 IBM MQ MQI clients에서의 클라이언트 연결을 승인합니다.

지원되는 모든 서버 플랫폼에 기본 제품 및 서버 로 설치된 IBM MQ 는 다음 플랫폼의 IBM MQ MQI clients 에서 연결을 승인할 수 있습니다.

- 

IBM i
- 

AIX
- 

Linux
- 

Windows

클라이언트 연결은 CCSID(Coded Character Set ID)와 통신 프로토콜과의 차이에 따라 달라집니다.

IBM MQ MQI client에서 실행하는 애플리케이션

클라이언트 환경에서 전체 MQI가 지원됩니다. 이를 통해 IBM MQ MQI client 의 애플리케이션을 MQI 라이브러리가 아닌 MQIC 라이브러리에 링크하여 거의 모든 IBM MQ 애플리케이션을 IBM MQ MQI client 시스템에서 실행하도록 구성할 수 있습니다. 예외사항은 다음과 같습니다.

- 신호가 있는 MQGET
- 확장된 트랜잭션 클라이언트를 사용해야 하며 다른 자원 관리자와의 동기점 조정이 필요한 애플리케이션

미리 읽기를 사용하는 경우 비지속 메시징 성능을 향상시키는 데 모든 MQGET 옵션이 사용 가능한 것은 아닙니다. 테이블은 허용되는 옵션 및 이러한 옵션이 MOGET 호출 사이에서 대체될 수 있는지 여부를 표시합니다.

표 *15.* 미리 읽기가 사용 가능한 경우 허용되는 *MQGET* 옵션			
값	**미리 읽기가 사용 가능한 경우 허용되며 MQGET 호출 간에 대체 가능**	**미리 읽기를 사용하는 경우 허용되지만 MQGET 호출 간에 대체될 수 없음[1]**	**미리 읽기가 사용 가능한 경우 허용되지 않는 MQGET 옵션[2]**
MQGET MD 값	MsgId [3] CorrelId [3]	Encoding CodedCharSetId	
MQGET MQGMO 옵션	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
MQGMO 값		MsgHandle	

1. 이 옵션이 MQGET 호출 간에 대체되면 MQRC_OPTIONS_CHANGED 이유 코드가 리턴됩니다.
2. 이 옵션을 첫 번째 MQGET 호출에 지정할 경우 미리 읽기를 사용할 수 없습니다. 이 옵션을 후속 MQGET 호출에 지정할 경우 이유 코드 MQRC_OPTIONS_ERROR가 리턴됩니다.
3. 클라이언트 애플리케이션은 MsgId 및 CorrelId 값이 MQGET 호출 간에 대체된 경우, 이전 값을 가진 메시지가 이미 클라이언트에 전송되어 이용(또는 자동으로 제거)될 때까지 클라이언트 미리 읽기 버퍼에 남아있는지 알아야 합니다.
4. 첫 번째 MQGET 호출은 메시지를 찾아볼지 아니면 미리 읽기가 가능할 때 큐에서 가져올지 여부를 판별합니다. 애플리케이션이 찾아보기와 가져오기의 결합을 사용하려고 할 경우 MQRC_OPTIONS_CHANGED 이유 코드가 리턴됩니다.
5. MQGMO_MSG_UNDER_CURSOR는 미리 읽기에 사용할 수 없습니다. 메시지를 찾아보거나 미리 읽기가 가능할 때 가져올 수 있지만, 이 둘을 결합할 수는 없습니다.

IBM MQ MQI client에서 실행 중인 애플리케이션은 동시에 둘 이상의 큐 관리자에 연결되거나 MQCONN 또는 MQCONNX 호출에서 별표(*)가 있는 큐 관리자 이름을 사용할 수 있습니다(큐 관리자에 IBM MQ MQI client MQI 클라이언트 애플리케이션 연결의 예제 참조).

확장된 트랜잭션 클라이언트 개념

IBM MQ 확장된 트랜잭션 클라이언트는 외부 트랜잭션 관리자의 제어 하에 다른 자원 관리자가 관리하는 자원을 업데이트할 수 있습니다.

트랜잭션 관리의 개념에 익숙하지 않은 경우 131 페이지의 『트랜잭션 관리 및 지원』의 내용을 참조하십시오.

현재 XA 트랜잭션 클라이언트가 IBM MQ의 일부로 제공되고 있습니다.

클라이언트 애플리케이션은 연결된 큐 관리자가 관리하는 작업 단위에 참여할 수 있습니다. 작업 단위 내에서 클라이언트 애플리케이션은 해당 큐 관리자가 소유하는 큐에 메시지를 넣고 큐에서 메시지를 가져올 수 있습니다. 클라이언트 애플리케이션은 **MQCMIT** 호출을 사용하여 작업 단위를 커미트하거나 **MQBACK** 호출을 사용하여 작업 단위를 백아웃할 수 있습니다. 그러나 동일한 작업 단위 내에서 클라이언트 애플리케이션은 다른 자원 관리자

의 자원(예: Db2® 데이터베이스의 테이블)을 업데이트할 수 없습니다. IBM MQ 확장된 트랜잭션 클라이언트를 사용하면 이 제한사항이 제거됩니다.

IBM MQ 확장된 트랜잭션 클라이언트는 일부 추가 기능이 있는 IBM MQ MQI client입니다. 이 기능을 사용하면 동일한 작업 단위 내에 있는 클라이언트 애플리케이션에서 다음 태스크를 수행할 수 있습니다.

- 연결된 큐 관리자가 소유하는 큐에 메시지 넣기 및 큐에서 메시지 가져오기
- IBM MQ 큐 관리자가 아닌 자원 관리자의 자원 업데이트

이 작업 단위는 클라이언트 애플리케이션과 동일한 시스템에서 실행 중인 외부 트랜잭션 관리자에 의해 관리되어야 합니다. 작업 단위는 클라이언트 애플리케이션이 연결된 큐 관리자에 의해 관리될 수 없습니다. 이는 큐 관리자가 트랜잭션 관리자가 아닌 자원 관리자로서만 작업을 수행할 수 있음을 의미합니다. 또한 클라이언트 애플리케이션이 외부 트랜잭션 관리자가 제공한 API(Application Programming Interface)만을 사용하여 작업 단위를 커미트하거나 백아웃할 수 있음을 의미합니다. 클라이언트 애플리케이션은 따라서 MQI 호출, **MQBEGIN**, **MQCMIT** 및 **MQBACK**을 사용할 수 없습니다.

외부 트랜잭션 관리자는 큐 관리자에 연결된 클라이언트 애플리케이션이 사용하는 것과 동일한 MQI 채널을 사용하는 자원 관리자로서 큐 관리자와 통신합니다. 하지만 장애를 복구하는 상황에서, 실행되는 애플리케이션이 없을 때 트랜잭션 관리자는 전용 MQI 채널을 사용하여 큐 관리자가 실패 시에 참여하고 있었던 불완전한 작업 단위를 복구할 수 있습니다.

이 절에서 확장 트랜잭션 기능이 없는 IBM MQ MQI client는 IBM MQ 기본 클라이언트로 참조됩니다. 따라서 IBM MQ 확장된 트랜잭션 클라이언트가 확장 트랜잭션 기능이 추가된 IBM MQ 기본 클라이언트로 구성되는 것으로 간주할 수 있습니다.

참고: IBM i IBM i 의 IBM MQ MQI client 는 IBM MQ 확장 트랜잭션 기능을 지원하지 않습니다.

확장된 트랜잭션 클라이언트의 플랫폼 지원

Multi

확장된 트랜잭션 클라이언트는 기본 클라이언트를 지원하는 모든 멀티플랫폼에 대해 사용 가능합니다. z/OS에 대해 클라이언트가 사용 불가능합니다.

확장 트랜잭션 클라이언트를 사용하는 클라이언트 애플리케이션은 다음 IBM MQ 9.0이상 제품의 큐 관리자에만 연결할 수 있습니다.

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ for Linux
- Windows IBM MQ for Windows

z/OS z/OS에서 실행되는 확장된 트랜잭션 클라이언트가 없는 경우에도 확장된 트랜잭션 클라이언트를 사용 중인 클라이언트 애플리케이션을 z/OS에서 실행되는 큐 관리자에 연결할 수 있습니다.

각 플랫폼의 경우, 확장된 트랜잭션 클라이언트에 대한 하드웨어 및 소프트웨어 요구사항은 IBM MQ 기본 클라이언트에 대한 요구사항과 동일합니다. 프로그래밍 언어는 사용 중인 IBM MQ 기본 클라이언트 및 트랜잭션 관리자가 지원하는 경우에는 확장된 트랜잭션 클라이언트에 의해 지원됩니다.

모든 플랫폼의 외부 트랜잭션 관리자에 대한 정보는 IBM MQ 의 시스템 요구사항의 내용을 참조하십시오.

클라이언트를 서버에 연결하는 방법

클라이언트는 MQCONN 또는 MQCONNX를 사용하여 서버에 연결되고 채널을 통해서 통신합니다.

IBM MQ 클라이언트 환경에서 실행 중인 애플리케이션은 클라이언트와 서버 시스템 사이의 활성 연결을 유지해야 합니다.

MQCONN 또는 MQCONNX를 발행하는 애플리케이션을 통해 연결이 설정되었습니다. 클라이언트 및 서버는 *MQI* 채널을 통해 통신하거나 공유 대화를 사용하는 경우에는 각각의 대화가 MQI 채널 인스턴스를 공유합니다.

호출에 성공한 경우 MQI 채널 인스턴스 또는 대화는 애플리케이션이 MQDISC 호출을 발행할 때까지 연결된 상태로 유지됩니다. 이는 애플리케이션이 연결해야 하는 모든 큐 관리자에 해당됩니다.

동일한 시스템의 클라이언트 및 큐 관리자

시스템에 큐 관리자가 설치되어 있는 경우 IBM MQ MQI client 환경에서 애플리케이션을 실행할 수도 있습니다.

이 상황에서 큐 관리자 라이브러리에 링크할지 또는 클라이언트 라이브러리에 링크할지에 대한 선택을 할 수 있지만 클라이언트 라이브러리에 링크하는 경우 여전히 채널 연결을 정의할 필요가 있다는 점에 유의하십시오. 이는 애플리케이션의 개발 단계에서 유용할 수 있습니다. 자체 시스템에서 다른 시스템에 대한 종속성 없이 프로그램을 테스트할 수 있으며 프로그램을 독립적 IBM MQ MQI client 환경으로 이동시켜도 이 프로그램이 작동하는지에 대해 확인할 수 있습니다.

다른 플랫폼의 클라이언트

이 예에서는 서버 시스템이 다른 플랫폼에 있는 세 개의 IBM MQ MQI clients와 통신합니다.

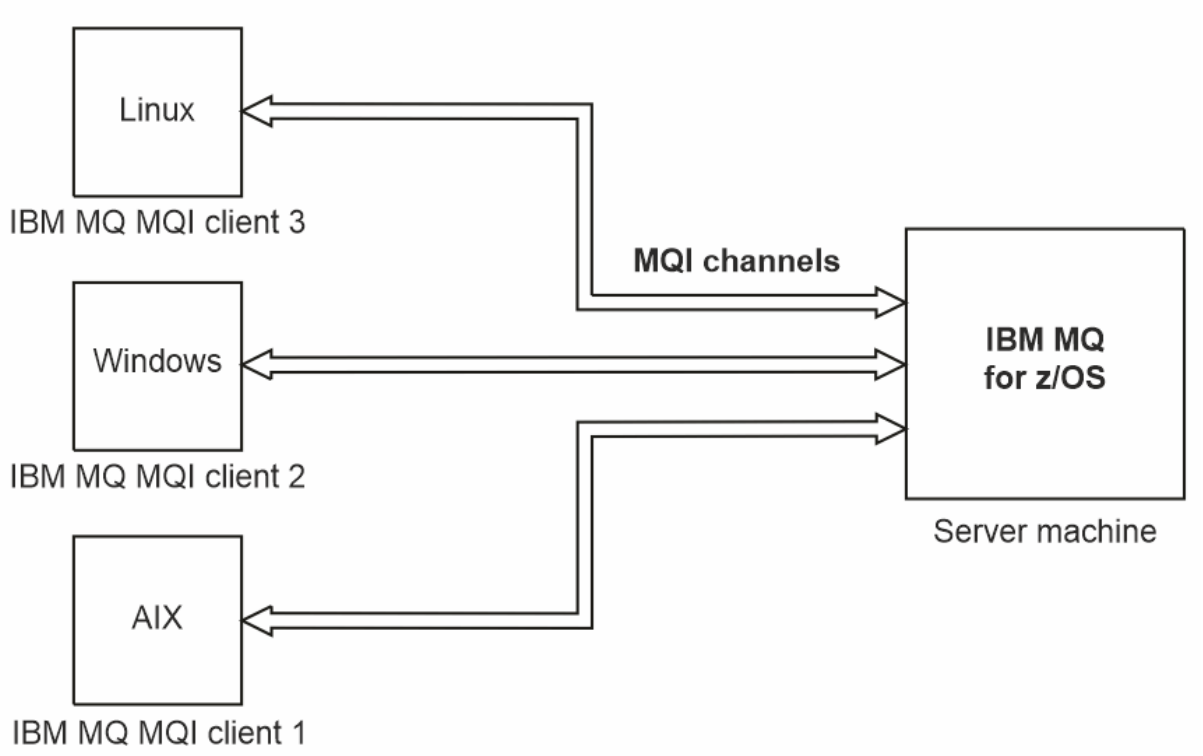


그림 *52.* 다른 플랫폼의 클라이언트에 연결된 *IBM MQ* 서버

다른 좀 더 복잡한 환경도 가능합니다. 예를 들어, IBM MQ 클라이언트는 둘 이상의 큐 관리자 또는 큐 공유 그룹의 일부로서 연결된 모든 큐 관리자에 연결할 수 있습니다.

다른 버전의 클라이언트 및 서버 소프트웨어 사용

이전 버전의 IBM MQ 제품을 사용하는 경우 클라이언트의 CCSID에서 코드 변환을 서버가 지원하는지 확인하십시오.

IBM MQ 클라이언트는 지원되는 모든 버전의 큐 관리자에 연결할 수 있습니다. 이전 버전 큐 관리자에 연결하는 경우, 클라이언트의 IBM MQ 애플리케이션에서 제품의 이후 버전 기능 및 구조를 사용할 수 없습니다.

IBM MQ 큐 관리자는 상호 지원되는 최상위 프로토콜 레벨로 협상하여 서로 다른 버전의 클라이언트와 자체적으로 통신할 수 있습니다. 이는 이전 클라이언트가 이후 큐 관리자 레벨에서 사용될 수 있음을 의미합니다. 문제점 진단을 용이하게 하고 IBM의 지원을 사용 가능하게 하기 위해 클라이언트 및 서버가 모두 현재 지원되는 IBM MQ 버전에 있는 것이 좋습니다.

자세한 정보는 애플리케이션 개발에서 지원되는 프로그래밍 언어를 참조하십시오.

트랜잭션 관리 및 지원

트랜잭션 관리 및 IBM MQ가 트랜잭션을 지원하는 방법을 소개합니다.

자원 관리자는 애플리케이션이 액세스하고 업데이트할 수 있는 자원을 소유하고 관리하는 컴퓨터 서브시스템입니다. 다음은 자원 관리자의 예입니다.

- 큐 자원이 있는 IBM MQ 큐 관리자
- 테이블 자원이 있는 Db2 데이터베이스

애플리케이션이 하나 이상의 자원 관리자의 자원을 업데이트할 때, 특정 업데이트가 모두 그룹으로 완료되거나 어떠한 업데이트도 완료되지 않도록 하는 비즈니스 요구사항이 있을 수 있습니다. 이 유형의 요구사항에 대한 이유는 이러한 업데이트 중 일부가 성공적으로 완료되었지만 그 외의 업데이트는 그렇지 못한 경우에 비즈니스 데이터가 일관되지 않은 상태로 남겨지기 때문입니다.

이러한 방식으로 관리되는 자원에 대한 업데이트는 작업 단위 또는 트랜잭션 내에서 발생합니다. 애플리케이션 프로그램은 한 세트의 업데이트를 하나의 작업 단위로 그룹화할 수 있습니다.

작업 단위 중에 애플리케이션은 자원 관리자에게 요청을 발행하여 자원 관리자의 자원을 업데이트합니다. 작업 단위는 애플리케이션이 모든 업데이트 커미트 요청을 발행할 때 종료됩니다. 업데이트가 커미트될 때까지 동일한 자원에 액세스 중인 다른 애플리케이션에 아무 업데이트도 표시되지 않습니다. 그렇지 않은 경우 애플리케이션이 어떤 이유로 작업 단위를 완료할 수 없다고 결정하면 애플리케이션은 요청했던 모든 업데이트를 해당 지점까지 백아웃하도록 하는 요청을 발행할 수 있습니다. 이러한 경우에 모든 업데이트는 다른 애플리케이션에 표시되지 않습니다. 이러한 업데이트는 보통 논리적으로 연관되어 있으며 보존되어야 하는 데이터 무결성에 대해 모두 성공적이어야 합니다. 다른 업데이트가 실패할 때 한 업데이트가 성공하면 데이터 무결성을 잃게 됩니다.

작업 단위가 성공적으로 완료되면 커미트라고 표시됩니다. 일단 커미트되면 해당 작업 단위 내에서 수행된 모든 업데이트는 영구적이 되며 되돌릴 수 없습니다. 하지만 작업 단위가 실패하면 모든 업데이트는 대신에 백아웃됩니다. 작업 단위가 커미트되거나 무결성으로 백아웃되는 이 프로세스를 동기점 조정이라고 합니다.

작업 단위 내 모든 업데이트가 커미트되거나 백아웃될 때의 지점을 동기점이라고 합니다. 작업 단위 내 업데이트는 동기점 제어 내에서 발생한다고 합니다. 애플리케이션이 동기점 제어 범위 외에 있는 업데이트를 요청하는 경우 진행 중인 작업 단위가 있고 업데이트를 나중에 백아웃할 수 없더라도 자원 관리자는 즉시 업데이트를 커미트합니다.

작업 단위를 관리하는 컴퓨터 서브시스템을 트랜잭션 관리자 또는 지점 통합기라고 합니다.

로컬 작업 단위에서는 업데이트된 유일한 자원이 IBM MQ 큐 관리자의 자원입니다. 여기에 1단계 커미트 프로세스를 사용하는 큐 관리자 스스로 동기점 조정을 제공합니다.

글로벌 작업 단위에서는 XA 준수 데이터베이스와 같이 다른 자원 관리자에 속하는 자원 또한 업데이트됩니다. 여기에, 2단계 커미트 프로시저가 사용되어야 하며 작업 단위는 큐 관리자 자체에서 조정할 수 있거나 IBM TXSeries® 또는 BEA Tuxedo와 같이 외부에서 다른 XA 준수 트랜잭션 관리자에 의해 조정할 수 있습니다.

트랜잭션 관리자는 작업 단위 내에서의 자원에 대한 모든 업데이트가 성공적으로 완료되는지 또는 업데이트가 전혀 완료되지 않는지 확인해야 합니다. 애플리케이션은 작업 단위의 커미트 또는 백아웃 요청을 트랜잭션 관리자에게 발행합니다. 트랜잭션 관리자의 예는 CICS 및 WebSphere Application Server이며, 이 두 트랜잭션 관리자는 모두 다른 기능도 가지고 있습니다.

일부 자원 관리자는 자체 트랜잭션 관리 기능을 제공합니다. 예를 들어, IBM MQ 큐 관리자는 자체 자원으로의 업데이트와 Db2 테이블로의 업데이트를 포함하여 작업 단위를 관리할 수 있습니다. 큐 관리자는 사용자가 요구하는 경우 트랜잭션 관리자를 사용할 수 있기는 하지만 이 기능을 수행할 별도의 트랜잭션 관리자를 필요로 하지 않습니다. 별도의 트랜잭션 관리자가 사용되는 경우 외부 트랜잭션 관리자로 참조됩니다.

작업 단위를 관리하는 외부 트랜잭션 관리자의 경우 트랜잭션 관리자와 작업 단위에 참여하는 모든 자원 관리자 사이에는 표준 인터페이스가 있어야 합니다. 이 인터페이스를 통해 트랜잭션 관리자와 자원 관리자가 서로 통신할 수 있습니다. 이러한 인터페이스 중 하나는 *XA* 인터페이스이며 이 인터페이스는 다수의 트랜잭션 관리자 및 자원 관리자가 지원하는 표준 인터페이스입니다. XA 인터페이스는 분배된 트랜잭션 처리*: XA* 스펙에서 The Open Group에 의해 발행됩니다.

둘 이상의 자원 관리자가 작업 단위에 참여한 경우 트랜잭션 관리자는 *2*단계 커미트 프로토콜을 사용하여 시스템 장애가 있다 하더라도 작업 단위 내의 모든 업데이트가 성공적으로 완료되었는지 또는 업데이트가 전혀 완료되지 않았는지 확인해야 합니다. 애플리케이션이 트랜잭션 관리자에게 작업 단위 커미트 요청을 발행하면 트랜잭션 관리자는 다음을 수행합니다.

단계 1(커미트 준비)
트랜잭션 관리자가 작업 단위에 참여하고 있는 각 자원 관리자에게 자원에 대해 예정된 업데이트에 대한 모든 정보가 복구 가능한 상태에 있는지 확인할 것을 요청합니다. 자원 관리자는 일반적으로 로그에 정보를 쓰고 정보가 하드 디스크를 통해 쓰여졌는지 확인하여 이러한 작업을 수행합니다. 단계 1은 트랜잭션 관리자가 각 자원 관리자로부터 자원에 대해 예정된 업데이트에 대한 정보가 복구 가능한 상태에 있다는 알림을 수신하면 완료됩니다.

단계 2(커미트)
단계 1이 완료되면 트랜잭션 관리자는 취소 불가능한 작업 단위 커미트 결정을 내립니다. 트랜잭션 관리자는 작업 단위에 참여하는 각 자원 관리자에게 자원에 대한 업데이트 커미트를 요청합니다. 자원 관리자가 이 요청을 수신하면 업데이트를 커미트해야 합니다. 자원 관리자에게는 이 단계에서 업데이트를 백아웃하는 옵션이 없습니다. 단계 2는 트랜잭션 관리자가 각 자원 관리자로부터 자원에 대한 업데이트를 커미트했다는 알림을 수신하면 완료됩니다.

XA 인터페이스는 2단계 커미트 프로토콜을 사용합니다.

자세한 정보는 트랜잭션 지원 시나리오를 참조하십시오.

또한 IBM MQ는 Microsoft Transaction Server(COM+)에 대한 지원을 제공합니다. Microsoft Transaction Server(COM+) 사용에서 COM+ 지원을 활용하도록 IBM MQ를 설정하는 방법에 대한 정보를 제공합니다.

큐 관리자 기능 확장

사용자 엑시트, API 엑시트 또는 설치 가능 서비스를 사용하여 큐 관리자 기능을 확장할 수 있습니다.

사용자 엑시트

사용자 엑시트는 사용자가 고유 코드를 큐 관리자 기능에 삽입할 수 있도록 하는 메커니즘을 제공합니다. 지원되는 사용자 엑시트에는 다음이 포함됩니다.

채널 엑시트
이 엑시트는 채널이 작동하는 방식을 변경합니다. 채널 엑시트는 메시지 채널에 대한 채널 엑시트 프로그램에 설명되어 있습니다.

데이터 변환 엑시트
이러한 엑시트는 데이터 형식을 하나의 형식에서 다른 형식으로 변환하는 애플리케이션 프로그램에 넣을 수 있는 소스 코드 단편을 작성합니다. 데이터 변환 엑시트는 데이터 변환 엑시트 작성에 설명되어 있습니다.

클러스터 워크로드 엑시트
이 엑시트가 수행하는 기능은 엑시트의 제공자에 의해 정의됩니다. 호출 정의 정보는 MQ_CLUSTER_WORKLOAD_EXIT - 호출 설명에 제공되어 있습니다.

API 엑시트

API 엑시트를 통해 사용자는 MQPUT 및 MQGET과 같이 IBM MQ API 호출의 작동을 변경하는 코드를 작성하고 해당 코드를 이러한 호출 바로 앞이나 뒤에 삽입할 수 있습니다. 삽입은 자동이며 큐 관리자는 등록된 지점에서 엑시트 코드를 드라이브합니다. API 엑시트에 대한 자세한 정보는 API 엑시트 사용 및 작성을 참조하십시오.

설치 가능 서비스

설치 가능 서비스가 다중 시작점이 있는 인터페이스(API)를 형식화했습니다.

설치 가능 서비스의 구현을 서비스 컴포넌트라고 합니다. IBM MQ에서 제공된 컴포넌트를 사용하거나 고유 컴포넌트를 작성하여 필요한 기능을 수행할 수 있습니다.

현재 다음 설치 가능 서비스가 제공됩니다.

권한 서비스
권한 서비스를 통해 고유 보안 기능을 빌드할 수 있습니다.

서비스를 구현하는 기본 서비스 컴포넌트는 오브젝트 권한 관리자(OAM)입니다. 기본적으로 OAM은 활성이며 OAM 구성을 위해 어떤 조치도 실행할 필요가 없습니다. 권한 서비스 인터페이스를 사용하여 OAM을 바꾸거나 보강시킬 다른 컴포넌트를 작성하십시오. OAM에 대한 자세한 정보는 AIX, Linux, and Windows 시스템에서 보안 설정을 참조하십시오.

이름 서비스

이름 서비스를 사용하면 애플리케이션이 리모트 큐를 로컬 큐인 것처럼 식별하여 큐를 공유할 수 있습니다.

고유의 이름 서비스 컴포넌트를 작성할 수 있습니다. 예를 들어 IBM MQ에 대해 이름 서비스를 사용하려는 경우 이러한 작업을 수행하려고 할 수 있습니다. 이름 서비스를 사용하려면 사용자 작성 또는 다른 소프트웨어 벤더가 공급한 컴포넌트가 있어야 합니다. 기본적으로, 이름 서비스는 비활성 상태입니다.

관련 개념

사용자 엑시트, API 엑시트 및 IBM MQ 설치 가능 서비스

IBM MQ Java 언어 인터페이스

IBM MQ 는 Java 애플리케이션에서 사용할 세 개의 API (Application Programming Interface) (IBM MQ classes for Jakarta Messaging, IBM MQ classes for JMS및 IBM MQ classes for Java) 를 제공합니다.

IBM 는 개방형 표준을 지원하며 개방형 표준의 활성 참여자입니다.

- IBM MQ 8.0부터 제품은 공유 구독과 같은 기능과 함께 단순화된 새 API를 도입한 JMS 2.0 표준을 구현합니다.
- IBM MQ 9.3.0부터 Jakarta Messaging 3.0 도 지원됩니다.
- 또한 WebSphere Liberty 는 IBM MQ와 함께 JMS 2.0 및 Jakarta Messaging 3.0 를 지원합니다.

IBM MQ 내에는 Java 애플리케이션에서 사용할 세 개의 API가 있습니다.

JM 3.0 **IBM MQ classes for Jakarta Messaging**

IBM MQ classes for Jakarta Messaging 는 IBM MQ 용 Jakarta Messaging 인터페이스를 메시징 시스템으로 구현하는 Jakarta Messaging 제공자입니다. Jakarta Connectors Architecture 는 Jakarta EE 환경에서 실행 중인 애플리케이션을 IBM MQ 또는 Db2와 같은 EIS (Enterprise Information System) 에 연결하는 표준 방법을 제공합니다.

JMS 2.0 **IBM MQ classes for JMS**

IBM MQ classes for JMS 는 IBM MQ 용 JMS 인터페이스를 메시징 시스템으로 구현하는 JMS 제공자입니다. Java Platform, Enterprise Edition Connector Architecture(JCA)는 Java EE 환경에서 실행 중인 애플리케이션을 EIS(Enterprise Information System)(예: IBM MQ 또는 Db2)에 연결하는 표준 방법을 제공합니다.

IBM MQ classes for Java

IBM MQ classes for Java 를 사용하면 Java 환경에서 IBM MQ 를 사용할 수 있습니다. IBM MQ classes for Java를 사용하면 Java 애플리케이션을 IBM MQ 클라이언트로 IBM MQ에 연결하거나 IBM MQ 큐 관리자에 직접 연결할 수 있습니다.

참고:

- JMS 2.0 은 Jakarta Messaging로 대체되었습니다. IBM MQ classes for JMS 는 JMS 2.0 표준을 계속 지원하지만 Java 메시징에 대한 향후 개선사항은 Jakarta Messaging에서만 나타나므로 IBM MQ classes for Jakarta Messaging에서도 나타납니다. IBM MQ classes for JMS 는 기존 JMS 2.0 애플리케이션을 유지보수하고 확장하는 경우에만 권장됩니다. IBM MQ classes for Jakarta Messaging 는 새 개발에 선호되는 기술이어야 합니다.
- Stabilized IBM MQ classes for Java는 IBM MQ 8.0에서 제공된 레벨에서 기능을 안정적으로 사용할 수 있습니다. IBM MQ classes for Java를 사용하는 기존 애플리케이션이 계속해서 완전히 지원되지만 이 API가 안정되었으므로 새 기능이 추가되지 않으며 개선 요청도 수락되지 않습니다. 완전히 지원이란 IBM MQ 시스템 요구사항의 변경에 따라 필요한 모든 변경사항과 함께 결함이 수정되는 것을 의미합니다.

JM 3.0 IBM MQ 9.3부터 IBM MQ classes for Java, IBM MQ classes for JMS및 IBM MQ classes for Jakarta Messaging 는 Java 8를 사용하여 빌드됩니다. 이러한 레벨 이상의 Java 런타임 환경은 이러한 인터페이스를 사용하여 애플리케이션을 실행하는 데 사용해야 합니다.

관련 개념

Java 에서 IBM MQ 에 액세스-API 선택

Jakarta Messaging에서 IBM MQ 클래스를 사용해야 하는 이유는 무엇입니까?

IBM MQ classes for JMS를 사용해야 하는 이유가 무엇입니까?

IBM MQ classes for Java를 사용해야 하는 이유가 무엇입니까?

IBM MQ classes for JMS/Jakarta Messaging

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 는 IBM MQ와 함께 제공되는 메시징 제공자입니다. 이러한 각 제공자는 메시징 API에 대한 두 개의 확장 세트도 제공합니다. Java Platform, Standard Edition (Java SE) 및 Java Platform, Enterprise Edition (Java EE) 애플리케이션 모두 이러한 메시징 제공자를 사용할 수 있습니다.

JM 3.0 IBM MQ 9.3.0 에서는 Jakarta Messaging 3.0에 대한 지원을 도입했습니다. JMS 2.0 여전히 완전히 지원됩니다.

JMS 및 Jakarta Messaging 스펙은 애플리케이션이 메시징 조작을 수행하는 데 사용할 수 있는 인터페이스 세트를 정의합니다. 제품은 JMS 표준의 JMS 2.0 버전을 지원합니다. 이 구현은 클래식 API의 모든 기능을 제공하지만 필요한 인터페이스가 더 적고 사용이 더 단순합니다. 자세한 정보는 137 페이지의 『JMS 및 Jakarta Messaging 모델』 및 JMS 2.0 스펙 (Java.net) 을 참조하십시오. JM 3.0 IBM MQ 9.3.0에서는 Jakarta Messaging 도 지원됩니다.

jakarta.jms (Jakarta Messaging 3.0) 또는 javax.jms (JMS 2.0) 패키지는 메시징 인터페이스의 세부사항을 지정하며 메시징 제공자는 특정 메시징 제품에 대해 이러한 인터페이스를 구현합니다. 예를 들면, 다음과 같습니다.

- IBM MQ classes for JMS 는 IBM MQ 용 JMS 인터페이스를 구현하는 JMS 제공자이며 JMS API에 대한 다음 두 개의 확장 세트도 제공합니다.
  - IBM MQ JMS 확장
  - IBM JMS 확장
- javax.dims또는 jakarta.jms인터페이스 또는 JMS 확장 세트를 사용하여 작성된 연결 팩토리, 큐 또는 토픽 오브젝트는 이러한 API중 하나를 사용하여 주소 지정할 수 있습니다. 즉, 임의의 인터페이스로 캐스트할 수 있습니다. 애플리케이션 이식 가능성을 최고 수준으로 유지하려면 사용자의 요구사항에 맞는 가장 일반적인 API를 사용하십시오.

JMS 및 Jakarta Messaging 는 공통적으로 많이 공유하므로 이 주제에서 JMS 에 대한 추가 참조는 둘 다를 참조하는 것으로 간주할 수 있습니다. 필요에 따라 차이점이 강조표시됩니다.

IBM MQ JMS 확장

IBM MQ classes for JMS는 또한 JMS API에 대한 확장 기능을 제공합니다. IBM MQ classes for JMS 에는 MQConnectionFactory, MQQueue및 MQTopic 오브젝트에서 구현되는 확장이 포함되어 있습니다. 이러한 오브젝트에는 IBM MQ에 특정한 특성 및 메소드가 있습니다. 오브젝트는 관리 대상 오브젝트일 수 있거나 애플리케이션은 런타임에 오브젝트를 동적으로 작성할 수 있습니다. 이러한 확장을 IBM MQ JMS 확장이라고 합니다. 이 문서에서 런타임 시 애플리케이션에 의해 동적으로 작성되는 오브젝트는 관리 오브젝트로 간주되지 않습니다.

IBM JMS 확장

IBM MQ JMS 확장 외에도 IBM MQ classes for JMS 는 사용되는 프로그래밍 언어로 JMS API 또는 Java 에 보다 일반적인 확장 세트를 제공합니다. 이러한 확장을 IBM JMS 확장이라고 하며 다음과 같은 광범위한 목표가 있습니다.

- IBM JMS 제공자 간에 더 높은 레벨의 일관성을 제공합니다.
- 두 IBM 메시징 시스템 간에 브릿지 애플리케이션을 더 쉽게 작성할 수 있습니다.
- 한 IBM JMS 제공자에서 다른 제공자로 애플리케이션을 더 쉽게 이식할 수 있습니다.

이러한 확장의 주요 초점은 런타임에 동적으로 연결 팩토리 및 목적지를 작성 및 구성하는 점에 맞춰져 있지만, 확장에서는 문제점 판별 기능과 같이 메시징과 직접적으로 관련되지 않은 기능도 제공합니다.

관련 태스크

JMS/Jakarta Messaging에 IBM MQ 클래스 사용

JMS및 Jakarta Messaging 자원 구성

JM 3.0 IBM MQ classes for Jakarta Messaging: 개요

IBM MQ 9.3.0 에서는 Jakarta Messaging에 대한 지원을 소개합니다. Jakarta Messaging 3.0의 경우 JMS 스펙의 제어가 Oracle 에서 Java 커뮤니티 프로세스로 이동되었습니다. 그러나 Oracle 는 다른 Java 기술에서 사용되는 "javax" 이름의 제어를 보유합니다. 따라서 Jakarta Messaging 3.0 는 기능적으로 JMS 2.0와 동일하지만 이름 지정에는 몇 가지 차이점이 있습니다. 버전 3.0 의 공식 이름은 Java Message Service가 아니라 Jakarta Messaging 이며, 패키지 및 상수 이름 앞에는 `javax`가 아닌 `jakarta` 가 붙습니다.

백그라운드

수년 동안 Java 플랫폼은 두 가지 양식 (Standard Edition 및 Enterprise Edition) 으로 제공되었습니다.

Java Platform, Standard Edition (때로는 Java SE로 축약됨) 는 독립형 컨텍스트에서 실행할 수 있는 코어 언어 및 클래스 라이브러리입니다. Java SE 의 대부분의 Java 패키지 이름은 "java." 로 시작합니다.

Java Platform, Enterprise Edition (Java EE) 는 메시징, 다양한 Bean, 트랜잭션성 등과 같은 기능을 추가하여 이를 확장합니다. 이러한 기술 중 일부는 Java SE 컨텍스트에서도 사용할 수 있습니다. Java EE 의 대부분의 Java 패키지에는 "javax" 로 시작하는 이름이 있습니다. 그러나 일부 교차가 있으므로 일부 Java SE 패키지에는 "javax" 가 있습니다. 이름에 대한 접두부로 사용됩니다.

Java Message Service (JMS) 는 Java Platform, Enterprise Edition의 일부입니다. Java EE 7 는 JMS 2.0를 통합합니다.

Java EE 7까지 이 기술은 Oracle의 관리 하에 있었습니다.

Java EE 기술은 최근에 Oracle 의 스튜워드십에서 Eclipse Foundation이 감독하는 커뮤니티 프로세스로 이동했습니다.

"javax." 이름을 새 프로젝트로 이동할 수 없습니다. 새 이름 지정이 채택되었습니다. 모든 패키지 및 특성 이름에 "jak자카르타" 라는 접두부가 추가되었습니다. Java Platform, Enterprise Edition 은 나중에 "Jakarta EE" 로 이름 지정됩니다. 버전 번호 지정이 계속되었습니다. 버전 8은 대부분 무시할 수 있는 임시 버전이며 Jakarta EE 9 는 "jak자카르타" 가 있는 지점입니다. 접두부가 적용됩니다.

IBM MQ 컨텍스트에 적용되는 기본 Jakarta EE 기술은 Jakarta Messaging 3.0 - Java Message Service (JMS) 2.0의 후속 기술입니다. 따라서 Jakarta EE 9 는 Jakarta Messaging 3.0를 통합합니다.

Jakarta EE 9 및 Jakarta Messaging 3.0에 대한 지원을 도입하는 동안 IBM MQ 는 계속해서 Java EE 7 및 JMS 2.0를 지원합니다.

제공되는 내용: Java SE

Java Platform, Standard Edition의 경우 IBM MQ classes for JMS (IBM MQ에서 JMS 2.0 조작을 지원함) IBM MQ 9.3.0 및 이후 버전에서는 IBM MQ classes for Jakarta Messaging를 제공합니다. 이러한 클래스는 IBM MQ와 통합되는 Jakarta Messaging 3.0 제공자를 제공하여 Jakarta Messaging 조작을 용이하게 하기 위해 IBM MQ 큐 관리자를 사용할 수 있도록 합니다.

이 파일은 IBM MQ 설치의 `java/lib` 서브디렉토리에 표준 JAR 파일 `com.ibm.mq.jakarta.client.jar` 로 제공됩니다.

OSGi 컨테이너 (예: Apache Felix 또는 Eclipse Equinox IBM MQ) 에서 사용하기 위해 다음과 같은 OSGi번들 쌍도 제공합니다.

- `com.ibm.mq.osgi.jms30.clientprereqs_`*`V.R.M.F`*`.jar`
- `com.ibm.mq.osgi.jms30.client_`*`V.R.M.F`*`.jar`

여기서 *V.R.M.F* 는 IBM MQ의 버전을 나타냅니다 (예: 9.3.0.0). 이러한 번들은 IBM MQ 설치의 `java/lib/OSGi` 서브디렉토리에서 찾을 수 있습니다.

제공되는 내용: Jakarta EE 9

Jakarta EE 9 호환 가능 애플리케이션 서버에서 IBM MQ기반 메시징을 지원하기 위해 IBM MQ 는 Jakarta EE 9 호환 가능 자원 어댑터 (`wmq.jakarta.jmsra.rar`) 를 제공합니다. 이는 IBM MQ 설치의 `java/lib/jca` 서브디렉토리에서 찾을 수 있습니다.

IBM MQ 는 계속해서 IBM MQ 설치의 `java/lib/jca` 서브디렉토리에 Java EE 7 호환 가능 자원 어댑터 `wmq.jmsra.rar`를 제공합니다.

이러한 아티팩트가 전달되는 방법

자원 어댑터의 이러한 JAR및 RAR 파일은 일반적인 IBM MQ 설치 매체 ("`.rpm`" 파일과 같은 플랫폼 특정 설치 매체 및 자체 추출 재분배 가능 클라이언트 JAR 파일과 같은 재분배 가능 매체 둘 다) 에 기존 아티팩트와 함께 패키지됩니다.

JMS 2.0 및 Jakarta Messaging 3.0 간에 변경된 사항

Jakarta EE 9 및 Jakarta Messaging 3.0 에서는 새 기능을 도입하지 않습니다. 모든 변경사항은 이름입니다. 예를 들어, JMS 2.0에서 "javax.jms.Connection" 을 사용하는 경우 Jakarta Messaging 3.0에서 "jakarta.jms.Connection" 을 사용합니다.

Eclipse Foundation은 Jakarta EE 플랫폼을 앞으로 사용하므로 이 기반에서 빌드되며 이 이름 지정 규칙은 나중에 도입되는 새 기능에 사용됩니다.

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 간에 변경된 사항

요약

JMS 2.0에 대한 지원을 제공하는 IBM MQ classes for JMS는 계속 사용 가능하며 기본적으로 기존 애플리케이션을 유지보수하고 확장하는 데 권장됩니다. 완전히 지원됩니다.

Jakarta Messaging 3.0에 대한 지원을 제공하는 IBM MQ classes for Jakarta Messaging는 새 개발에 권장됩니다.

IBM MQ 9.3.0에서 이 두 오퍼링은 기능적으로 동등합니다. 이름만 다릅니다. 그러나 새 메시징 기능은 IBM MQ classes for JMS에서보다 IBM MQ classes for Jakarta Messaging 에서 나타날 가능성이 높습니다.

두 오퍼링은 상호 운용 가능합니다. IBM MQ classes for JMS 에서 생성된 메시지는 IBM MQ classes for Jakarta Messaging에서 이용할 수 있으며 그 반대의 경우도 가능합니다. 그러나 두 오퍼링은 단일 애플리케이션에서 공존할 수 없습니다.

이름 지정 변경사항

표 *16.* 패키지 이름에 대한 변경사항

IBM MQ classes for JMS 패키지 이름	**IBM MQ classes for Jakarta Messaging 패키지 이름**
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.*

공통 서비스 (추적, 로깅, 자국어 지원 등) 및 JMQI 구현 (로컬 및 원격) 과 관련된 패키지는 IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging모두에 공통되므로 이러한 영역에서 변경이 필요하지 않습니다.

특성 이름도 변경되었습니다. 예를 들어, IBM MQ classes for Jakarta Messaging 에서 IBM MQ 확장을 사용으로 설정하는 특성은 **`com.ibm.mq.jakarta.jms.SupportMQExtensions`**입니다.

IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging에 독립적인 특성 이름 (예: 다양한 **`com.ibm.msg.client.commonservices.trace.*`** 특성) 은 두 오퍼링 모두에 동일하게 적용됩니다.

관리 유틸리티

`crtmqenv` 및 **`setmqenv`** 유틸리티는 이제 클래스 경로가 IBM MQ classes for JMS (`-j 2.0`) 또는 IBM MQ classes for Jakarta Messaging (`-j 3.0`) 에 대해 구성되어야 하는지 여부를 지정하는 옵션을 승인하

며, **runjms30** 및 유사한 이름이라고 하는 **runjms** 유틸리티의 IBM MQ classes for Jakarta Messaging 변형이 있습니다.

dspmqver 유틸리티는 Java 구성요소에 대해 보고하도록 요청될 때 출력에 IBM MQ classes for Jakarta Messaging 를 포함합니다.

JNDI를 통해 IBM MQ classes for Jakarta Messaging 오브젝트를 검색하도록 구성하기 위해 새 **JMS30Admin** 유틸리티는 IBM MQ classes for JMS용 **JMSAdmin** 유틸리티와 동일합니다.

기본 오브젝트는 서로 다른 패키지에서 가져온 것입니다. **JMSAdmin** 에 의해 작성된 JNDI 정의는 IBM MQ classes for Jakarta Messaging에 의해 사용될 수 없으며, **JMS30Admin** 에 의해 작성된 JNDI 정의는 IBM MQ classes for JMS에 의해 사용될 수 없습니다.

참고: IBM MQ Explorer에서 제공하는 IBM MQ classes for Jakarta Messaging 오브젝트에 대한 지원은 없습니다. 해당 JNDI 통합은 IBM MQ classes for JMS 전용입니다.

관련 개념

Jakarta Messaging에서 IBM MQ 클래스를 사용해야 하는 이유는 무엇입니까?

JMS 및 Jakarta Messaging 모델

JMS 및 Jakarta Messaging 모델은 Java 애플리케이션이 메시징 조작을 수행하는 데 사용할 수 있는 인터페이스 세트를 정의합니다. IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 는 Java 메시징 오브젝트가 IBM MQ 개념과 관련되는 방법을 정의하는 메시징 제공자입니다. JMS 및 Jakarta Messaging 스펙은 특정 메시징 오브젝트가 관리 오브젝트가 될 것으로 예상합니다.

IBM MQ 8.0부터 제품은 JMS 1.1의 클래식 API를 유지하면서 단순화된 API를 도입한 JMS 표준의 JMS 2.0 버전을 지원합니다.

JM 3.0 IBM MQ 9.3.0 에서는 Jakarta Messaging 3.0에 대한 지원을 도입했습니다. JMS 2.0 여전히 완전히 지원됩니다. JMS 및 Jakarta Messaging 는 공통적으로 많이 공유하므로 이 주제에서 JMS 에 대한 추가 참조는 둘 다를 참조하는 것으로 간주할 수 있습니다. 필요에 따라 차이점이 강조표시됩니다.

간소화된 API

JMS 2.0 에서는 JMS 1.1의 도메인 특정 및 도메인 독립 인터페이스를 유지하면서 단순화된 API를 도입했습니다. 간소화된 API에서는 메시지를 송신하고 수신하는 데 필요한 오브젝트 수가 줄어들며 다음 인터페이스로 구성됩니다.

ConnectionFactory
ConnectionFactory는 JMS 클라이언트에서 Connection을 작성하는 데 사용하는 관리 대상 오브젝트입니다. 이 인터페이스는 클래식 API에서도 사용합니다.

JMS컨텍스트
이 오브젝트는 클래식 API의 Connection 및 Session 오브젝트를 결합합니다. JMSContext 오브젝트는 다른 JMSContext 오브젝트에서 작성할 수 있으며 기본 연결은 복제됩니다.

JMS 생성자
JMSProducer는 JMSContext를 통해 작성되며 큐나 토픽에 메시지를 송신하는 데 사용합니다. JMSProducer 오브젝트를 사용하면 메시지를 송신하는 데 필요한 오브젝트가 작성됩니다.

JMS 이용자
JMSConsumer는 JMSContext를 통해 작성되며 토픽이나 큐에서 메시지를 송신하는 데 사용합니다.

간소화된 API의 효과는 다음과 같이 여러 가지가 있습니다.

- JMSContext 오브젝트에서 항상 자동으로 기본 연결을 시작합니다.
- JMSProducers와 JMSConsumers는 이제 Message의 getBody 메소드를 사용하여 전체 Message 오브젝트를 가져올 필요 없이 메시지 본문과 직접 작업할 수 있습니다.

- Message 특성은 메시지 컨텐츠인 'body'를 송신하기 전에 메소드 체인을 사용하여 JMSProducer 오브젝트에 설정할 수 있습니다. JMSProducer는 메시지를 송신하는 데 필요한 모든 오브젝트의 작성을 핸들링합니다. JMS 2.0을 사용하여 특성을 설정할 수 있으며 메시지가 다음과 같이 송신됩니다.

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

JMS 2.0 에서는 여러 이용자 간에 메시지를 공유할 수 있는 공유 구독도 도입했습니다. 모든 JMS 1.1 구독은 공유하지 않는 구독으로 처리됩니다.

클래식 API

다음 목록은 클래식 API의 기본 JMS 인터페이스를 요약합니다.

목적지
목적지는 애플리케이션이 메시지를 보내는 위치이거나 애플리케이션이 메시지를 수신하는 소스입니다.

ConnectionFactory
ConnectionFactory 오브젝트는 연결을 위한 구성 특성 세트를 캡슐화합니다. 애플리케이션에서 연결 팩토리를 사용하여 연결을 작성합니다.

연결
Connection 오브젝트는 메시징 서버에 대한 애플리케이션의 활성화된 연결을 캡슐화합니다. 애플리케이션은 연결을 사용하여 세션을 작성합니다.

세션
세션은 메시지 송신 및 수신을 위한 단일 스레드 컨텍스트입니다. 애플리케이션은 세션을 사용하여 메시지, 메시지 생성자 및 메시지 이용자를 작성합니다. 세션은 트랜잭션되거나 트랜잭션되지 않습니다.

메시지
Message 오브젝트는 애플리케이션이 송신하거나 수신하는 메시지를 캡슐화합니다.

MessageProducer
애플리케이션은 메시지 생성자를 사용하여 메시지를 목적지에 송신합니다.

MessageConsumer
애플리케이션은 메시지 이용자를 사용하여 목적지로 송신된 메시지를 수신합니다.

138 페이지의 그림 53에서는 이러한 오브젝트와 해당 관계를 표시합니다.

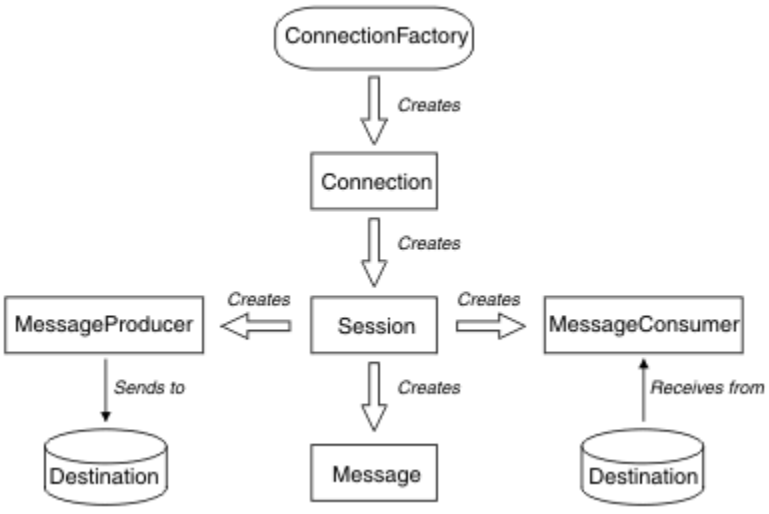


그림 *53. JMS* 오브젝트 및 해당 관계

다이어그램은 기본 인터페이스인 연결 팩토리, 연결, 세션, 메시지 생성자, 메시지 이용자, 메시지, 목적지를 보여줍니다. 애플리케이션은 연결 팩토리를 사용하여 연결을 작성하고 연결을 사용하여 세션을 작성합니다. 그런 다

음 애플리케이션은 세션을 사용하여 메시지, 메시지 생성자 및 메시지 이용자를 작성할 수 있습니다. 애플리케이션은 메시지 생성자를 사용하여 메시지를 목적지로 송신하고 메시지 이용자를 사용하여 목적지로 송신된 메시지를 수신합니다.

Destination, ConnectionFactory 또는 Connection 오브젝트는 멀티스레드 애플리케이션의 여러 다른 스레드에서 동시에 사용할 수 있지만, Session, MessageProducer 또는 MessageConsumer 오브젝트는 여러 다른 스레드에서 동시에 사용할 수 없습니다. Session, MessageProducer 또는 MessageConsumer 오브젝트를 동시에 사용하지 않는 가장 간단한 방법은 각 스레드의 개별 Session 오브젝트를 작성하는 것입니다.

JMS에서는 다음 두 스타일의 메시징을 지원합니다.

- 포인트-투-포인트 메시징
- 발행/구독 메시징

이러한 스타일의 메시징은 메시징 도메인이라고도 하며 한 애플리케이션에 두 스타일의 메시징을 모두 결합할 수 있습니다. 포인트-투-포인트 도메인에서는 목적지가 큐이며 발행/구독 도메인에서는 목적지가 토픽입니다.

JMS 1.1이전의 JMS 버전에서는 지점간 도메인에 대한 프로그래밍이 한 세트의 인터페이스 및 메소드를 사용하고 발행/구독 도메인에 대한 프로그래밍이 다른 세트를 사용합니다. 두 세트는 비슷하지만 분리되어 있습니다. JMS 1.1에서 두 메시징 도메인을 모두 지원하는 공통 인터페이스와 메소드 세트를 사용할 수 있습니다. 공통 인터페이스에서는 각 메시징 도메인의 도메인 독립적 보기를 제공합니다. 139 페이지의 표 17에서는 JMS 도메인 독립적 인터페이스와 대응하는 도메인 특정 인터페이스를 나열합니다.

표 *17. JMS* 도메인 독립적 인터페이스와 해당하는 도메인 특정 인터페이스

도메인 독립적 인터페이스	**포인트-투-포인트 도메인의 도메인 특정 인터페이스**	**발행/구독 도메인의 도메인 특정 인터페이스**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
연결	QueueConnection	TopicConnection
목적지	큐	토픽
세션	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 2.0 IBM MQ classes for JMS 2.0 는 이전 JMS 1.1 도메인 특정 인터페이스와 JMS 2.0의 단순화된 API를 모두 지원합니다. 따라서 IBM MQ classes for JMS 2.0 는 기존 애플리케이션에서 새 기능 개발을 포함하여 기존 애플리케이션을 유지보수하는 데 사용할 수 있습니다.

JM 3.0 IBM MQ classes for Jakarta Messaging 3.0 는 동일한 인터페이스의 Jakarta Messaging 버전을 지원하며 새 애플리케이션 개발에 권장됩니다.

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging에서 JMS 오브젝트는 다음과 같은 방식으로 IBM MQ 개념과 관련됩니다.

- Connection 오브젝트에는 연결을 작성하는 데 사용한 연결 팩토리의 특성에서 파생된 특성이 있습니다. 이러한 특성은 애플리케이션이 큐 관리자에 연결하는 방식을 제어합니다. 이러한 특성의 예로는 큐 관리자의 이름이 있으며, 클라이언트 모드에서 큐 관리자에 연결하는 애플리케이션의 경우에는 큐 관리자가 실행 중인 시스템의 호스트 이름 또는 IP 주소입니다.
- Session 오브젝트가 IBM MQ 연결 핸들을 캡슐화하므로, 세션의 트랜잭션 범위를 정의합니다.
- MessageProducer 오브젝트와 MessageConsumer 오브젝트는 각각 IBM MQ 오브젝트 핸들을 캡슐화합니다.

IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging를 사용하는 경우 IBM MQ 의 모든 일반 규칙이 적용됩니다. 특히, 애플리케이션에서 리모트 큐에 메시지를 송신할 수 있지만, 애플리케이션이 연결된 큐 관리자가 소유한 큐에서만 메시지를 수신할 수 있다는 점에 유의하십시오.

JMS 스펙에서는 ConnectionFactory 및 Destination 오브젝트가 관리 대상 오브젝트여야 합니다. 관리자가 중앙 저장소에서 관리 대상 오브젝트를 작성 및 유지보수하며 JMS 애플리케이션은 JNDI(Java Naming and Directory Interface)를 사용하여 이러한 오브젝트를 검색합니다.

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging에서 Destination 인터페이스의 구현은 Queue및 Topic의 추상 수퍼클래스이므로 Destination의 인스턴스는 Queue 오브젝트 또는 Topic 오브젝트입니다. 도메인 독립적 인터페이스에서는 큐나 토픽을 목적지로 처리합니다. MessageProducer 또는 MessageConsumer 오브젝트의 메시징 도메인은 목적지가 큐인지 아니면 토픽인지에 따라 결정됩니다.

따라서 IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 에서 다음 유형의 오브젝트는 관리 오브젝트일 수 있습니다.

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- 큐
- 토픽
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory

IBM MQ classes for JMS/Jakarta Messaging 아키텍처

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 에는 계층화된 아키텍처가 있습니다. 코드의 맨 위 계층은 IBM Java 메시징 제공자가 사용할 수 있는 공통 계층입니다.

JM 3.0 IBM MQ 9.3.0 에서는 Jakarta Messaging 3.0에 대한 지원을 도입했습니다. JMS 2.0 여전히 완전히 지원됩니다.

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 에는 141 페이지의 그림 54다이어그램에 표시된 대로 계층화된 아키텍처가 있습니다. 코드의 맨 위 계층은 IBM JMS 또는 Jakarta Messaging제공자가 사용할 수 있는 공통 계층입니다. 애플리케이션이 JMS 또는 Jakarta Messaging메소드를 호출할 때 메시징 시스템에 특정하지 않은 호출의 처리는 공통 계층에 의해 수행되며, 이는 호출에 대한 일관된 응답도 제공합니다. 메시징 시스템에 특정적인 호출의 모든 처리는 하위 계층에 위임됩니다. 다음 다이어그램에서 두 개의 추가 메시징 제공자(메시징 제공자 A 및 메시징 제공자 B)와 함께 IBM MQ 메시징 제공자가 하위 계층에 표시됩니다.

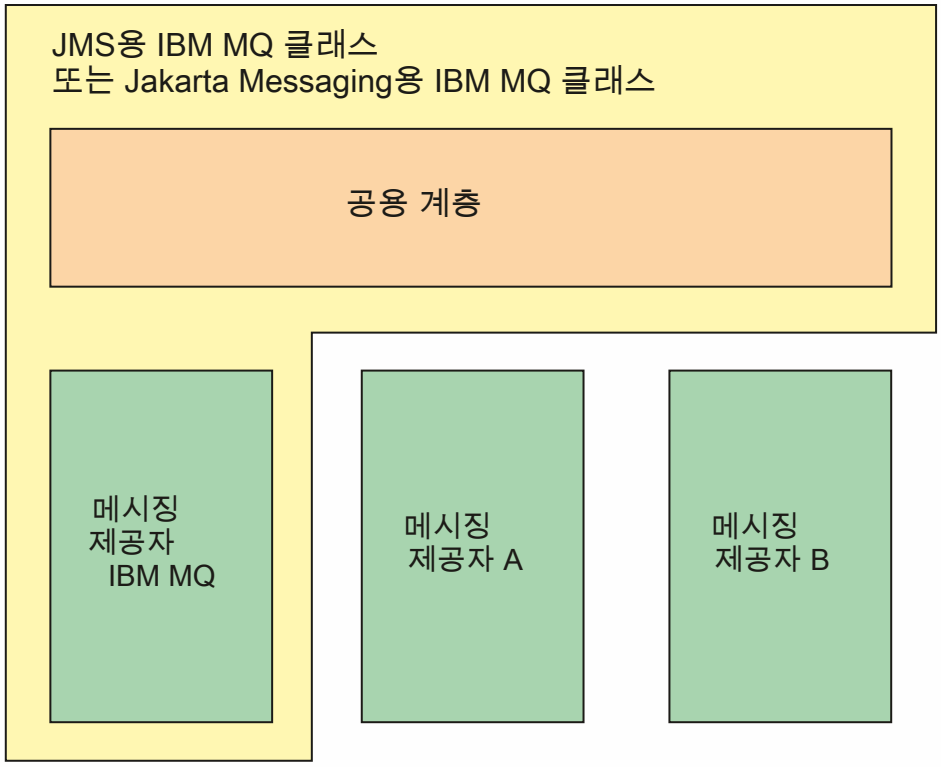


그림 *54. IBM JMS* 및 *Jakarta Messaging* 제공자의 계층화된 아키텍처

계층 아키텍처는 다음 목표를 달성합니다.

- 다양한 IBM JMS 및 Jakarta Messaging 제공자의 동작 일관성을 개선하기 위해
- 두 IBM 메시징 시스템 사이에서 브릿지 애플리케이션을 더 쉽게 작성
- 하나의 IBM JMS 또는 Jakarta Messaging 제공자에서 다른 제공자로 애플리케이션을 더 쉽게 이식하기 위해

관련 태스크

JMS/Jakarta Messaging에 IBM MQ 클래스 사용

관리 오브젝트 지원

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 는 관리 오브젝트의 사용을 지원합니다.

JM 3.0 IBM MQ 9.3.0부터 새 애플리케이션 개발을 위해 Jakarta Messaging 3.0 가 지원됩니다. IBM MQ 9.3.0 이상은 기존 애플리케이션에 대한 JMS 2.0 를 계속 지원합니다. 동일한 애플리케이션에서 Jakarta Messaging 3.0 API및 JMS 2.0 API를 모두 사용하는 것은 지원되지 않습니다. 자세한 정보는 JMS/Jakarta Messaging에 대한 IBM MQ 클래스 사용을 참조하십시오.

JMS 또는 IBM MQ classes for Jakarta Messaging 애플리케이션 내의 로직 플로우는 ConnectionFactory 및 Destination 오브젝트로 시작합니다. 애플리케이션은 Connection 오브젝트를 작성하기 위해 ConnectionFactory 오브젝트를 사용하며, 이는 애플리케이션에서 메시징 서버로의 활성 연결을 나타냅니다. 애플리케이션은 Session 오브젝트를 작성하기 위해 Connection 오브젝트를 사용하며, 이는 메시지를 생성하고 사용하기 위한 단일 스레드 컨텍스트입니다. 그런 다음 지정된 목적지로 메시지를 송신하는 데 사용하는 이 애플리케이션은 Session 오브젝트와 Destination 오브젝트를 사용하여 MessageProducer 오브젝트를 작성할 수 있습니다. 목적지는 메시징 시스템의 토픽 또는 큐 중 하나이며, Destination 오브젝트에 의해 캡슐화됩니다. 이 애플리케이션은 또한 Session 오브젝트와 Destination 오브젝트를 사용하여 MessageConsumer 오브젝트를 작성하며, 이 애플리케이션은 지정된 목적지로 송신된 메시지를 수신하기 위해 사용됩니다.

JMS 및 Jakarta Messaging 스펙은 ConnectionFactory 및 Destination 오브젝트가 관리 오브젝트가 될 것으로 예상합니다. 관리자는 중앙 저장소에서 관리 오브젝트를 작성하고 유지보수하며 JMS 또는 Jakarta Messaging 애플리케이션은 Java Naming Directory Interface (JNDI) 를 사용하여 이러한 오브젝트를 검색합니다. 관리 오

브젝트의 저장소는 단순 파일에서 LDAP (Lightweight Directory Access Protocol) 디렉토리까지 다양할 수 있습니다.

IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 는 관리 오브젝트의 사용을 지원합니다. 애플리케이션은 IBM MQ특정 정보를 애플리케이션 자체에 하드 코딩하지 않고 IBM MQ 를 통해 노출되는 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 의 모든 기능을 사용할 수 있습니다. 이러한 배열은 애플리케이션이 기반의 IBM MQ 구성에서 상당히 독립적이게 합니다.

이러한 독립성을 얻기 위해 애플리케이션은 JNDI를 사용하여 관리 오브젝트로 저장된 연결 팩토리 및 대상을 검색하고 `javax.jms` (JMS 2.0) 또는 `jakarta.jms` (Jakarta Messaging 3.0) 패키지에 정의된 인터페이스만 사용하여 메시징 조작을 수행할 수 있습니다.

JMS 2.0 JMS 2.0의 경우 관리자는 IBM MQ JMS 관리 도구 **JMSAdmin**또는 IBM MQ Explorer를 사용하여 중앙 저장소에서 관리 오브젝트를 작성하고 유지보수할 수 있습니다.

JM 3.0 Jakarta Messaging 3.0의 경우 IBM MQ Explorer를 사용하여 JNDI를 관리할 수 없습니다. JNDI 관리는 **JMSAdmin**의 Jakarta Messaging 3.0 변형인 **JMS30Admin**에 의해 지원됩니다.

Application Server는 일반적으로 관리 오브젝트에 대한 자체 저장소 및 오브젝트를 작성하고 유지보수하기 위한 자체 도구를 제공합니다. 따라서 Java EE JM 3.0 또는 Jakarta EE 애플리케이션은 JNDI 를 사용하여 애플리케이션 서버 저장소 또는 중앙 저장소에서 관리 오브젝트를 검색할 수 있습니다.

관련 태스크

JMS및 Jakarta Messaging 자원 구성

Java EE 및 Jakarta EE 플랫폼에서 지원되는 통신 유형

Java EE 및 Jakarta EE 플랫폼에서 IBM MQ classes for JMS 및 IBM MQ classes for Jakarta Messaging 는 애플리케이션의 컴포넌트와 IBM MQ 큐 관리자 간에 두 가지 유형의 통신을 지원합니다.

JM 3.0 IBM MQ 9.3.0 에서는 Jakarta Messaging 3.0에 대한 지원을 도입했습니다. JMS 2.0 여전히 완전히 지원됩니다. JMS 및 Jakarta Messaging 는 공통적으로 많이 공유하므로 이 주제에서 JMS 에 대한 추가 참조는 둘 다를 참조하는 것으로 간주할 수 있습니다. 필요에 따라 차이점이 강조표시됩니다.

다음 두 가지 유형의 통신이 애플리케이션의 컴포넌트와 IBM MQ 큐 관리자 사이에서 지원됩니다.

- 아웃바운드 통신
- 인바운드 통신

아웃바운드 통신

애플리케이션 컴포넌트는 JMS 또는 Jakarta Messaging API를 직접 사용하여 큐 관리자에 대한 연결을 작성한 후 메시지를 송수신합니다.

예를 들어, 애플리케이션 컴포넌트는 애플리케이션 클라이언트, 서블릿, JSP(JavaServer Page), Enterprise Java Bean(EJB) 또는 MDB(Message Driven Bean)일 수 있습니다. 이러한 유형의 통신에서 애플리케이션 서버 컨테이너는 메시징 조작(연결 풀링 및 스레드 관리 등)의 지원에서 낮은 레벨의 기능을 제공합니다.

인바운드 통신

인바운드 통신의 경우 목적지에 도착하는 메시지는 MDB로 전달되고 MDB가 메시지를 처리합니다.

Java EE JM 3.0 및 Jakarta EE 애플리케이션은 MDB를 사용하여 메시지를 비동기식으로 처리합니다. MDB는 JMS 메시지 리스너로서 작동하고 onMessage() 메소드에 의해 구현됩니다. 이 메소드는 메시지가 처리되는 방법을 정의합니다. MDB는 어플리케이션 서버의 EJB 컨테이너에 배치됩니다. MDB가 구성되는 정확한 방법은 사용 중인 애플리케이션 서버에 따라 결정되지만 구성 정보는 연결할 큐 관리자, 큐 관리자에 연결하는 방법, 메시지를 모니터할 목적지, MDB의 트랜잭션 작동을 지정해야 합니다. 그러면 이 정보가 EJB 컨테이너에 의해 사용됩니다. MDB의 선택 기준을 충족하는 메시지가 지정된 대상에 도착하면 EJB 컨테이너는 IBM MQ classes for JMS 또는 IBM MQ classes for Jakarta Messaging 를 사용하여 큐 관리자에서 메시지를 검색한 후 onMessage() 메소드를 호출하여 메시지를 MDB에 전달합니다.

IBM MQ classes for Java와의 관계

IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging 및 IBM MQ classes for JMS 는 MQI에 대한 공용 Java 인터페이스를 사용하는 피어입니다.

143 페이지의 그림 55 에서는 IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging 및 IBM MQ classes for Java간의 관계를 보여줍니다.

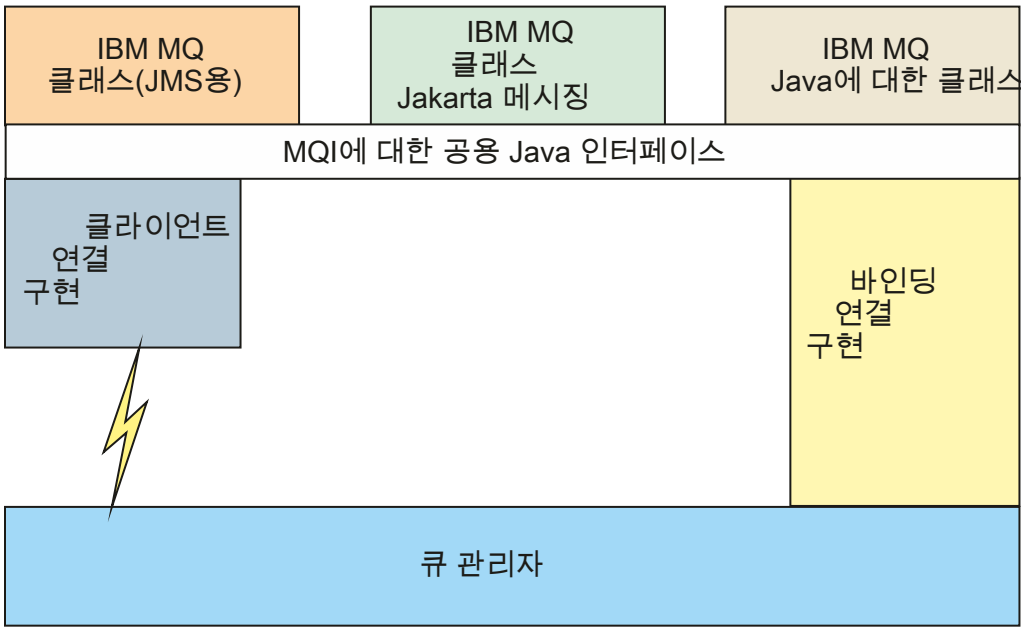


그림 *55. IBM MQ classes for JMS, IBM MQ classes for Jakarta Messaging* 및 *IBM MQ classes for Java* 간의 관계

일반적으로 Java 프로그램은 IBM MQ - IBM MQ classes for Java, IBM MQ classes for Jakarta Messaging 또는 IBM MQ classes for JMS와 인터페이스하기 위해 하나의 인터페이스만 사용해야 합니다. 인터페이스 혼합은 한 가지 예외를 제외하고 지원되지 않습니다. IBM WebSphere MQ 7.0이전 릴리스와의 호환성을 유지하기 위해, 채널 엑시트 클래스가 IBM MQ classes for JMS에서 호출되는 경우에도 Java 로 작성된 채널 엑시트 클래스는 여전히 IBM MQ classes for Java 인터페이스를 사용할 수 있습니다. 그러나 IBM MQ classes for Java 인터페이스를 사용하는 것은 애플리케이션이 여전히 다음 중 하나에 종속됨을 의미합니다.

- JMS 2.0 IBM MQ classes for Java JAR 파일, `com.ibm.mq.jar`. 사용자 클래스 경로에서 `com.ibm.mq.jar`를 원하지 않을 경우, 대신 `com.ibm.mq.exits` 패키지의 인터페이스 세트를 사용할 수 있습니다.
- JM 3.0 IBM MQ classes for Jakarta Messaging와 상호 운용할 때 `com.ibm.mq.jakarta.client.jar`를 사용합니다.

관련 개념

Jakarta Messaging에서 IBM MQ 클래스를 사용해야 하는 이유는 무엇입니까?
JMS용 IBM MQ 클래스를 사용해야 하는 이유는 무엇입니까?
Java용 IBM MQ 클래스를 사용해야 하는 이유는 무엇입니까?

IBM MQ 메시징 제공자

IBM MQ 메시징 제공자에는 3개의 조작 모드 즉, 정상 모드, 제한적 정상 모드, 마이그레이션 모드가 있습니다.

IBM MQ 메시징 제공자에는 3개의 조작 모드가 있습니다.

- IBM MQ 메시징 제공자 정상 모드
- 제한이 있는 IBM MQ 메시징 제공자 정상 모드
- IBM MQ 메시징 제공자 마이그레이션 모드

IBM MQ 메시징 제공자 정상 모드는 IBM MQ 큐 관리자의 모든 기능을 사용하여 JMS를 구현합니다. 이 모드는 JMS 2.0 JM 3.0 또는 Jakarta Messaging 3.0 API및 기능을 사용하도록 최적화되어 있습니다.

다음과 같은 경우

- 클라이언트는 **ConnectionFactory**에서 제공자 버전 6을 지정하며, 클라이언트는 IBM WebSphere MQ 6.0에서 제공되는 클라이언트와 호환 가능한 방식으로 작동합니다. JMS 1.1 및 JMS 2인터페이스만 지원되지만 일부 JMS 2기능 (예: 공유 구독, 전달 지연 및 비동기 송신) 은 사용 불가능합니다. 연결 공유가 없습니다.
- 클라이언트는 **ConnectionFactory**에서 제공자 버전 7을 지정합니다. JMS 1.1 및 JMS 2인터페이스 모두 완전히 지원됩니다.
- 제공자 버전이 지정되지 않았습니다. 제공자 버전 7과 연결하려고 시도합니다. 이에 실패하면 제공자 버전 6으로 추가 시도가 이루어집니다.

IBM MQ Enterprise Transport를 사용하여 IBM Integration Bus 에 연결하려면 마이그레이션 모드를 사용하십시오. IBM MQ Real-Time Transport를 사용하는 경우에는 연결 팩토리 오브젝트에서 명시적으로 특성을 선택했기 때문에 마이그레이션 모드가 자동으로 선택됩니다. IBM MQ Enterprise Transport를 사용하는 IBM Integration Bus 에 대한 연결은 JMS **PROVIDERVERSION** 특성 구성에 설명된 모드 선택에 대한 일반 규칙을 따릅니다.

관련 태스크

JMS 자원 구성

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 145
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 146
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 148
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 150
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 194
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 207
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 211
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 232
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 248
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 254
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 259
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 222
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 257
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 146 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

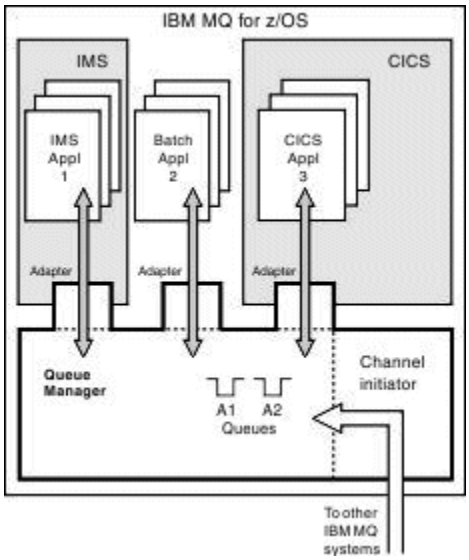


Figure 56. Overview of IBM MQ for z/OS

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 147 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.

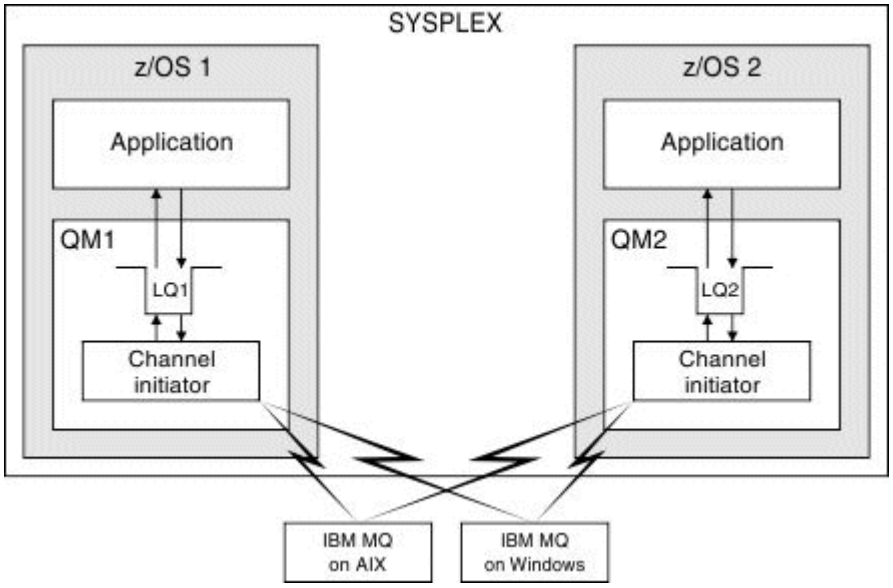


Figure 57. Communication between queue managers

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 211.

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 222.

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 232.

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 248.

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 254.

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 257.

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 261.

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 151 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.

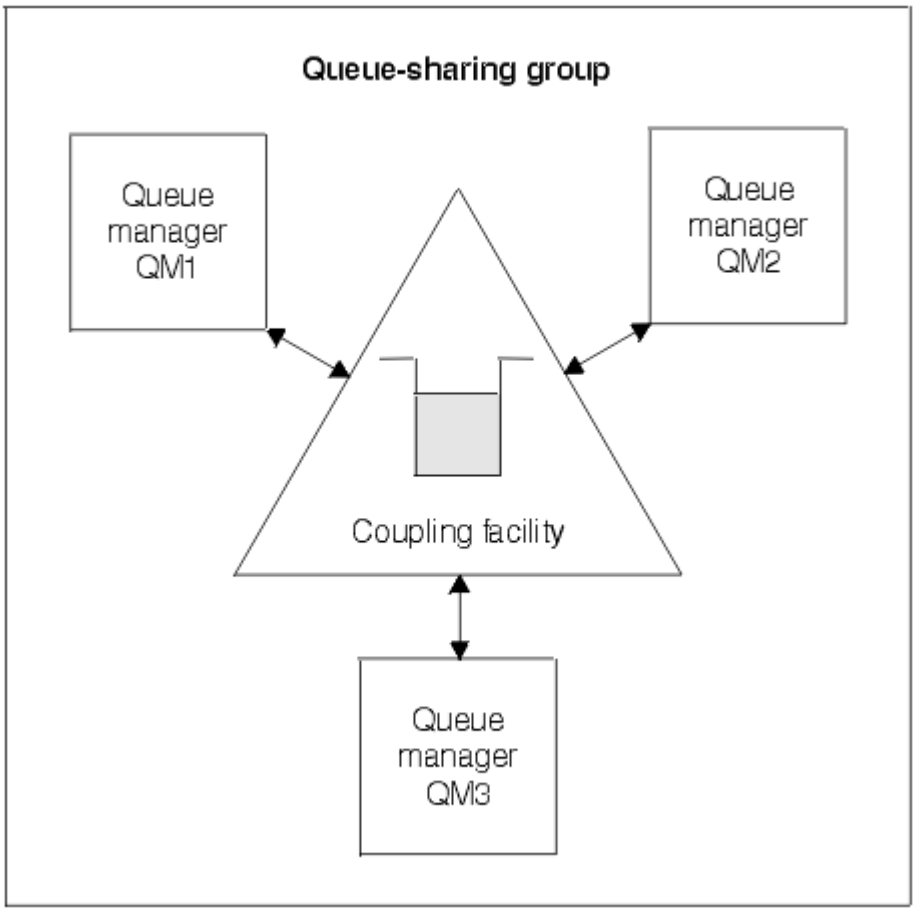


Figure 58. A queue sharing group

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 152 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.

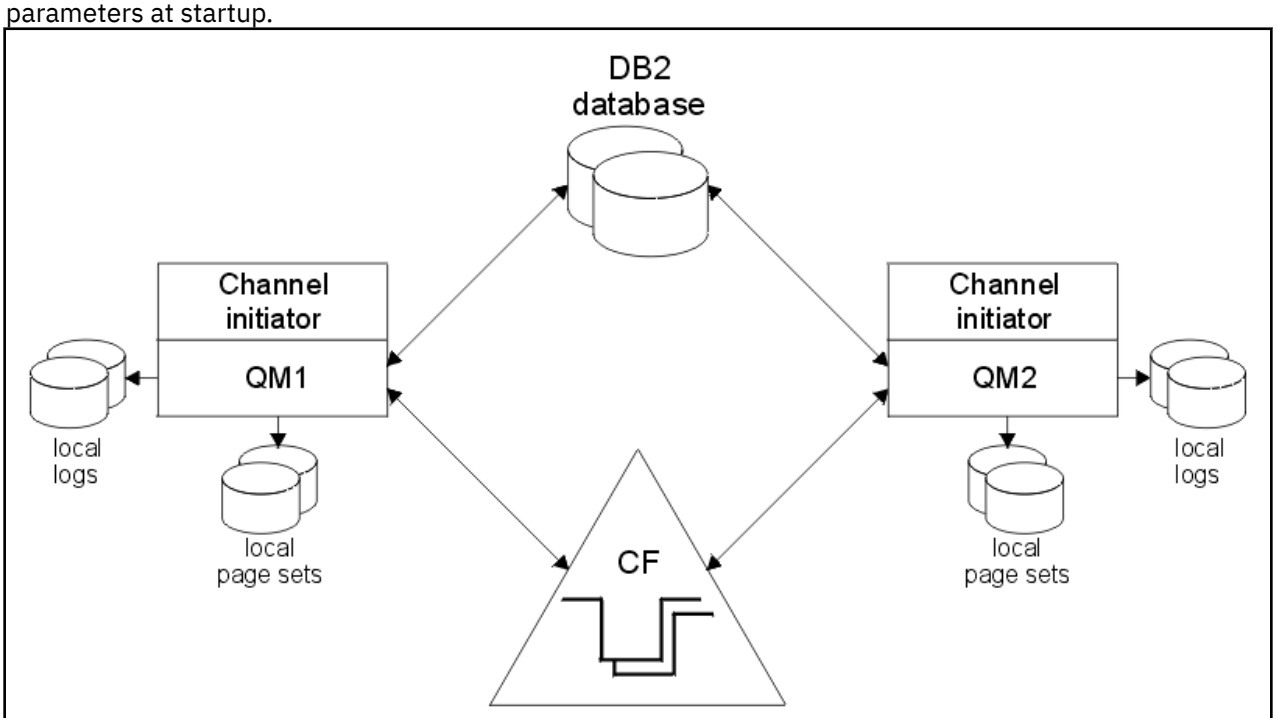


Figure 59. The components of queue managers in a queue sharing group

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 153
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 169
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 189
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 192
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 193
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 155 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 171 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 155
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 157
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 168

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.
- 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.

[1] `(Hits / (Hits+Misses))* 100`

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 170.

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 189.

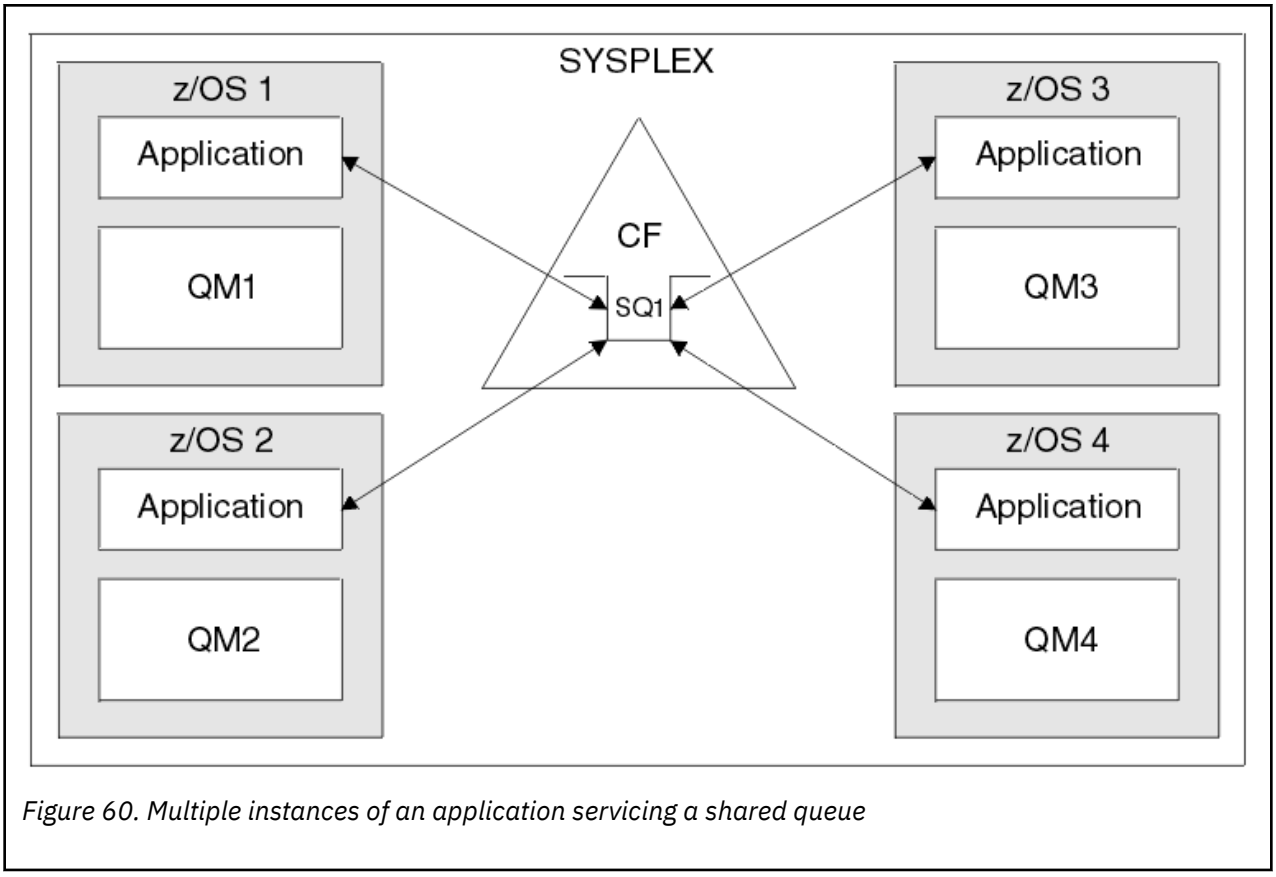


Figure 60. Multiple instances of an application servicing a shared queue

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 177
- "Improved performance - basic configuration" on page 183

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 172 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 178.

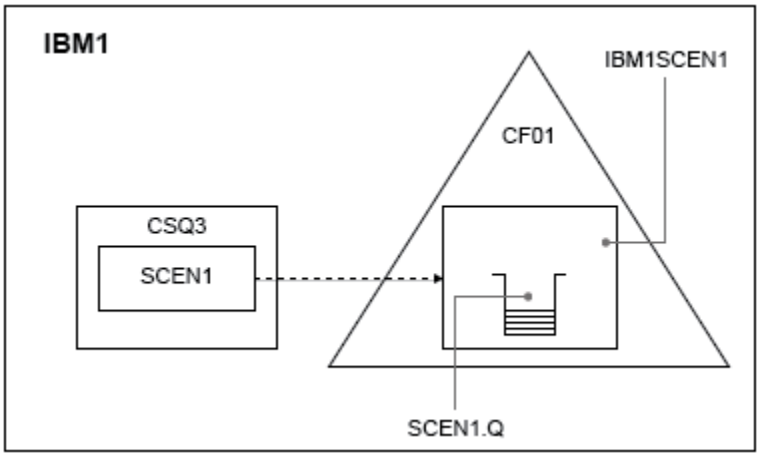


Figure 61. Basic configuration

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 171

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 177. 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 180.

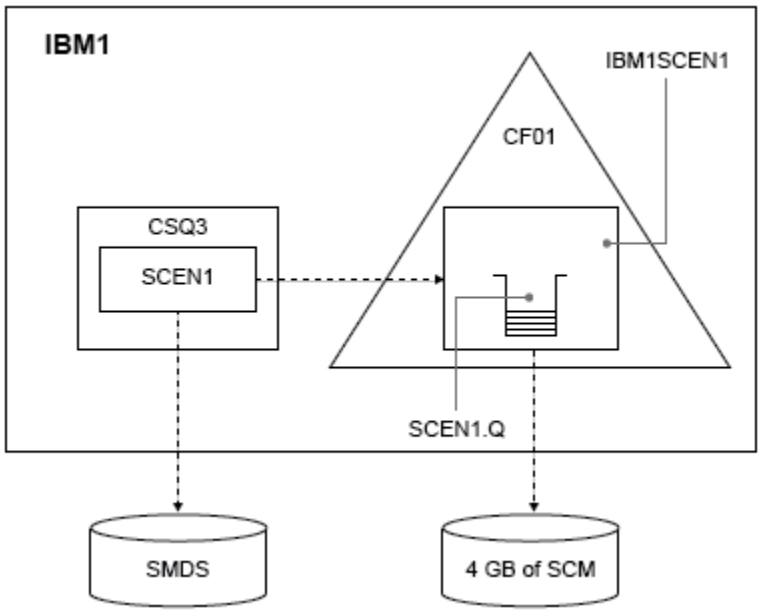


Figure 62. Configuration adding SMDS and SCM for emergency storage

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 182 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 184.

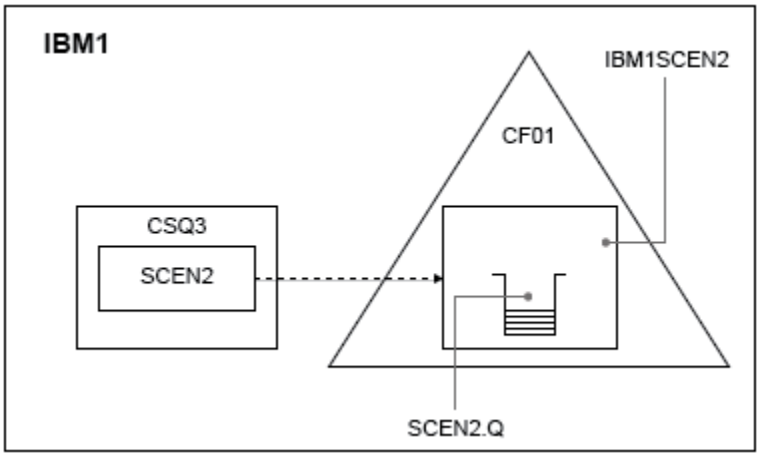


Figure 63. Basic configuration

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 187.

Related concepts

“Use of storage class memory with shared queues” on page 171

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 183. The scenario describes the addition of SCM to the initial structure.

This final configuration is illustrated in Figure 64 on page 187.

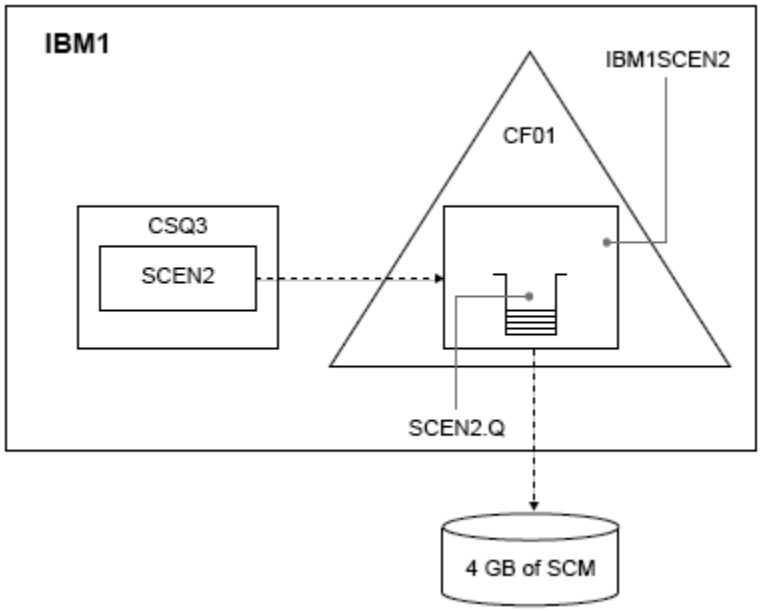


Figure 64. Configuration adding SCM for improved performance

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 183:
  - 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 183.

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 189 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.

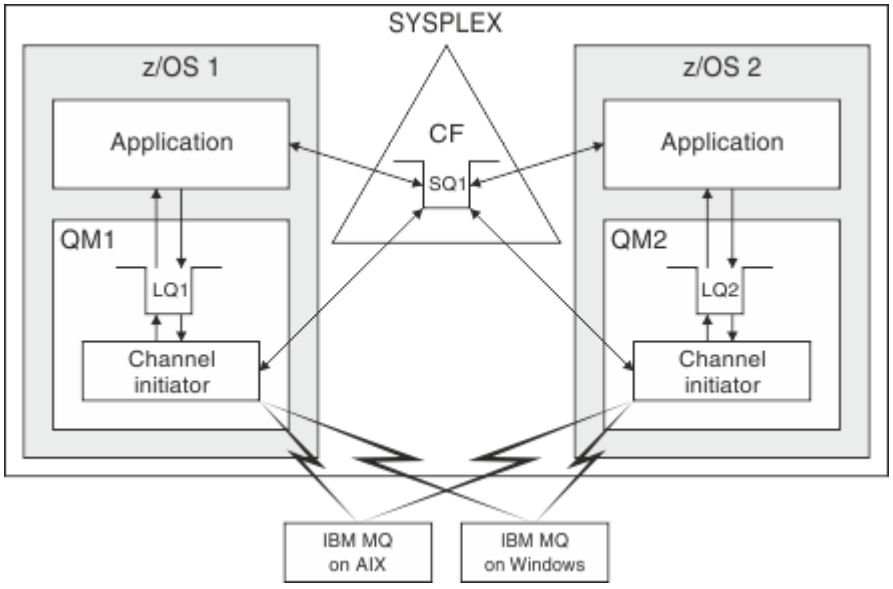


Figure 65. Distributed queuing and queue sharing groups

Related concepts

"Shared channels" on page 190
Use this topic to understand the concepts of shared channels and their use with IBM MQ for z/OS.

"Intra-group queuing" on page 194
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 191
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 192 shows how members of a cluster can access a shared queue through any member of the queue sharing group.

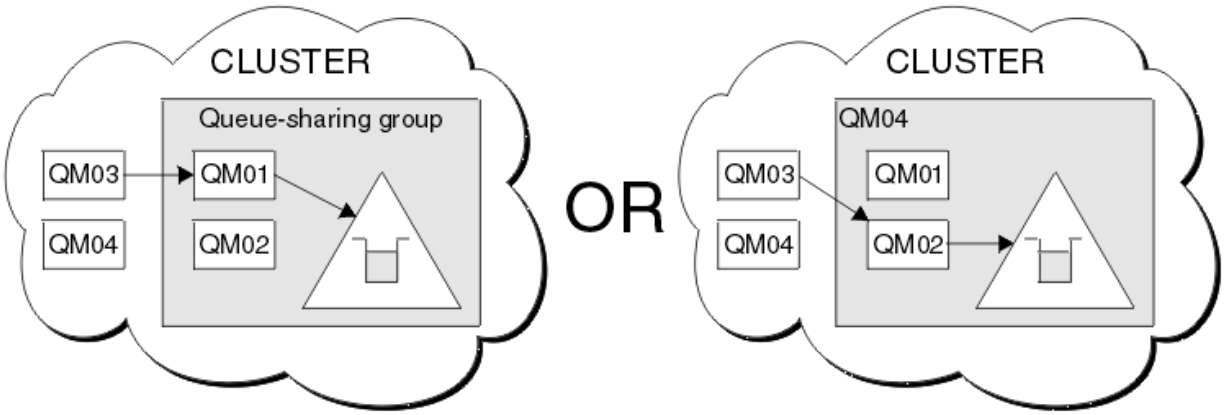


Figure 66. A queue sharing group as part of a cluster

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 232
Queue sharing group security	"Security concepts in IBM MQ for z/OS" on page 248
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

Table 19. Where to find more information about shared queues and queue sharing groups (continued)	
Topic	**Where to look**
Planning your Db2 environment	Planning your Db2 environment
Setting up your shared queues System parameters	"Shared queues and queue sharing groups" on page 150
Utility programs Migrating queues	IBM MQ utilities on z/OS reference
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 150.

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.

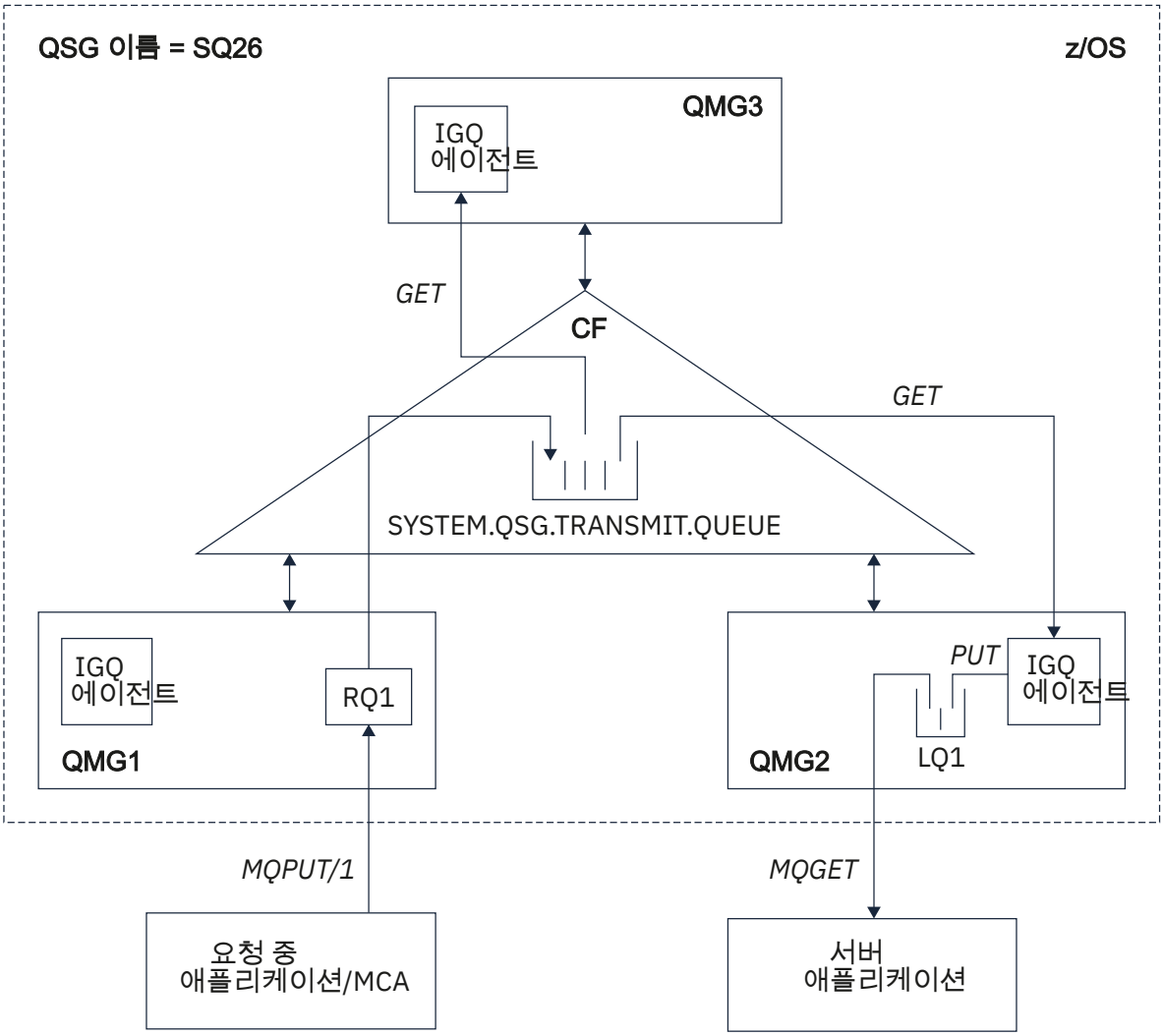


Figure 67. An example of intra-group queuing

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.

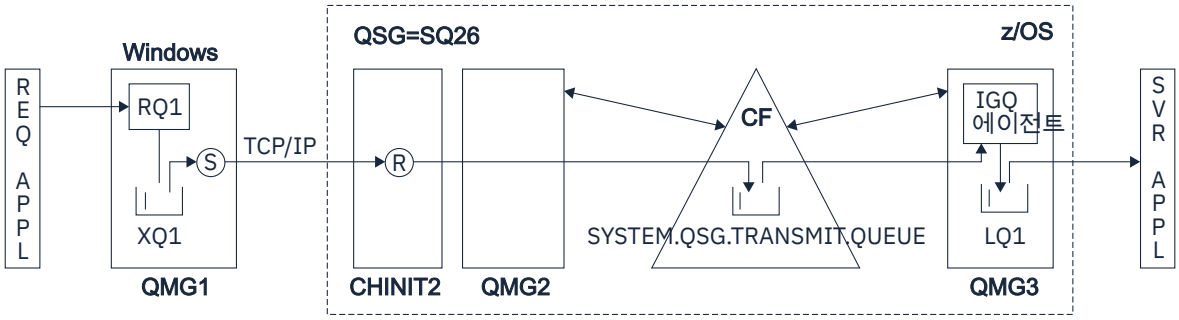


Figure 68. An example of migration support

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 206 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 198 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 206 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 198 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 194 describes the typical configuration.

Related concepts

"Distributed queuing with intra-group queuing (multiple delivery paths)" on page 199
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 201
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 203
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.

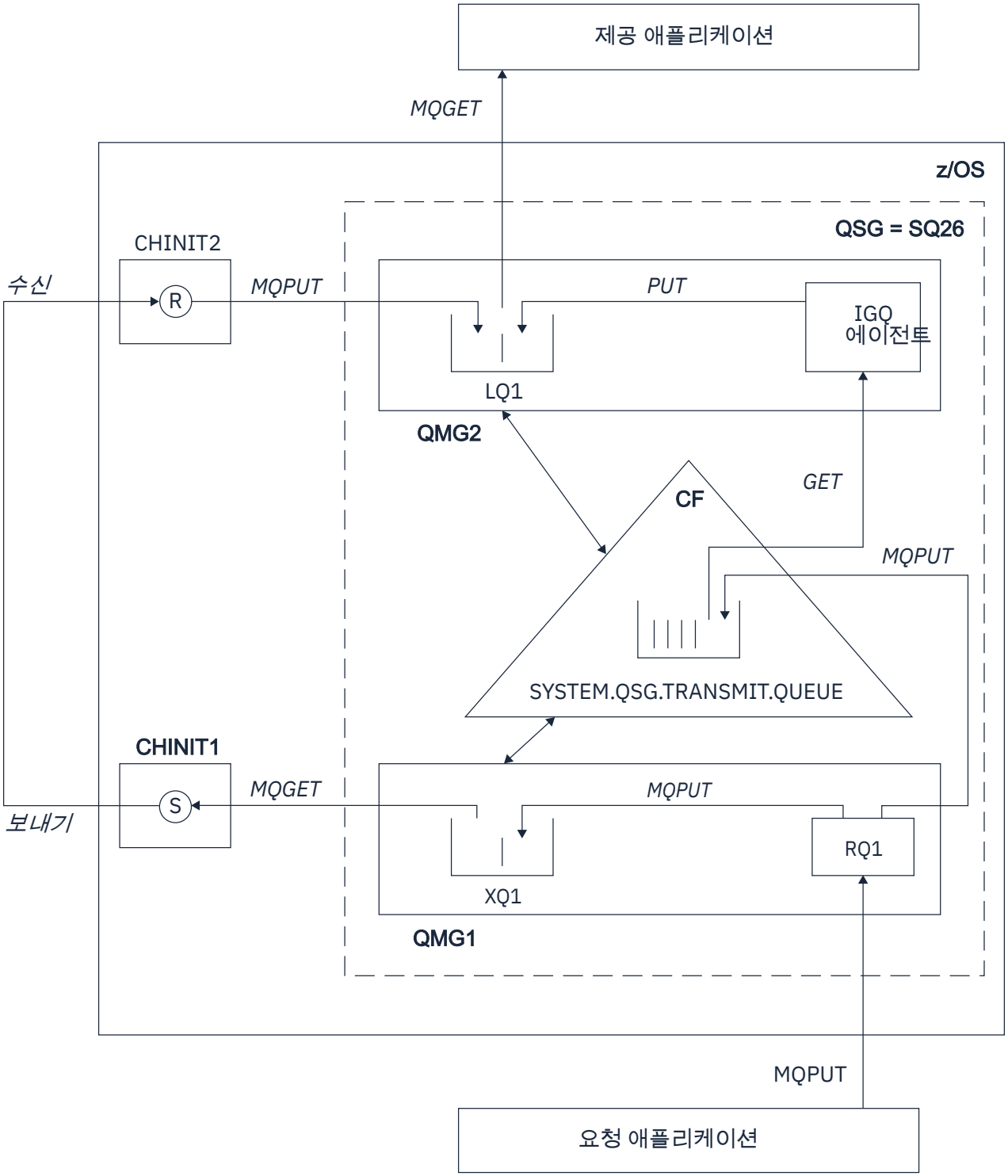


Figure 69. An example configuration

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).

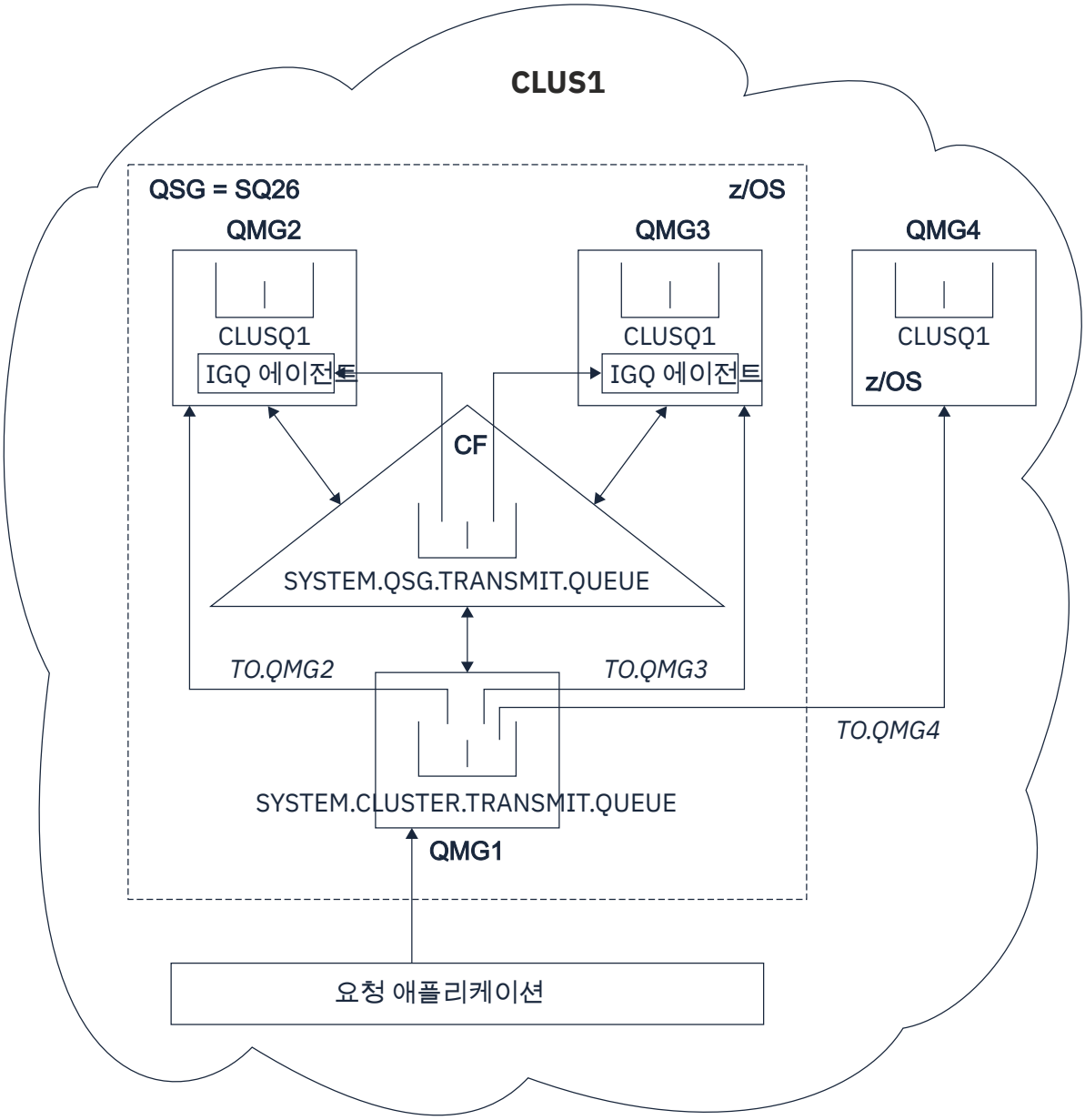


Figure 70. An example of clustering with intra-group queuing

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 199.

Clustering with intra-group queuing is described in "Clustering with intra-group queuing (multiple delivery paths)" on page 201.

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 204
This topic describes what happens to undelivered and unprocessed messages on the SYSTEM.QSG.TRANSMIT.QUEUE.

“Report messages - Intra Group Queuing” on page 205
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 206.
- 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 206.
- 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 207
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 208
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 210
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 211
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 150, 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 263).

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 209, 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.

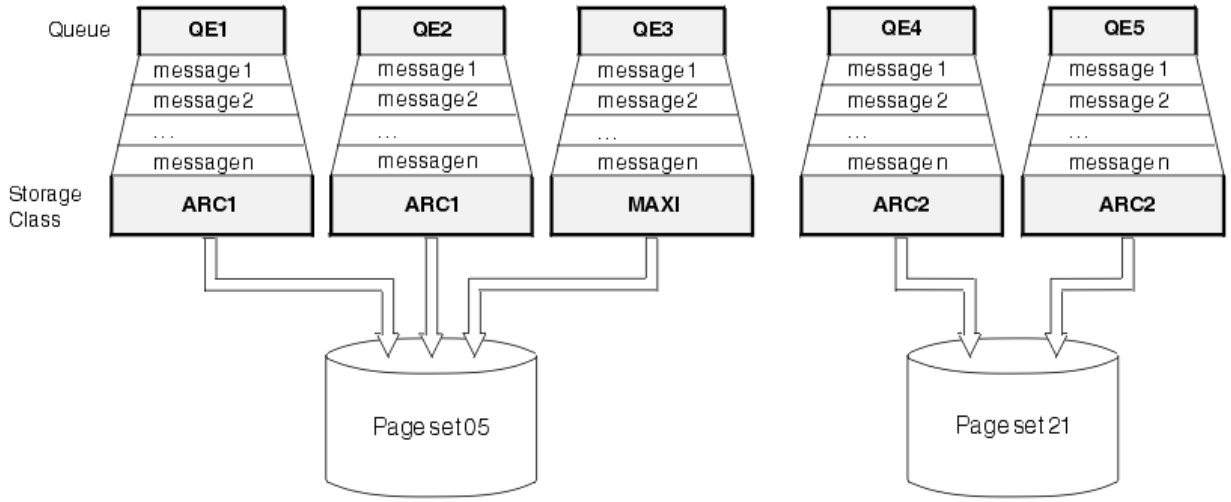


Figure 71. Mapping queues to page sets through storage classes

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 210, 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 210 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.

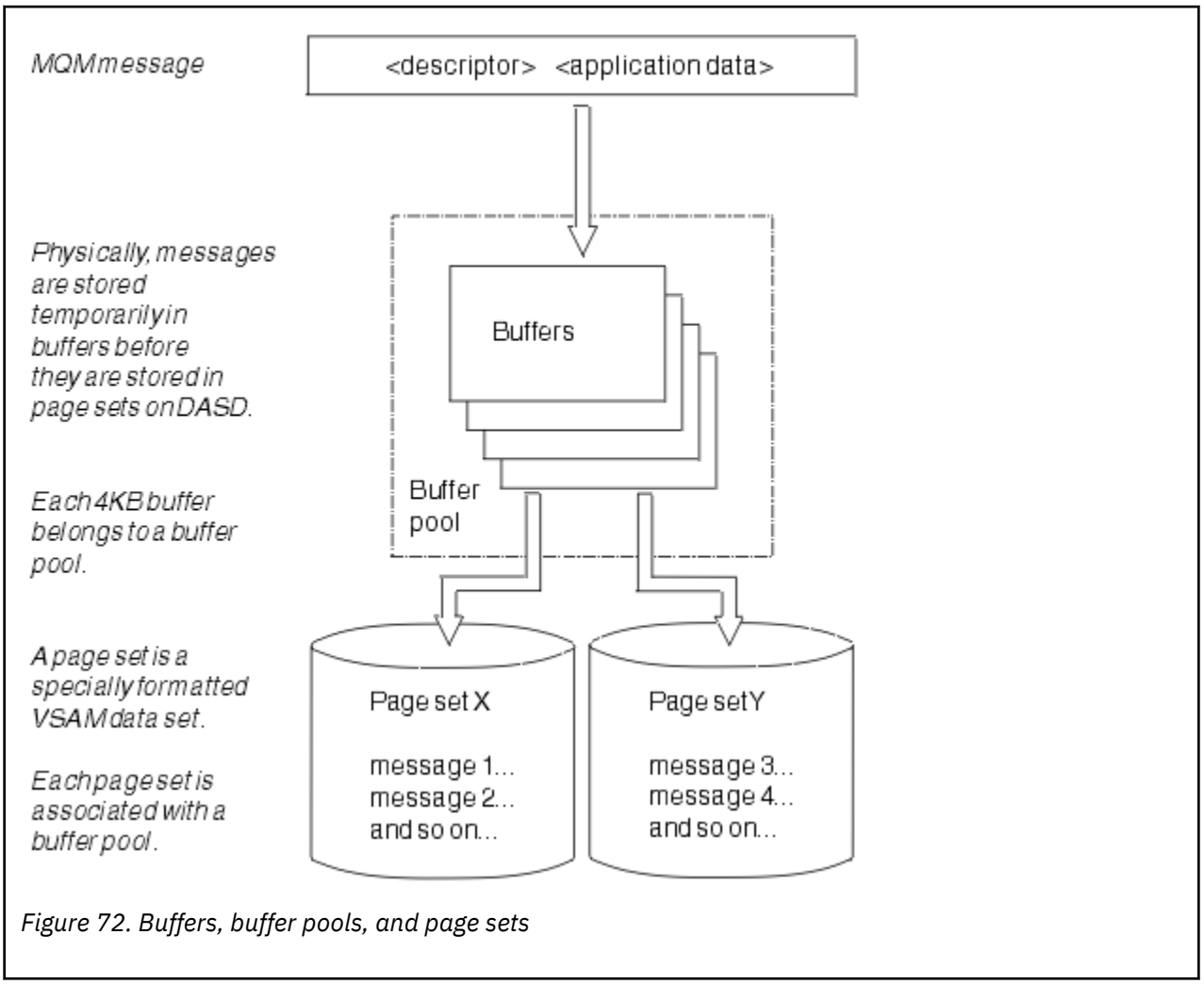


Figure 72. Buffers, buffer pools, and page sets

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 212
- "How the log is structured" on page 216
- "How the IBM MQ for z/OS logs are written" on page 216
- "Larger log Relative Byte Address" on page 219

- "The bootstrap data set" on page 220

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 217 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.

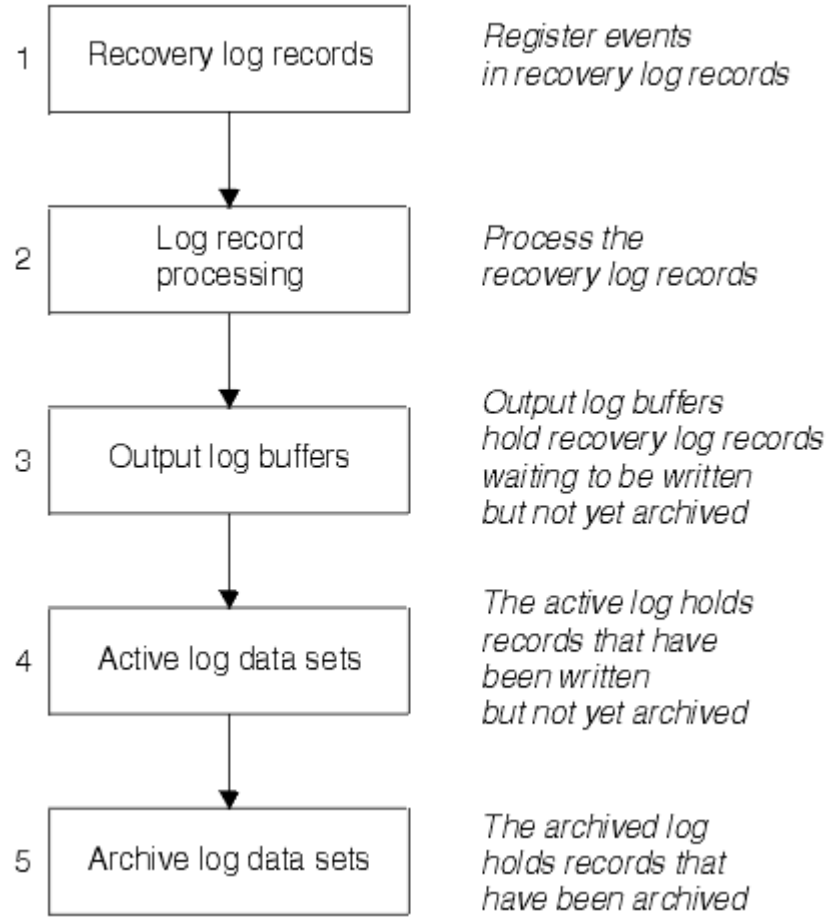


Figure 73. The logging process

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 218
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 218.

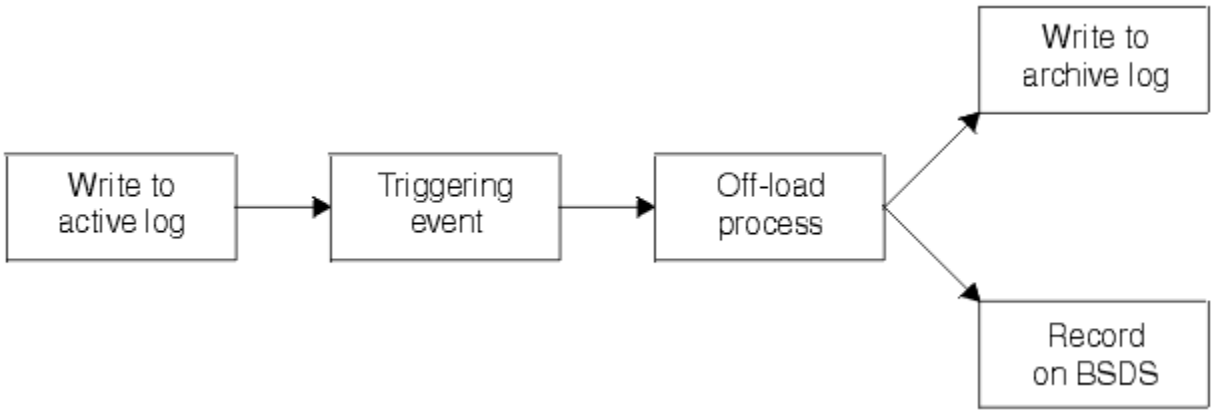


Figure 74. The offloading process

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 IBM MQ for z/OS 관리. 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 IBM MQ for z/OS 관리.

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 219 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 222
- "Tuning your queue manager on IBM MQ for z/OS" on page 227
- "Sample definitions supplied with IBM MQ for z/OS" on page 228

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 226

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 - z/OS 용 IBM MQ 용량 계획 및 튜닝 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 233
- "How consistency is maintained in IBM MQ for z/OS" on page 234
- "What happens during termination in IBM MQ for z/OS" on page 236
- "What happens during restart and recovery in IBM MQ for z/OS" on page 237
- "How in-doubt units of recovery are resolved" on page 239
- "Shared queue recovery" on page 242

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.

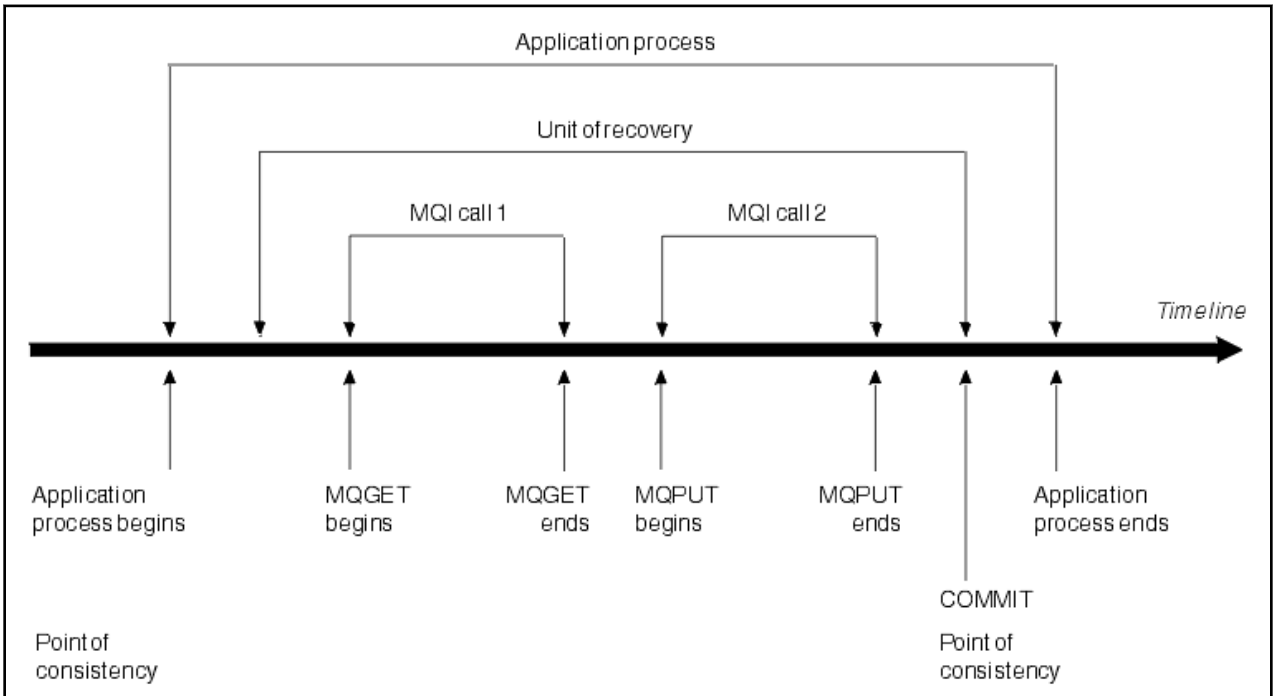


Figure 75. A unit of recovery within an application program

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 233 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 234.

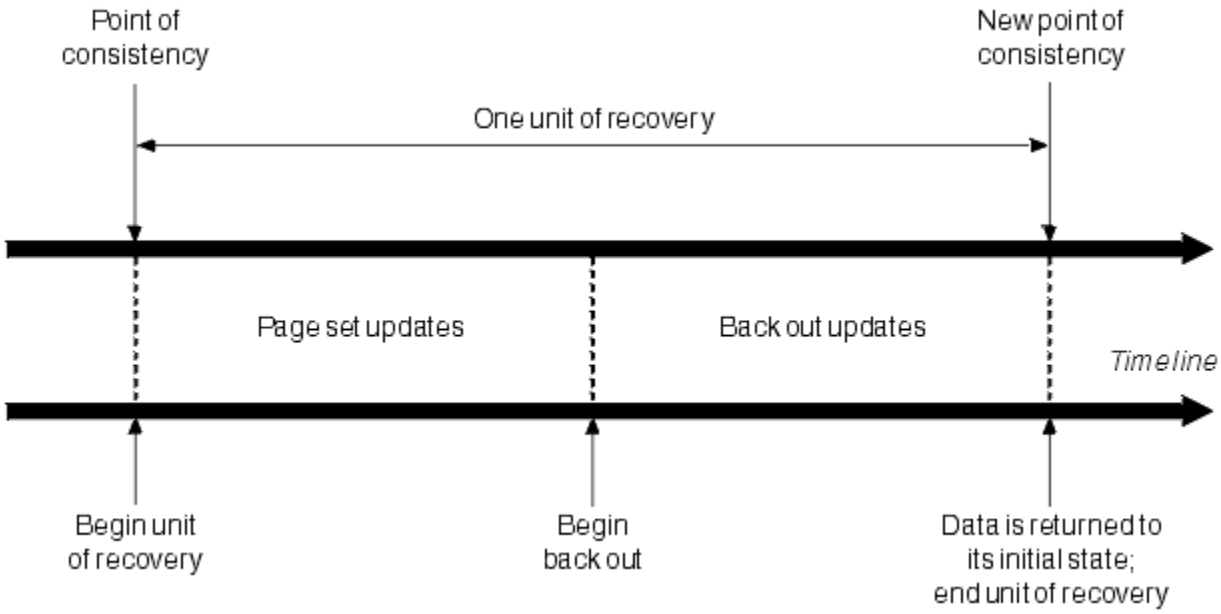


Figure 76. A unit of recovery showing back out

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 235 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.

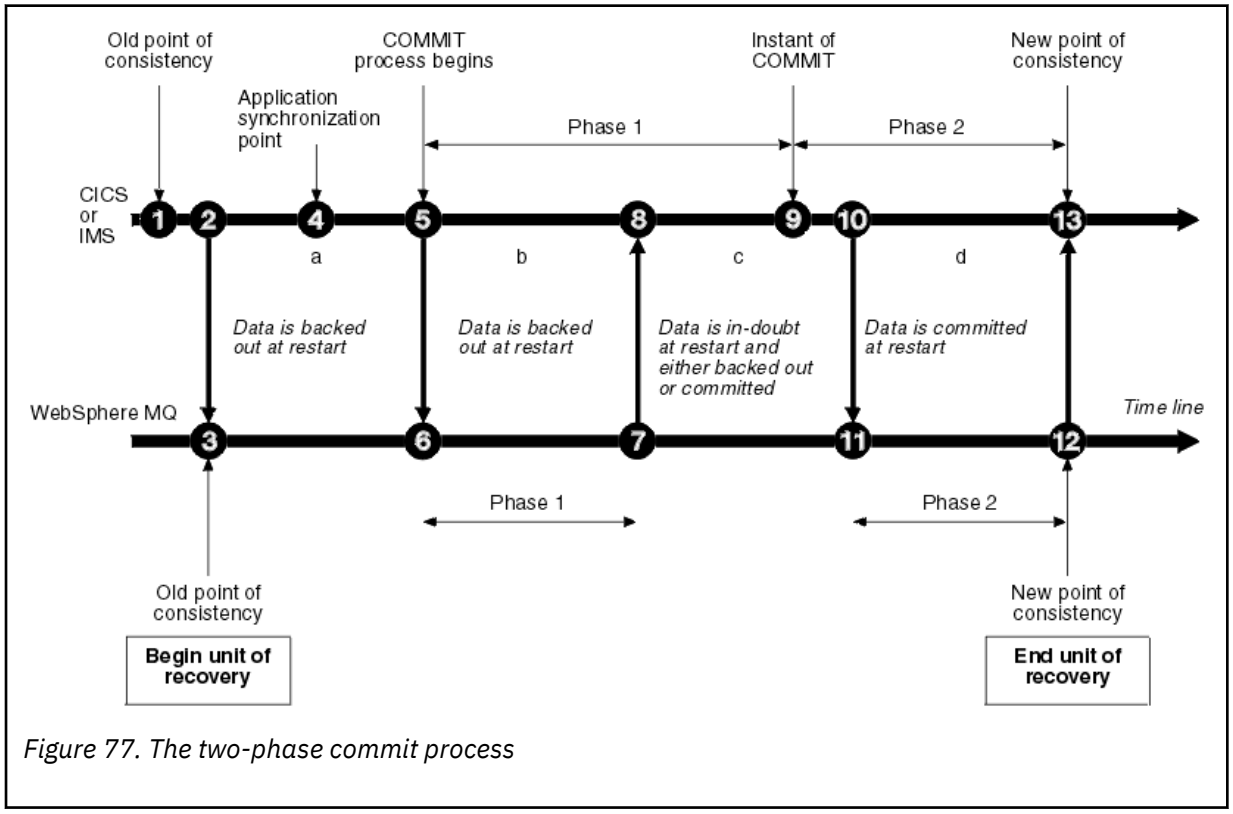


Figure 77. The two-phase commit process

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 235 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 237.

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 239).

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 IBM MQ for z/OS 메시지, 완료 및 이유 코드 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 IBM MQ for z/OS 관리.

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 IBM MQ for z/OS 관리.

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 IBM MQ for z/OS 메시지, 완료 및 이유 코드 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 IBM MQ for z/OS 관리.

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 시스템 명령 참조서.

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 252.

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 253
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 169.

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 242.

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 256.

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 190.

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 150
System parameters	Configuring system parameters
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 258

- "IBM MQ trace" on page 258
- "Events" on page 258

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 259
A queue sharing group can configure and enable support for GROUP units of recovery.

"Application support" on page 260
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 259 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 260.

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 260
Use this page to determine how IBM MQ extended transactional client applications can use the GROUP unit of recovery disposition.

“CICS applications” on page 261
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 통신 서버 and z/OS 기본 기술: 동적 가상 주소 지정 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 261
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 268
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 263
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 266
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 262
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 259

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 264.

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 265.

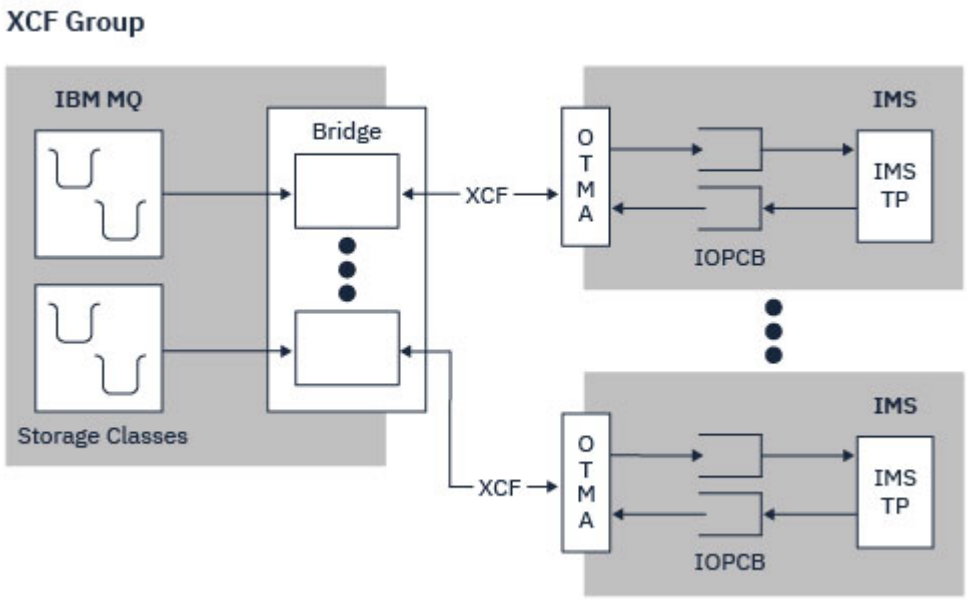


Figure 78. The IBM MQ - IMS bridge

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 263

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

Table 25. Where to find more information about using z/OS Batch with IBM MQ (continued)	
Topic	**Where to look**
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.

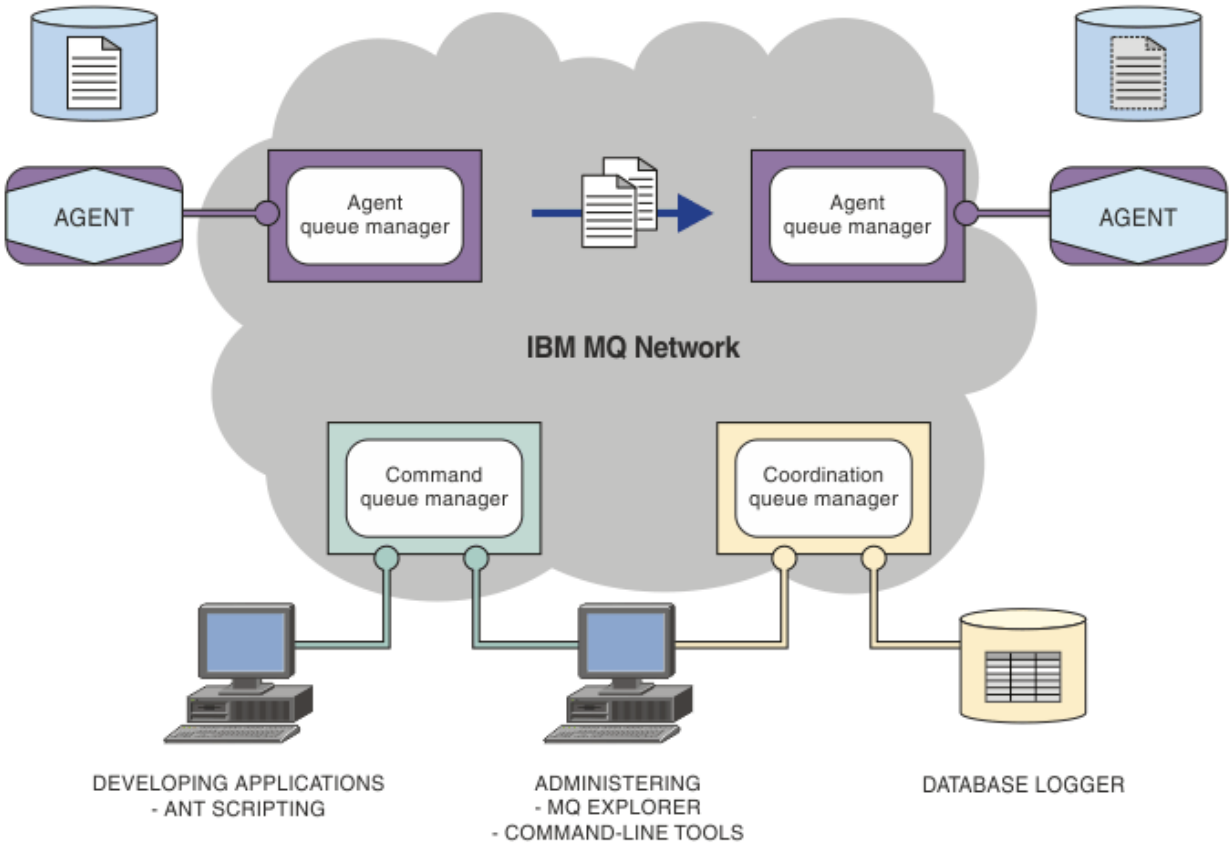
Using IBM MQ functions from JMS applications

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 Transfer는 파일 크기 또는 사용되는 운영 체제에 관계 없이 관리되고 감사 가능한 방법으로 시스템 간에 파일을 전송합니다.

Managed File Transfer를 사용하여 파일 전송을 관리하고 신뢰 가능하며 보안을 유지할 수 있게 해주는 사용자 정의되고 확장 가능하며 자동화된 솔루션을 빌드할 수 있습니다. Managed File Transfer는 많은 비용이 드는 불필요한 중복을 제거하고 유지보수 비용을 낮추며 기존 IT 환경을 최대한 활용합니다.



다이어그램에서는 간단한 Managed File Transfer 토폴로지를 보여줍니다. 두 개의 에이전트가 있으며 각각 IBM MQ 네트워크 내에서 해당하는 고유 에이전트 큐 관리자에 연결됩니다. 파일은 다이어그램의 한쪽에 있는 에이전트에서 IBM MQ 네트워크를 통해 다이어그램의 다른 쪽에 있는 에이전트로 전송됩니다. 또한 IBM MQ 네트워크에는 조정 큐 관리자와 명령 큐 관리자가 있습니다. 애플리케이션 및 도구는 이러한 큐 관리자에 연결하여 IBM MQ 네트워크에서 Managed File Transfer 활동을 구성, 관리, 조작 및 로깅합니다.

Managed File Transfer는 운영 체제 및 전체 설정에 따라 네 가지의 다른 옵션으로 설치할 수 있습니다. Managed File Transfer Agent, Managed File Transfer Logger, Managed File Transfer Service 또는 Managed File Transfer Tools가 이러한 옵션에 해당합니다. 자세한 정보는 Managed File Transfer 제품 옵션 을 참조하십시오.

Managed File Transfer를 사용하여 다음 태스크를 수행할 수 있습니다.

- 관리 파일 전송을 작성합니다.
  - Windows Linux Linux 또는 Windows 플랫폼의 IBM MQ Explorer 에서 새 파일 전송을 작성하십시오.
  - 지원되는 모든 플랫폼의 명령행에서 새 파일 전송을 작성합니다.
  - 파일 전송 기능을 Apache Ant 도구에 통합하십시오.
  - 에이전트 명령 큐에 메시지를 넣어 Managed File Transfer를 제어하는 애플리케이션을 씁니다.
  - 나중에 수행할 파일 전송을 스케줄합니다. 또한 파일 시스템 이벤트(예: 작성된 새 파일) 등을 기반으로 하여 스케줄된 파일 전송을 트리거할 수 있습니다.
  - 디렉토리와 같은 자원을 계속 모니터하며 해당 자원의 컨텐츠가 일부 사전정의된 조건을 충족하는 경우 태스크를 시작합니다. 이 태스크는 파일 전송, Ant 스크립트 또는 JCL 작업일 수 있습니다.
  - IBM MQ 큐로나 큐로부터 파일을 전송합니다.
  - FTP, FTPS 또는 SFTP 서버 간에 파일을 전송합니다.
  - Connect:Direct® 노드 간 파일 전송

  - 텍스트와 2진 파일 모두를 전송합니다. 텍스트 파일은 소스 및 목적지 시스템의 행의 끝 규칙과 코드 페이지 사이에서 자동 변환됩니다.
  - SSL(Secure Socket Layer) 기반 연결에 대한 산업 표준을 사용하여 전송 시 보안을 설정할 수 있습니다.
- 진행 중인 전송을 보고, 네트워크의 모든 전송에 대한 정보를 로그합니다.
  -  Linux 또는 Windows 플랫폼의 IBM MQ Explorer 에서 진행 중인 전송 상태를 보십시오.
  -  Linux 또는 Windows 플랫폼에서 IBM MQ Explorer 를 사용하여 완료된 전송의 상태를 확인하십시오.
  - Managed File Transfer 데이터베이스 로거 기능을 사용하여 Db2 또는 Oracle 데이터베이스에 로그 메시지를 저장합니다.

Managed File Transfer는 애플리케이션 사이에서 문제 없이 메시지를 확실히 전달하는 IBM MQ에 빌드됩니다. 사용자는 IBM MQ의 다양한 기능을 활용할 수 있습니다. 예를 들어, 채널 압축을 사용하여 IBM MQ 채널을 통해 에이전트 간에 보내는 데이터를 압축하고 SSL 채널을 사용하여 에이전트 간에 보내는 데이터의 보안을 유지할 수 있습니다. 파일이 안정적으로 전송되고 실행된 파일 전송이 수행되는 인프라의 실패에 영향을 받지 않을 수 있습니다. 네트워크의 연결이 끊긴 경우, 연결이 복원되면 파일 전송이 중지된 위치에서 재시작됩니다.

파일 전송을 기존 IBM MQ 네트워크와 통합하면 두 개의 개별 인프라를 유지보수하는 데 필요한 자원의 소비를 피할 수 있습니다. IBM MQ 고객이 아닌 경우 IBM MQ 네트워크를 작성하여 Managed File Transfer를 지원함으로써 나중에 SOA를 구현하기 위한 백본을 빌드합니다. 이미 IBM MQ 고객인 경우 Managed File Transfer 는 IBM MQ Internet Pass-Thru 및 IBM Integration Bus를 포함한 기존 IBM MQ 인프라를 이용할 수 있습니다.

IBM MQ 고가용성 솔루션을 활용하여 Managed File Transfer 구성의 복원성을 개선할 수 있습니다. 에이전트가 복제된 데이터 큐 관리자 (RDQM) 를 사용하는 경우 부동 IP 주소 기능을 사용하도록 구성해야 합니다. 이는 에이전트가 동일한 IP 주소를 사용하여 현재 실행 중인 세 개의 RDQM 인스턴스 중 하나와 통신하고 장애 복구 시 자동으로 다시 연결함을 의미합니다 (RDQM 고가용성 및 부동 IP 주소 작성 및 삭제참조). 다중 인스턴스 큐 관리자 솔루션을 사용하는 경우 애플리케이션은 서로 다른 IP 주소를 사용하여 각 인스턴스와 통신하며, 이는 장애 복구 시 클라이언트 다시 연결에 의해 처리됩니다 (다중 인스턴스 큐 관리자 및 채널 및 클라이언트 다시 연결참조).

Managed File Transfer는 다양한 기타 IBM 제품과 통합됩니다.

IBM Integration Bus

IBM Integration Bus 플로우의 일부로 Managed File Transfer 에 의해 전송된 파일을 처리합니다. 자세한 정보는 IBM Integration Bus에서 MFT 에 대한 작업을 참조하십시오.

IBM Sterling Connect:Direct

Managed File Transfer Connect:Direct 브릿지를 사용하여 기존 Connect:Direct 네트워크 간에 파일을 전송합니다. 자세한 정보는 Connect:Direct 브릿지를 참조하십시오.

IBM Tivoli® Composite Application Manager

IBM Tivoli Composite Application Manager가 조정 큐 관리자에 발행된 정보를 모니터하는 데 사용할 수 있는 에이전트를 제공합니다.

관련 개념

Managed File Transfer 제품 옵션

271 페이지의 『MFT 토폴로지 개요』
Managed File Transfer 에이전트를 IBM MQ 네트워크의 조정 큐 관리자와 연결하는 방법에 대한 개요입니다.

270 페이지의 『MFT가 IBM MQ에 대해 작업하는 방법』
Managed File Transfer는 IBM MQ와 다양한 방법으로 상호작용합니다.

MFT가 IBM MQ에 대해 작업하는 방법

Managed File Transfer는 IBM MQ와 다양한 방법으로 상호작용합니다.

- Managed File Transfer는 각 파일을 하나 이상의 메시지로 나누고 IBM MQ 네트워크를 통해 메시지를 전송하여 에이전트 프로세스 간에 파일을 전송합니다.

- 에이전트 프로세스는 IBM MQ 로그에 대한 영향을 최소화하기 위해 비지속 메시지를 사용하여 파일 데이터를 이동합니다. 에이전트 프로세스는 다른 프로세스와 통신하여 파일 데이터가 포함된 메시지 플로우를 조절합니다. 이는 IBM MQ 전송 큐에 빌드되는 파일 데이터가 메시지에 포함되는 것을 막으며 비지속 메시지가 전달되지 않는 경우 파일 데이터가 다시 송신되도록 합니다.
- Managed File Transfer 에이전트는 다수의 IBM MQ 큐를 사용합니다. 자세한 정보는 MFT 시스템 큐 및 시스템 토픽을 참조하십시오.
- 이러한 큐 중 일부는 내부용으로 엄격하게 제한되어 있지만 에이전트는 에이전트가 읽는 특정 큐로 송신된 특별하게 형식화된 명령 메시지 양식의 요청을 승인할 수 있습니다. 명령행 명령 및 IBM MQ Explorer 플러그인 모두 IBM MQ 메시지를 에이전트에 전송하여 에이전트가 원하는 조치를 수행하도록 지시합니다. 이러한 방법으로 에이전트와 상호작용하는 IBM MQ 애플리케이션을 작성할 수 있습니다. 자세한 정보는 에이전트 명령 큐에 메시지를 넣어 MFT 제어를 참조하십시오.
- Managed File Transfer 에이전트는 조정 큐 관리자로 지정된 MQ 큐 관리자에 대한 전송의 진행 및 결과와 해당 상태에 대한 정보를 송신합니다. 이 정보는 조정 큐 관리자에 의해 발행되며 전송 진행 상태를 모니터하거나 일어난 전송을 기록할 애플리케이션에 의해 구독될 수 있습니다. 명령행 명령 및 IBM MQ Explorer 플러그인은 둘 다 발행되는 정보를 사용할 수 있습니다. 이 정보를 사용하는 IBM MQ 애플리케이션을 작성할 수 있습니다. 정보가 발행되는 토픽에 대한 자세한 정보는 SYSTEM.FTE 주제.
- Managed File Transfer의 주요 컴포넌트는 IBM MQ 큐 관리자의 기능을 활용하여 메시지를 저장하고 전달합니다. 이는 정지 상태가 되는 경우, 이에 영향을 받지 않는 인프라의 일부가 계속 파일을 전송할 수 있음을 의미합니다. 이러한 내용은 조정 큐 관리자에게도 해당되어, 저장 및 전달의 조합 및 지속 가능 구독을 사용하면 조정 큐 관리자가 발생한 파일 전송에 관한 핵심 정보를 잃지 않고 사용 불가능 상태에서도 작동할 수 있습니다.

MFT 토폴로지 개요

Managed File Transfer 에이전트를 IBM MQ 네트워크의 조정 큐 관리자와 연결하는 방법에 대한 개요입니다.

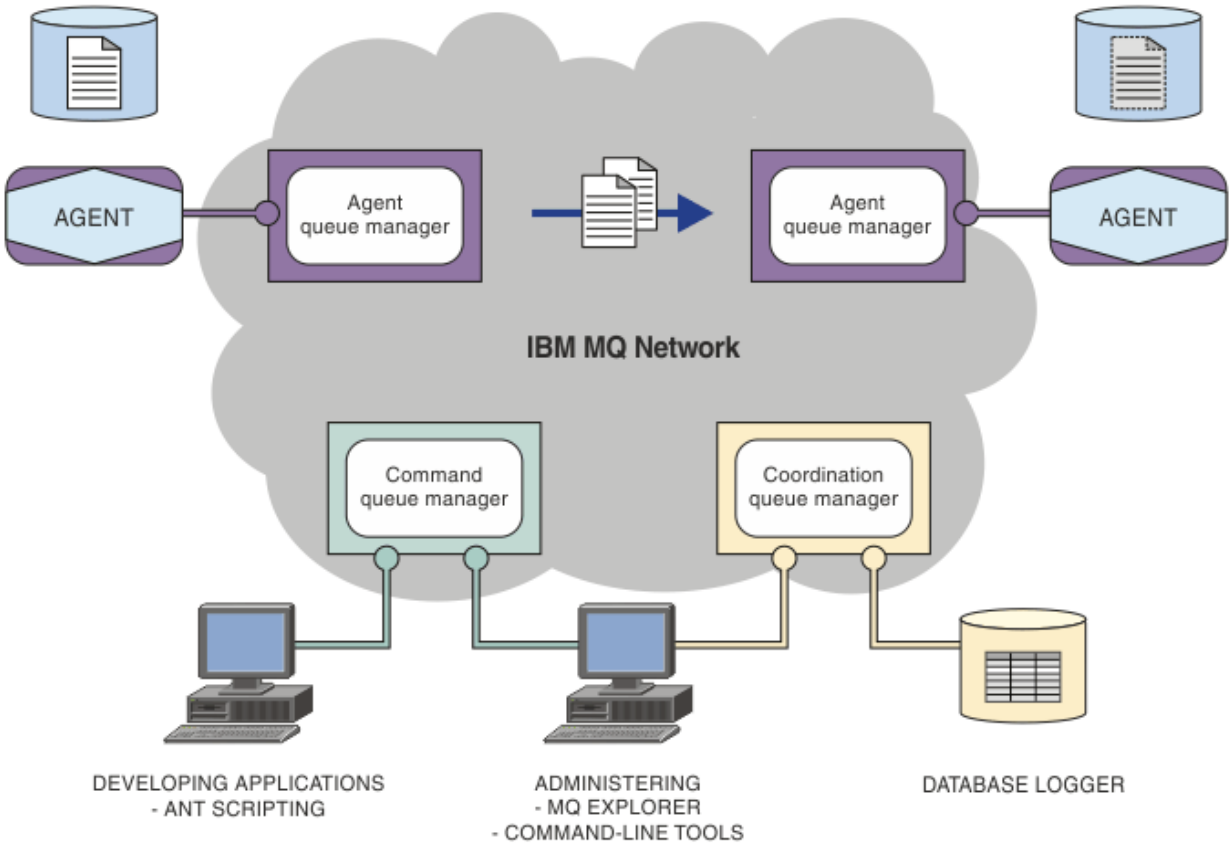
Managed File Transfer 에이전트는 전송되는 파일을 송수신합니다. 각 에이전트는 연관된 큐 관리자에 고유한 큐 세트를 가지고 있으며 에이전트는 바인딩 또는 클라이언트 모드로 큐 관리자에 연결됩니다. 에이전트는 조정 큐 관리자를 큐 관리자로 사용할 수도 있습니다.

조정 큐 관리자는 감사 및 파일 전송 정보를 브로드캐스트합니다. 조정 큐 관리자는 에이전트, 전송 상태 및 전송 감사 정보 콜렉션의 단일 지점을 나타냅니다. 전송이 이루어지도록 하기 위해 조정 큐 관리자가 사용 가능해야 하는 것은 아닙니다. 조정 큐 관리자를 일시적으로 사용할 수 없는 경우에도 전송은 정상적으로 계속 됩니다. 조정 큐 관리자가 사용 가능하게 되어 정상적으로 처리될 수 있을 때까지 감사 및 상태 메시지가 에이전트 큐 관리자에 저장됩니다.

에이전트는 조정 큐 관리자와 함께 등록되며 세부사항을 해당 큐 관리자에 발행합니다. 이 에이전트 정보는 Managed File Transfer 플러그인이 IBM MQ Explorer에서 전송 시작을 사용으로 설정하는 데 사용됩니다. 또한 에이전트 정보 및 에이전트 상태를 표시하는 명령이 조정 큐 관리자에 수집된 에이전트 정보를 사용하기도 합니다.

전송 상태 및 전송 감사 정보는 조정 큐 관리자에서 발행됩니다. 전송 상태 및 전송 감사 정보는 Managed File Transfer 플러그인이 IBM MQ Explorer에서 전송 진행 상황을 모니터하는 데 사용됩니다. 조정 큐 관리자에 저장된 전송 감사 정보는 감사 가능성을 제공하기 위해 보유할 수 있습니다.

명령 큐 관리자는 IBM MQ 네트워크에 연결하는 데 사용되며 Managed File Transfer 명령을 실행할 때 연결된 큐 관리자입니다.



관련 개념

268 페이지의 『Managed File Transfer』
Managed File Transfer는 파일 크기 또는 사용되는 운영 체제에 관계 없이 관리되고 감사 가능한 방법으로 시스템 간에 파일을 전송합니다.

270 페이지의 『MFT가 IBM MQ에 대해 작업하는 방법』
Managed File Transfer는 IBM MQ와 다양한 방법으로 상호작용합니다.

Managed File Transfer 시나리오

MFT REST API 개요

REST API는 파일 전송 에이전트에 대한 세부사항 및 전송 목록을 포함하여 특정 Managed File Transfer 명령을 지원합니다.

REST API는 현재 모든 Managed File Transfer 전송을 나열하고 Managed File Transfer 에이전트의 상태를 조회하는 옵션을 포함합니다. 자세한 정보는 REST API MFT 시작하기를 참조하십시오.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru(MQIPT)는 인터넷을 통한 원격 사이트 간 메시징 솔루션을 구현하는 데 사용할 수 있는, IBM MQ의 선택적 컴포넌트입니다.

IBM MQ 9.4.x에 대한 MQIPT 설치 파일을 얻으려면 IBM MQ 용 IBM Fix Central로 이동하십시오.

MQIPT 를 사용하여 지원되는 모든 IBM MQ버전을 연결할 수 있습니다. MQIPT와 동일한 버전의 다른 IBM MQ 컴포넌트를 설치할 필요가 없습니다.

IBM MQ 인타이틀먼트를 구매한 경우 필요한 만큼 MQIPT의 사본을 설치할 수 있습니다. MQIPT 설치는 구매한 IBM MQ 인타이틀먼트로 계수되지 않습니다. IBM MQ 라이센싱에 대한 자세한 정보는 IBM MQ 라이센스 정보를 참조하십시오.

참고: 이 문서는 IBM MQ 9.4의 MQIPT 와 관련되어 있습니다. IBM Documentation의 MQIPT 지원 팩 (버전 2.1) 문서는 IBM MQ 9.0 문서의 MQIPT (SupportPac MS81) 을 참조하십시오.

참고: MQIPT 2.1 또는 이전 버전을 사용하는 경우 IBM MQ 9.4용 MQIPT 로 업그레이드하는 것이 좋습니다. MQIPT 지원 팩에 대한 지원 종료 날짜가 2020년 9월 30th 이기 때문입니다.

IBM MQ Internet Pass-Thru는 두 개의 IBM MQ 큐 관리자 또는 IBM MQ 클라이언트와 IBM MQ 사이에서 IBM MQ 메시지 플로우를 수신 및 전달할 수 있는 독립형 서비스로 실행됩니다.

클라이언트 및 서버가 동일한 물리적 네트워크에 존재하지 않을 경우 MQIPT에서 이 연결을 사용으로 설정합니다.

하나 이상의 MQIPT 인스턴스를 두 개의 IBM MQ 큐 관리자 또는 IBM MQ 클라이언트와 IBM MQ 큐 관리자 간의 통신 경로에 배치할 수 있습니다. MQIPT 인스턴스를 사용하는 경우 두 개의 시스템 간에 직접 TCP/IP 연결 없이도 두 개의 IBM MQ 시스템에서 메시지를 교환할 수 있습니다. 이 아키텍처는 방화벽 구성이 두 시스템 간의 직접 TCP/IP 연결을 금지하는 경우에 유용합니다.

MQIPT 는 하나 이상의 TCP/IP 포트에서 수신 연결을 청취합니다. 이러한 연결은 일반 IBM MQ 메시지, HTTP 내부에서 터널링된 IBM MQ 메시지 또는 TLS (Transport Layer Security) 또는 SSL (Secure Sockets Layer) 을 사용하여 암호화된 메시지를 전달할 수 있습니다. MQIPT는 복수의 동시 연결을 핸들링할 수 있습니다.

초기 TCP/IP 연결 요청을 작성하는 IBM MQ 채널은 호출자로 참조되고, 연결을 시도하는 대상 채널은 응답자로 참조되며, 최종적으로 연결을 시도하는 대상 큐 관리자는 대상 큐 관리자로 참조됩니다.

MQIPT는 소스에서 목적지로 전달할 때 메모리에 데이터를 보관합니다. 디스크에 데이터가 저장되지 않습니다 (운영 체제가 디스크에 페이징하는 메모리 제외). MQIPT 가 디스크에 명시적으로 액세스하는 유일한 경우는 구성 파일을 읽고 연결 로그 및 추적 레코드를 쓰는 것입니다.

전체 범위의 IBM MQ 채널 유형은 하나 이상의 MQIPT인스턴스를 통해 연결할 수 있습니다. 통신 경로에 MQIPT 가 있어도 연결된 IBM MQ 구성요소의 기능 특성에는 영향을 미치지 않습니다. 그러나 메시지 전송 성능에 영향을 줄 수 있습니다.

MQIPT 는 276 페이지의 『MQIPT의 가능한 구성』 에 설명된 대로 IBM MQ 와 함께 사용할 수 있습니다.

MQIPT를 설치하려면 MQIPT 설치를 참조하십시오.

관련 태스크

IBM MQ Internet Pass-Thru 구성

IBM MQ Internet Pass-Thru 관리 및 구성

관련 참조

IBM MQ Internet Pass-Thru 구성 참조

MQIPT의 사용법

IBM MQ Internet Pass-Thru(MQIPT)에 대한 다양한 잠재적 사용법이 존재합니다.

MQIPT를 채널 집선기로 사용할 수 있음

이 방법으로 MQIPT를 사용하는 경우 방화벽에 독립적인 복수의 호스트와 통신하는 채널을 모두 MQIPT 호스트와 통신하는 것처럼 표시할 수 있습니다. 이 경우 더 간단히 방화벽 필터링 규칙을 정의하고 관리할 수 있습니다.

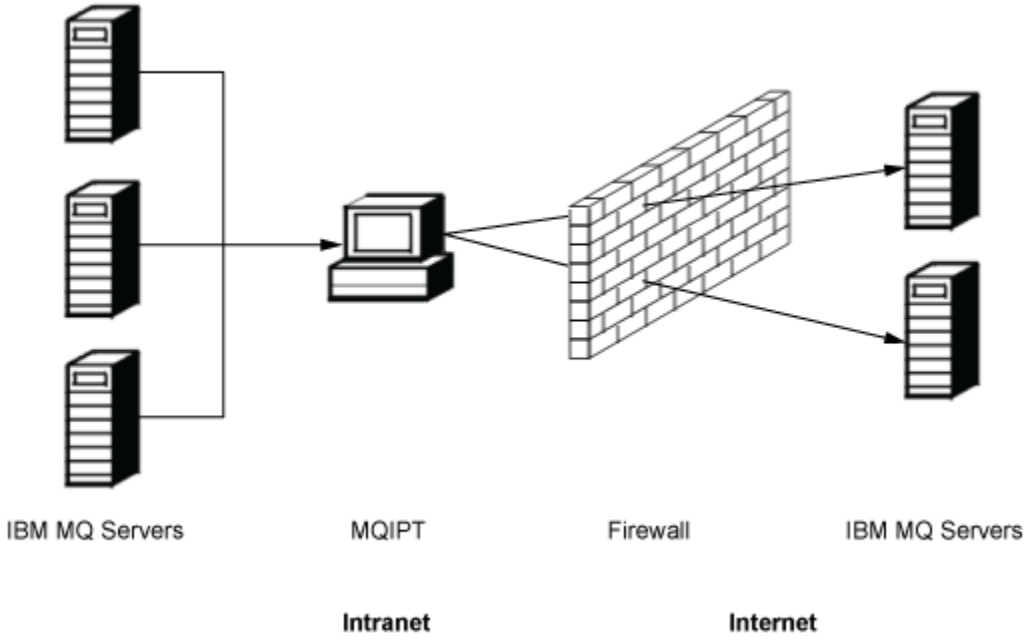


그림 *79.* 채널 집선기로 사용하는 *MQIPT* 예제

단일 액세스 지점을 제공하기 위해 MQIPT를 DMZ에 배치할 수 있음

MQIPT가 DMZ 방화벽(근거리 통신망(LAN) 보안을 위한 방화벽 구성) 내에 배치되는 경우 알려지고 신뢰할 수 있는 IP(Internet Protocol)를 보유한 컴퓨터에서 MQIPT를 사용하여 수신 IBM MQ 채널 연결을 청취한 후 신뢰할 수 있는 인트라넷에 전달할 수 있으며, 내부 방화벽에서 이 신뢰할 수 있는 컴퓨터가 인바운드 연결을 작성하도록 허용해야 합니다. 이 구성에서는 MQIPT가 액세스에 대한 외부 요청이 신뢰할 수 있는 인트라넷에 있는 컴퓨터의 실제 IP 주소를 수신할 수 없도록 차단합니다. 이러한 방식으로 MQIPT는 단일 액세스 지점을 제공합니다. 필요한 경우, MQIPT 는 TLS 연결을 승인하고 별도의 TLS 연결을 사용하여 데이터를 대상으로 전달하도록 구성할 수 있으므로 DMZ에서 TLS 세션을 종료합니다.

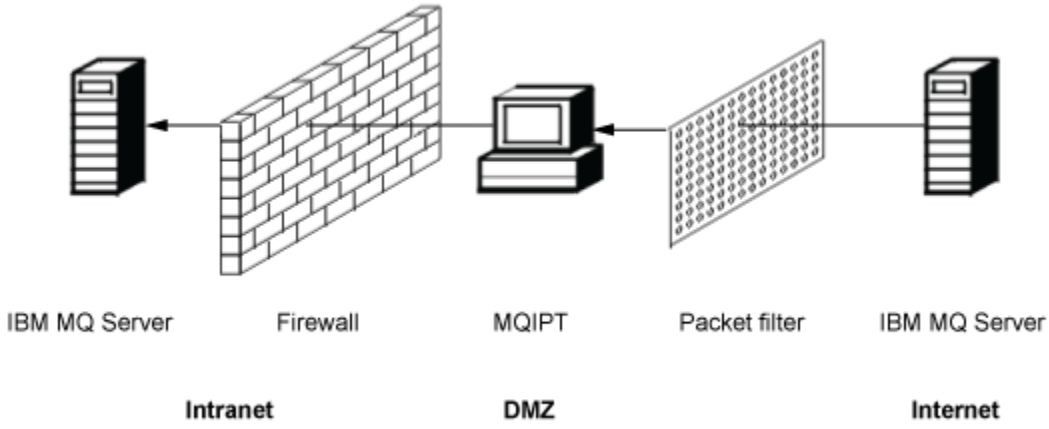


그림 *80. DMZ* 방화벽의 *MQIPT* 예제

MQIPT는 HTTP 터널링 방식으로 통신할 수 있음

두 개의 MQIPT 인스턴스가 나란히 배치되어 있는 경우 해당 인스턴스는 HTTP를 사용하여 통신할 수 있습니다. HTTP 터널링 기능을 사용하는 경우 기존 HTTP 프록시를 사용하여 방화벽을 통해 요청을 전송할 수 있습니다.

첫 번째 MQIPT는 IBM MQ 프로토콜을 HTTP에 삽입하고, 두 번째는 해당 HTTP 랩퍼에서 IBM MQ 프로토콜을 추출한 후 해당 프로토콜을 목적지 큐 관리자로 전달합니다.

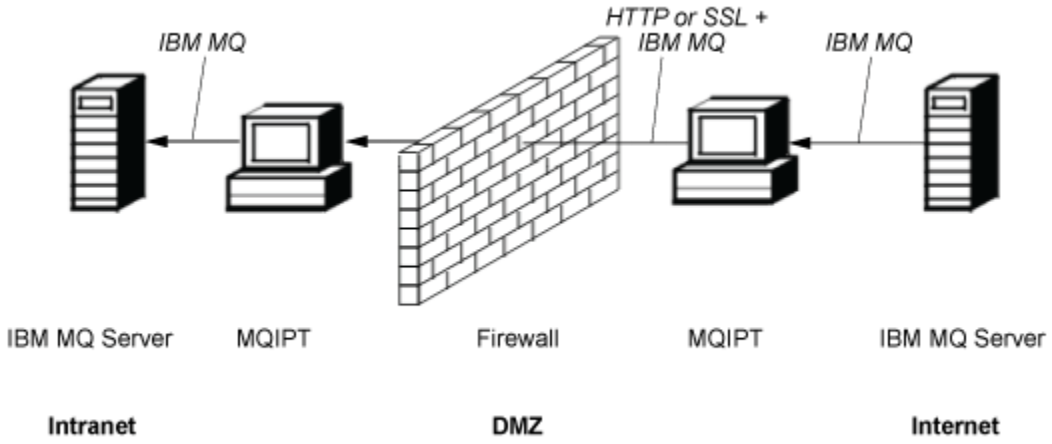


그림 *81. MQIPT* 및 *HTTP* 터널링 예제

MQIPT는 메시지를 암호화할 수 있음

MQIPT가 이전 예제와 같이 구성된 경우 방화벽을 통해 전송되기 전에 요청을 암호화할 수 있습니다. 첫 번째 MQIPT는 데이터를 암호화하고 두 번째는 목적지 큐 관리자로 송신하기 전에 SSL/TLS를 사용하여 데이터를 복호화합니다.

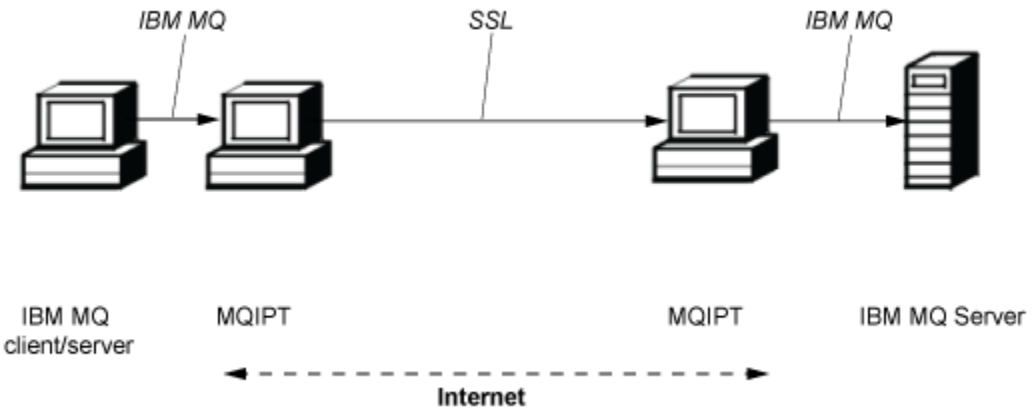


그림 *82. MQIPT* 및 *SSL/TLS* 예제

MQIPT의 작동 방식

가장 단순한 구성에서 MQIPT는 IBM MQ 프로토콜 전달자로 작동합니다. TCP/IP 포트를 청취하고 IBM MQ 채널로부터의 연결 요청을 승인합니다.

올바르게 구성된 요청이 수신되는 경우 MQIPT는 자신과 목적지 IBM MQ 큐 관리자 사이에 추가적인 TCP/IP 연결을 설정합니다. 그런 다음 해당 수신 연결에서 수신되는 모든 프로토콜 패킷을 목적지 큐 관리자로 전달한 후 목적지 큐 관리자의 프로토콜 패킷을 다시 원래 수신 연결로 리턴합니다.

어느 쪽에서도 중개자의 존재를 직접 인식하지 못하기 때문에 IBM MQ 프로토콜에 대한 변경사항(클라이언트/서버 또는 큐 관리자에서 큐 관리자로)이 포함되지 않습니다. 새 버전의 IBM MQ 클라이언트 또는 서버 코드는 필요하지 않습니다.

MQIPT를 사용하려면 목적지 큐 관리자의 호스트 이름 및 포트가 아닌 MQIPT 호스트 이름 및 포트를 사용하도록 호출자 채널이 구성되어야 합니다. 이는 IBM MQ 채널의 **CONNAME** 특성으로 정의됩니다. MQIPT는 수신 데이

터를 읽어들인 후 단순히 목적지 큐 관리자로 전달합니다. 클라이언트/서버 채널의 사용자 ID 및 비밀번호와 같은 다른 구성 필드도 마찬가지로 목적지 큐 관리자로 전달됩니다.

다중 큐 관리자

MQIPT를 사용하여 둘 이상의 목적지 큐 관리자에 대한 액세스를 허용할 수 있습니다. 이 작업을 수행하려면 MQIPT에 연결할 큐 관리자를 지정하는 메커니즘이 존재해야 하기 때문에 MQIPT는 수신 TCP/IP 포트 번호를 사용하여 연결할 큐 관리자를 판별합니다.

따라서 복수의 TCP/IP 포트를 청취하도록 MQIPT를 구성할 수 있습니다. 각각의 청취 포트는 MQIPT 라우트를 통해 목적지 큐 관리자에 맵핑됩니다. 이러한 라우트는 최대 100개까지 정의할 수 있으며 청취 TCP/IP 포트를 목적지 큐 관리자의 호스트 이름 및 포트와 연관시킵니다. 즉, 목적지 큐 관리자의 호스트 이름(IP 주소)은 원래 채널에 절대 표시되지 않습니다. 각각의 라우트는 해당 청취 포트와 목적지 사이에서 복수의 연결을 핸들링할 수 있으며, 각각의 연결은 독립적으로 수행됩니다.

MQIPT 구성 파일

MQIPT는 `mqipt.conf`라는 구성 파일을 사용합니다. 이 파일에는 모든 라우트 정의 및 연관된 특성이 포함되어 있습니다. `mqipt.conf`에 대한 자세한 정보는 IBM MQ Internet Pass-Thru 관리 및 구성을 참조하십시오.

MQIPT가 시작되면 구성 파일에 나열된 각각의 라우트가 시작됩니다. 시스템 콘솔에는 각 라우트의 상태를 표시하는 메시지가 기록됩니다. 라우트에 대해 MQCPI078 메시지가 표시되면 해당 라우트에서 연결 요청을 승인할 준비가 된 것입니다.

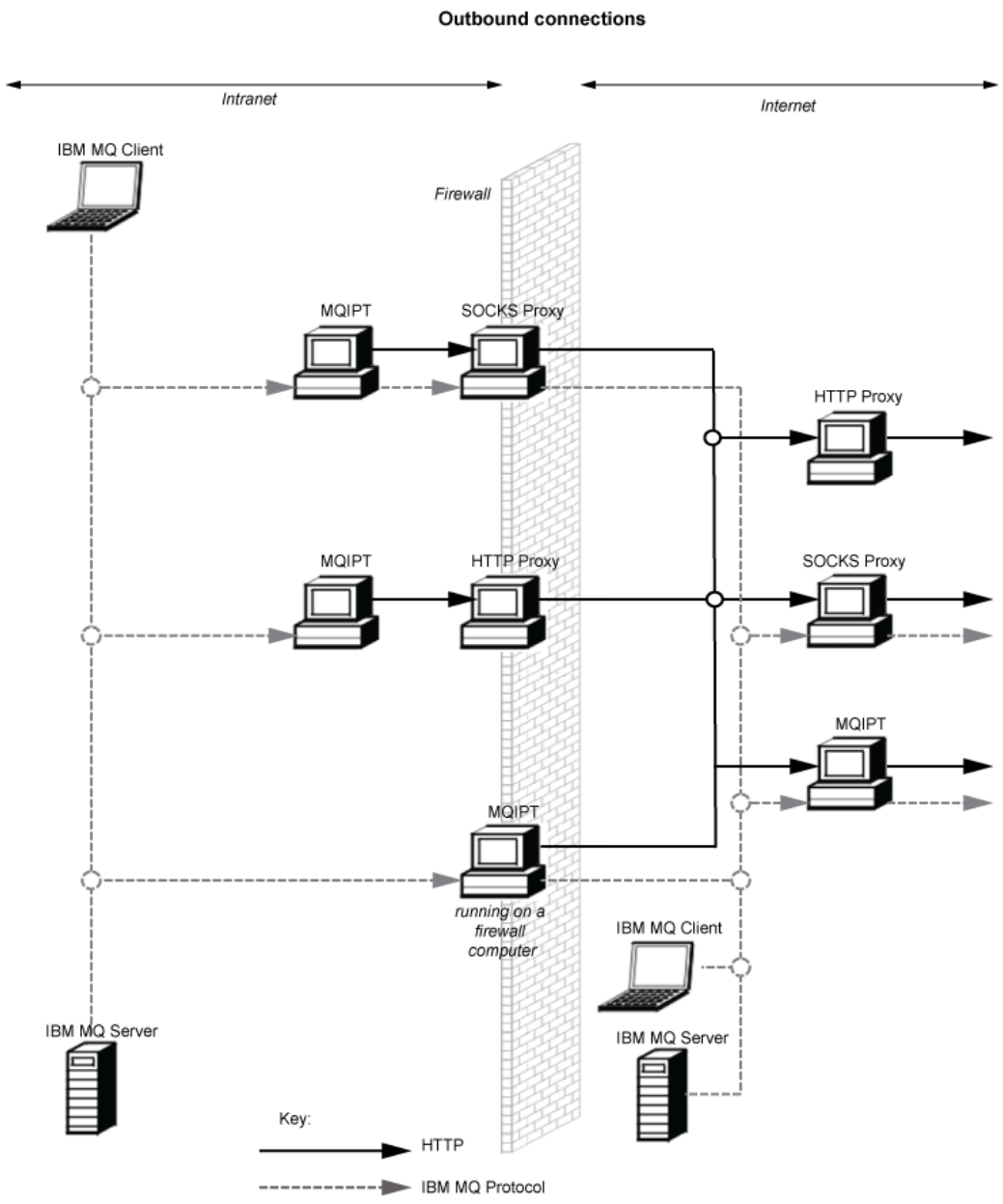
MQIPT의 가능한 구성

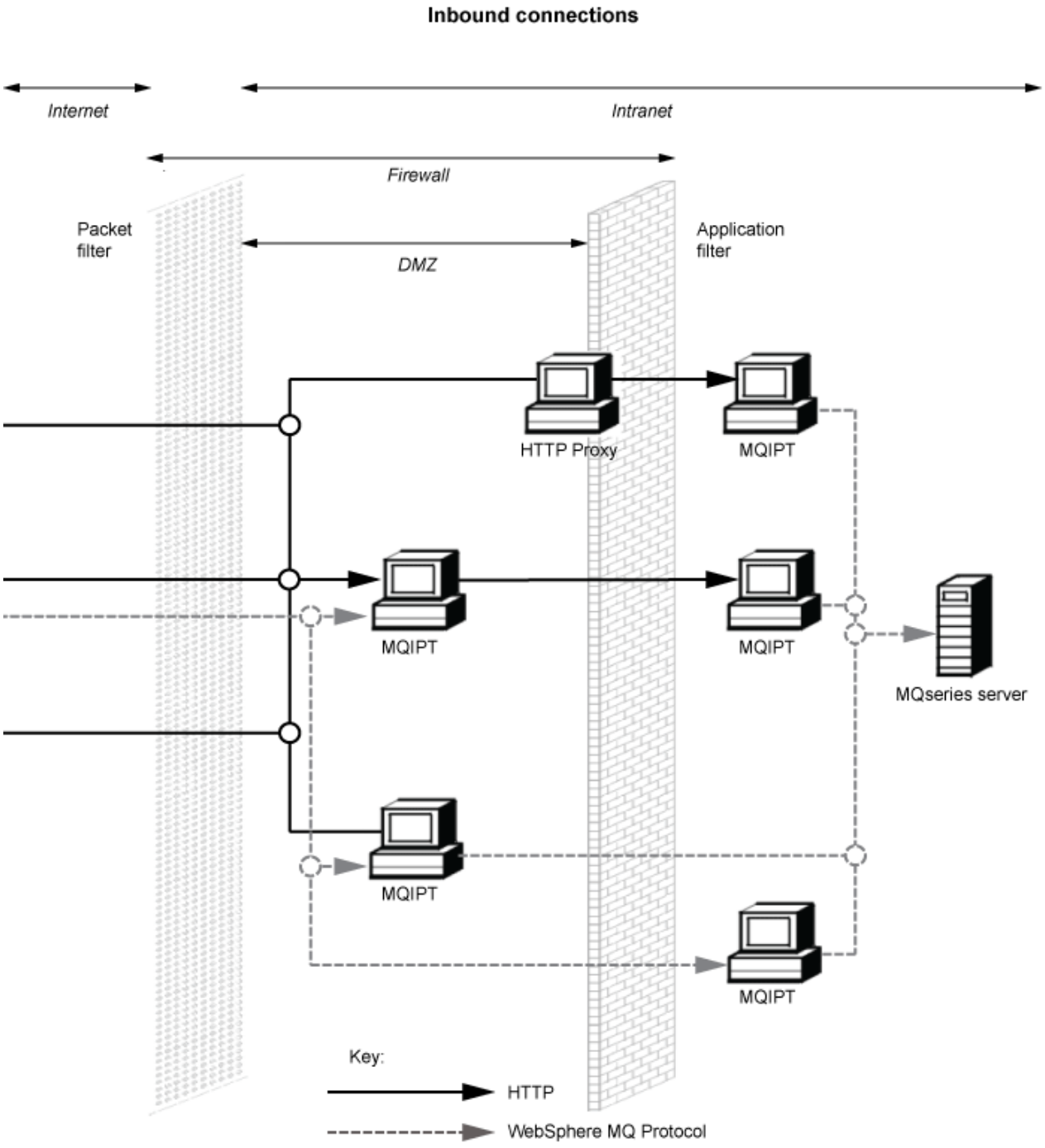
MQIPT는 IBM MQ 및 IBM Integration Bus과 함께 사용할 수 있습니다.

다음 다중 파트 그림은 IBM MQ 토폴로지에서 MQIPT 에 대해 가능한 많은 구성을 보여줍니다. 이 그림은 MQIPT에서 메시지를 송신하는 다양한 방법을 보여줍니다. MQIPT, HTTP 프록시 또는 SOCKS 프록시로 메시지를 전달하여 인트라넷, 방화벽 내부 및 방화벽 외부의 인터넷에서 해당 메시지를 전달하는 클라이언트 및 서버를 보여줍니다.

메시지는 인바운드 방화벽을 통해 서버로 전달하기 전에 DMZ의 MQIPT 프록시 또는 HTTP 프록시에서 수신됩니다.

방화벽의 인트라넷 측에 있는 HTTP 프록시, SOCKS 프록시 및 MQIPT 컴퓨터는 인터넷에서 서로 연결되어 있는 복수의 컴퓨터가 존재할 가능성을 나타냅니다. 예를 들어 MQIPT 컴퓨터는 대상에 도달하기 전에 하나 이상의 SOCKS 또는 HTTP 프록시 컴퓨터나 추가적인 MQIPT 컴퓨터를 통해 통신할 수 있습니다.


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



호환 가능 구성

IBM MQ 클라이언트 또는 큐 관리자가 MQIPT와 통신하는 호환 가능 연결 시나리오입니다. 목적지 큐 관리자와 통신하기 위해 동일한 또는 두 번째 MQIPT 라우트가 사용됩니다.

하나의 MQIPT 라우트가 포함된 호환 가능 구성

하나의 MQIPT 라우트를 사용하여 IBM MQ와 통신할 수 있습니다.

279 페이지의 표 26의 열에는 다음과 같은 정보가 포함되어 있습니다.

1. IBM MQ와 MQIPT 라우트 사이에 사용되는 프로토콜. 이 연결은 IBM MQ 클라이언트 또는 큐 관리자에서 작성할 수 있으며 IBM MQ FAP(Formats and Protocols) 또는 SSL/TLS 프로토콜을 사용할 수 있습니다.
2. MQIPT 라우트가 작동하는 모드. MQIPT와 IBM MQ 간의 인터넷을 통한 통신 형식은 MQIPT 라우트의 구성에 따라 판별됩니다. 테이블에서 SSL이 언급되는 위치를 메모해 두십시오. TLS를 사용할 수도 있습니다.
3. MQIPT 라우트와 목적지 큐 관리자 사이에 사용되는 프로토콜.

표 *26.* 하나의 *MQIPT* 인스턴스가 포함된 올바른 구성

1. IBM MQ 소스 프로토콜	**2. MQIPT 라우트의 모드**	**3. IBM MQ 대상 프로토콜**
FAP	FAP 프록시(기본값)	FAP
	FAP 서버 및 SSL 클라이언트	SSL/TLS
SSL/TLS	SSL 프록시	SSL/TLS
	SSL 서버 및 FAP 클라이언트	FAP
	SSL 서버 및 SSL 클라이언트	SSL/TLS

둘 이상의 MQIPT 라우트가 포함된 호환 가능 구성

둘 이상의 MQIPT 인스턴스에서 둘 이상의 라우트를 사용하여 IBM MQ와 통신하도록 선택할 수도 있습니다.

279 페이지의 표 27의 열에는 다음과 같은 정보가 포함되어 있습니다.

1. IBM MQ와 첫 번째 MQIPT 라우트 사이에 사용되는 프로토콜. 이 연결은 IBM MQ 클라이언트 또는 큐 관리자에서 작성할 수 있으며 IBM MQ FAP(Formats and Protocols) 또는 SSL/TLS 프로토콜을 사용할 수 있습니다.
2. 첫 번째 MQIPT 라우트가 작동하는 모드. MQIPT와 IBM MQ 간의 인터넷을 통한 통신 형식은 MQIPT 라우트의 구성에 따라 판별됩니다. 테이블에서 SSL이 언급되는 위치를 메모해 두십시오. TLS를 사용할 수도 있습니다.
3. 두 번째 MQIPT 라우트가 작동하는 모드.
4. 두 번째 MQIPT 라우트와 목적지 큐 관리자 사이에 사용되는 프로토콜.

표 *27.* 복수의 *MQIPT* 인스턴스가 포함된 올바른 구성

1. IBM MQ 소스 프로토콜	**2. 첫 번째 MQIPT 라우트의 모드**	**3. 두 번째 MQIPT 라우트의 모드**	**4. IBM MQ 대상 프로토콜**
FAP(기본값)	FAP 프록시(기본값)	FAP 프록시(기본값)	FAP
	FAP 서버 및 SSL 클라이언트	SSL 프록시	SSL/TLS
		SSL 서버 및 FAP 클라이언트	FAP
		SSL 서버 및 SSL 클라이언트	SSL/TLS
	HTTP 클라이언트	HTTP 서버 및 SSL 클라이언트	SSL/TLS
	HTTPS 클라이언트	HTTPS 서버 및 SSL 클라이언트	SSL/TLS
	HTTP 클라이언트	HTTP 서버	FAP
	HTTPS 클라이언트	HTTPS 서버	FAP

표 27. 복수의 *MQIPT* 인스턴스가 포함된 올바른 구성 (계속)

1. IBM MQ 소스 프로토콜	2. 첫 번째 MQIPT 라우트의 모드	3. 두 번째 MQIPT 라우트의 모드	4. IBM MQ 대상 프로토콜
SSL/TLS	SSL 프록시	SSL 프록시	SSL/TLS
		SSL 서버 및 FAP 클라이언트	FAP
		SSL 서버 및 SSL 클라이언트	SSL/TLS
	HTTP 클라이언트	HTTP 서버	FAP
	HTTPS 클라이언트	HTTPS 서버	SSL/TLS
	HTTP 클라이언트	HTTP 서버 및 SSL 클라이언트	FAP
	HTTPS 클라이언트	HTTPS 서버 및 SSL 클라이언트	SSL/TLS

지원되는 채널 구성

모든 IBM MQ 채널 유형이 지원되지만 구성은 TCP/IP 연결로 제한됩니다. IBM MQ 클라이언트 또는 큐 관리자에는 MQIPT가 목적지 큐 관리자인 것처럼 표시됩니다. 채널 구성에 목적지 호스트 및 포트 번호가 필요한 경우 MQIPT 호스트 이름 및 목적지 포트 번호가 지정됩니다.

클라이언트/서버 채널
MQIPT는 수신 클라이언트 연결 요청을 청취한 후 HTTP 터널링, SSL/TLS 또는 표준 IBM MQ 프로토콜 패킷을 사용하여 해당 요청을 전달합니다. MQIPT에서 HTTP 터널링 또는 SSL/TLS를 사용하는 경우 연결 시 해당 요청을 두 번째 MQIPT로 전달합니다. HTTP 터널링을 사용하지 않을 경우 연결 시 해당 요청을 목적지 큐 관리자로 표시되는 항목으로 전달합니다(결과적으로 추가적인 MQIPT가 될 수도 있지만). 목적지 큐 관리자에서 클라이언트 연결을 승인하면 클라이언트와 서버 사이에서 패킷이 릴레이됩니다.

클러스터 송신자/수신자 채널
MQIPT가 클러스터 송신자 채널로부터 수신 요청을 수신하는 경우 큐 관리자에서 SOCKS가 사용으로 설정되어 있으며 SOCKS 데이터 교환 프로세스 중에 실제 목적지 주소를 확보하는 것으로 가정합니다. 이 경우 요청을 클라이언트 연결 채널과 정확하게 동일한 방식으로 MQIPT 또는 목적지 큐 관리자로 전달합니다. 여기에는 자동 정의 클러스터 송신자 채널도 포함됩니다.

송신자/수신자
MQIPT가 송신자 채널로부터 수신 요청을 수신하는 경우 요청을 클라이언트 연결 채널과 정확하게 동일한 방식으로 다음 MQIPT 또는 목적지 큐 관리자로 전달합니다. 목적지 큐 관리자는 수신 요청을 유효성 검증한 후 적절한 경우 수신자 채널을 시작합니다. 송신자와 수신자 채널 간의 모든 통신(보안 플로우 포함)이 릴레이됩니다.

요청자/서버
이 결합은 이전 구성과 동일한 방식으로 핸들링됩니다. 연결 요청에 대한 유효성 검증은 목적지 큐 관리자의 서버 채널에서 수행됩니다.

요청자/송신자
두 개의 큐 관리자가 서로 직접 연결을 설정하도록 허용되지 않지만 둘 다 MQIPT에 연결하여 연결을 승인하도록 허용된 경우 "콜백" 구성을 사용할 수 있습니다.

서버/요청자 및 서버/수신자
이는 `Sender/Receiver` 구성을 처리하는 것과 동일한 방식으로 MQIPT 에 의해 처리됩니다.

채널 종료 및 실패 조건

MQIPT에서 IBM MQ 채널의 폐쇄(정상 또는 비정상)를 감지하는 경우 채널 폐쇄를 전파합니다. MQIPT를 사용하여 라우트를 닫을 경우 해당 채널을 통해 이동하는 모든 채널이 닫힙니다.

MQIPT는 선택적 유휴 제한시간 기능을 제공합니다. MQIPT에서 특정 채널이 제한시간을 초과하는 기간 동안 유효 상태임을 감지하는 경우 문제의 두 연결에서 즉시 종료를 수행합니다.

채널 양쪽의 IBM MQ 시스템은 이러한 비정상 종료 조건이 네트워크 장애인지 또는 파트너에 의한 채널 종료인지를 관찰합니다. 그런 다음 채널을 다시 시작하고 MQIPT가 사용되지 않은 것처럼 복구할 수 있습니다(프로토콜 인다우트 기간 중에 장애가 발생한 경우).

메시지의 안전성

IBM MQ 분산 큐 관리를 통해 메시지가 올바르게 전달되도록 할 수 있습니다. 두 채널 사이에 MQIPT가 존재하는 경우에도 해당됩니다. MQIPT는 메시지 데이터를 저장하거나 올바른 메시지 전달을 보증하는 동기점 프로시저에 참가하지 않습니다.

빠르고 비지속적인 IBM MQ 메시지를 사용할 때 MQIPT 라우트가 실패하거나 IBM MQ 메시지가 전송 중일 때 재시작되면 메시지가 유실될 수 있습니다. 라우트를 재시작하기 전에 MQIPT 라우트를 사용하는 모든 IBM MQ 채널이 비활성 상태인지 확인하십시오.

IBM MQ에서 메시지의 안전성에 대한 자세한 정보는 메시지의 안전성을 참조하십시오.

다중 인스턴스 큐 관리자 및 고가용성

MQIPT는 고가용성 환경에서 다중 인스턴스 큐 관리자와 함께 사용할 수 있습니다.

MQIPT에는 지속적 상태가 없기 때문에 MQIPT를 다른 시스템으로 장애 복구하는 경우 이점이 없습니다. 대신 동일한 `mqipt.conf` 구성 파일을 사용하는 다중 MQIPT 인스턴스가 서로 다른 시스템에서 실행되도록 하십시오. 고가용성을 위한 각각의 MQIPT 인스턴스를 모니터하고 필요한 경우 인스턴스를 다시 시작하십시오(동일한 시스템에서). 이 경우 연결을 라우팅하기 위해 사용할 수 있는 동일한 MQIPT 인스턴스가 제공됩니다. 그런 다음 IBM MQ에서 연결을 MQIPT로 라우팅할 수 있으며 MQIPT에서 해당 연결을 목적지 큐 관리자로 전달할 수 있는지 확인해야 합니다.

아웃바운드 IBM MQ 채널은 다양한 방법으로 사용 가능한 MQIPT 인스턴스로 전달할 수 있습니다. 예를 들면 다음과 같습니다.

- WebSphere Edge Components 제품의 IBM Network Dispatcher 와 같은 로드 밸런서 또는 고가용성 라우터를 사용하십시오.
- 쉼표로 구분된 목록을 사용하여 IBM MQ 채널 정의에서 복수의 연결 이름을 지정합니다. 이 경우 IBM MQ는 사용 가능한 MQIPT 인스턴스를 찾을 때까지 차례로 각각의 MQIPT 주소에 연결하려고 시도합니다.

또한 MQIPT에서 목적지 큐 관리자로의 연결도 전달해야 합니다. 고가용성 구성을 통해 목적지 큐 관리자에 대한 IP 주소 장애 복구가 보장된 경우 별도의 MQIPT 구성이 필요하지 않습니다. **Destination** 라우트 특성에 목적지 IP 주소를 지정하고 장애 복구 조작에서 해당 큐 관리자의 IP 주소를 이동할 수 있도록 허용하십시오.

하지만 장애 복구 이후에 큐 관리자의 IP 주소가 변경되는 경우 MQIPT에서 연결을 올바른 목적지로 전달하도록 조정해야 합니다. 이 작업은 다음 방법 중 하나를 사용하여 수행할 수 있습니다.

- 액세스할 수 있는 IP 주소 및 포트 번호를 검사하는 라우팅 엑시트를 작성한 후 각각의 연결에 대한 라우트 목적지를 대체하십시오. MQIPT에서는 몇 가지 샘플 라우팅 엑시트가 제공됩니다. 이러한 엑시트를 이 용도로 채택할 수 있습니다.
- 고가용성 로드 밸런서를 사용하여 연결을 경로 재지정하십시오.
- 큐 관리자를 실행할 수 있는 각각의 IP 주소 및 포트별로 하나씩, 복수의 MQIPT 라우트를 정의하십시오. 그런 다음 IBM MQ 연결을 다양한 MQIPT 라우트로 전달하십시오(예: 아웃바운드 채널의 연결 이름에서 쉼표로 구분된 목록으로 모든 라우트 IP 주소 및 포트 번호 나열).

또한 네트워크 경로에서 모든 엔드-투-엔드 컴포넌트를 성능 조정하는 것이 중요합니다.

1. 사용 불가능한 시스템에 대한 연결을 시도하는 경우 처음으로 사용 가능한 대상으로 재연결 시도가 이동할 수 있도록 즉시 실패해야 합니다.

   MQIPT SSL 라우트의 경우 사용 불가능한 대상에 대한 연결이 즉시 실패하도록 **SSLClientConnectTimeout** 라우트 특성을 성능 조정하십시오. IBM MQ 성능 조정 매개변수에 대한 자세한 정보는 IBM MQ 문서를 참조하십시오. 또한 운영 체제의 TCP/IP 성능 조정에 대한 자세한 정보는 운영 체제 문서를 참조하십시오. 모든 경우에 실패한 연결 시도는 신속하게 네트워크 실패(예: TCP 재설정 패킷)를 리턴하거나 과도한 지연 없이 제한시간이 초과되어야 합니다.

2. 실패한 시스템에 대한 활성 연결은 새 연결을 설정할 수 있도록 즉시 연결을 끊어야 합니다.

   또한 연결에서 MQIPT를 사용 중인 경우 장애 복구의 영향도 고려해야 합니다. 장애 복구 중에는 네트워크 연결이 끊어질 수 있습니다. 클라이언트 애플리케이션의 경우 IBM MQ 자동 클라이언트 재연결 기능을 사용하여 끊어진 연결을 재설정할 수 있습니다. 메시지 채널의 경우 채널이 즉시 재연결되도록 짧은 재시도 간격을 지정할 수 있습니다. 자동 클라이언트 재연결 및 메시지 채널 재시도 구성에 대한 자세한 정보는 IBM MQ 문서를 참조하십시오.

IBM MQ Console 및 REST API

IBM MQ Console 및 REST API 를 사용하여 IBM MQ를 관리하고 HTTP를 통해 메시징 조작을 수행할 수 있습니다.

- IBM MQ Console 를 사용하여 웹 브라우저에서 기본 관리 태스크를 수행할 수 있습니다. 자세한 정보는 IBM MQ Console를 사용한 관리를 참조하십시오.
- administrative REST API 를 사용하여 큐 관리자 및 큐, Managed File Transfer 에이전트 및 전송과 같은 IBM MQ 오브젝트를 관리할 수 있습니다. 자세한 정보는 REST API를 사용하여 관리를 참조하십시오.
- messaging REST API 를 사용하여 단순 지점간 및 공개 메시징을 수행할 수 있습니다. 자세한 정보는 REST API를 사용한 메시징을 참조하십시오.

설치 옵션

IBM MQ Console 및 REST API 는 mqweb라고 하는 WebSphere Liberty 서버에서 실행됩니다. IBM MQ 9.3.5부터는 IBM MQ 설치에서 선택적 구성요소로 또는 독립형 IBM MQ Web Server 설치로 mqweb 서버를 설치할 수 있습니다.

Linux V 9.4.0 독립형 IBM MQ Web Server 설치

IBM MQ 9.4.0부터 mqweb 서버는 IBM MQ Web Server의 독립형 설치에서 실행될 수 있습니다. 독립형 IBM MQ Web Server 설치를 사용하면 IBM MQ 설치와 별도의 시스템에 mqweb 서버를 설치하고 실행할 수 있습니다. 독립형 IBM MQ Web Server 를 설치하면 mqweb 서버를 실행하도록 선택하는 시스템 및 시스템 수에 대해 더 큰 유연성을 제공합니다. 필요한 경우, mqweb 서버의 여러 인스턴스를 다른 시스템에서 실행하여 필요한 확장성 및 가용성을 제공할 수 있습니다.

IBM MQ 인타이틀먼트를 구매한 경우 독립형 IBM MQ Web Server의 필요한 수만큼의 사본을 설치할 수 있습니다. IBM MQ Web Server 설치는 구매한 IBM MQ 인타이틀먼트로 계수되지 않습니다. IBM MQ 라이센싱에 대한 자세한 정보는 IBM MQ 라이센스 정보를 참조하십시오.

다음 제한사항은 독립형 IBM MQ Web Server 설치에 적용됩니다.

- IBM MQ Console 는 리모트 큐 관리자만 관리하는 데 사용할 수 있습니다.
- messaging REST API 는 리모트 큐 관리자에서만 사용할 수 있습니다.
- administrative REST API 를 사용할 수 없습니다.

독립형 IBM MQ Web Server 는 Linux 플랫폼에서만 지원됩니다.

독립형 IBM MQ Web Server설치에 대한 자세한 정보는 독립형 IBM MQ Web Server설치를 참조하십시오.

IBM MQ 설치의 선택적 구성요소

IBM MQ Console 및 REST API 구성요소를 IBM MQ 설치의 일부로 설치하도록 선택할 수 있습니다.

모든 IBM MQ Console 및 REST API 기능은 mqweb 서버가 IBM MQ 설치에서 실행될 때 사용 가능합니다.

- IBM MQ Console 를 사용하여 로컬 및 리모트 큐 관리자를 관리할 수 있습니다.
- messaging REST API 는 로컬 및 리모트 큐 관리자와 함께 사용할 수 있습니다.
- administrative REST API 를 사용하여 로컬 및 리모트 큐 관리자를 관리할 수 있습니다.

IBM MQ Console 및 REST API 구성요소를 사용하려면 IBM MQ 설치의 일부로 다음 구성요소를 설치하십시오.

- AIX AIX에서 `mqm.web.rte` 파일 세트를 설치하십시오.

- IBM i IBM i에 WEB 컴포넌트를 설치하십시오.
- Linux Linux에서 MQSeriesWeb 구성요소를 설치하십시오.
- Windows Windows에서 Web Administration 기능을 설치하십시오.
- z/OS z/OS에서 IBM MQ for z/OS UNIX System Services Web Components 기능을 설치하십시오.

주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

07326
서울특별시 영등포구
국제금융로 10, 3IFC
한국 아이.비.엠 주식회사
U.S.A.

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

Intellectual Property Licensing
2-31 Roppongi 3-chome, Minato-Ku
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 명시적 또는 묵시적인 일체의 보증 없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 변경된 사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

서울특별시 영등포구
서울특별시 강남구 도곡동 467-12,
군인공제회관빌딩
한국 아이.비.엠 주식회사
U.S.A.

이러한 정보는 해당 조건(예를 들면, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이센스가 부여된 프로그램 및 프로그램에 대해 사용 가능한 모든 라이센스가 부여된 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 단계의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을

통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM이 제시하는 방향 또는 의도에 관한 모든 언급은 특별한 통지 없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이센스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 애플리케이션 프로그래밍 인터페이스(API)에 부합하는 애플리케이션을 개발, 사용, 판매 또는 배포할 목적으로 IBM에 추가 비용을 지불하지 않고 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 진술하지 않습니다.

이 정보를 소프트카피로 확인하는 경우에는 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보는 본 프로그램과 함께 사용하기 위한 응용프로그램 소프트웨어 작성을 돕기 위해 제공됩니다.

이 책에는 고객이 IBM MQ의 서비스를 얻기 위해 프로그램을 작성할 수 있도록 하는 의도된 프로그래밍 인터페이스에 대한 정보가 들어 있습니다.

그러나 본 정보에는 진단, 수정 및 성능 조정 정보도 포함되어 있습니다. 진단, 수정 및 성능 조정 정보는 응용프로그램 소프트웨어의 디버그를 돕기 위해 제공된 것입니다.

중요사항: 이 진단, 수정 및 튜닝 정보는 변경될 수 있으므로 프로그래밍 인터페이스로 사용하지 마십시오.

상표

IBM, IBM 로고, ibm.com®는 전세계 여러 국가에 등록된 IBM Corporation의 상표입니다. 현재 IBM 상표 목록은 웹 "저작권 및 상표 정보"(www.ibm.com/legal/copytrade.shtml) 에 있습니다. 기타 제품 및 서비스 이름은 IBM 또는 타사의 상표입니다.

Microsoft 및 Windows는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 등록상표입니다.

이 제품에는 Eclipse 프로젝트 (https://www.eclipse.org/) 에서 개발한 소프트웨어가 포함되어 있습니다.

Java 및 모든 Java 기반 상표와 로고는 Oracle 및/또는 그 계열사의 상표 또는 등록상표입니다.

부품 번호:

(1P) P/N: